Expression Blend 4 с примерами на C# **ДЛЯ ПРОФЕССИОНАЛОВ**

Pro Expression Blend 4

Andrew Troelsen

Apress®

Apress, Inc.
233 Spring Street
New York, NY 10013
www.apress.com

Expression Blend 4 с примерами на C# для профессионалов

Эндрю Троелсен

ВИЛЬЯМС

Москва · Санкт-Петербург · Киев
2011

ББК 32.973.26-018.2.75
Т70
УДК 681.3.07

Издательский дом "Вильямс"
Зав. редакцией *С.Н. Тригуб*
Перевод с английского и редакция *И.В. Берштейна*
По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Троелсен, Эндрю.
Т70 Expression Blend 4 с примерами на C# для профессионалов. : Пер. с англ. — М. : ООО "И.Д. Вильямс", 2011. — 368 с. : ил. — Парал. тит. англ.

ISBN 978-5-8459-1756-0 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Эндрю Троелсен

Expression Blend 4 с примерами на C# для профессионалов

Литературный редактор *И.А. Попова*
Верстка *Л.В. Чернокозинская*
Художественный редактор *Е.П. Дынник*
Корректор *Л.А. Гордиенко*

Подписано в печать 25.08.2011. Формат 70x100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 23,0. Уч.-изд. л. 24,4.
Тираж 1500 экз. Заказ № 0000.

Отпечатано с готовых диапозитивов в ГУП "Типография «Наука»"
199034, Санкт-Петербург, 9-я линия В. О., 12.

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-1756-0 (рус.)
ISBN 978-1430-23377-0 (англ.)

Оглавление

Содержание

Об авторе

Эндрю Троелсен с нежностью вспоминает свой первый компьютер Atari 400, оснащенный устройством записи данных на магнитную ленту и подключавшийся к черно-белому телевизору, служившему в качестве монитора. Добрые родители разрешали Эндрю держать этот компьютер в своей комнате. Не в меньшей степени он благодарен некогда издававшемуся журналу *Compute!*, математической лингвистике, в области которой у него имеется степень бакалавра, а также формальному санскриту, который он изучал в течение трех лет. Все эти факторы оказали заметное влияние на его профессиональную карьеру.

В настоящее время Эндрю работает в Intertech (`www.intertech.com`) — учебном и консультационном центре по .NET и Java. Он является автором целого ряда книг, в том числе *Developer's Workshop to COM and ATL 3.0* (издательство Wordware Publishing, 2000 г.; *COM and .NET Interoperability* (издательство Apress, 2002 г.), а также *Pro C# 2010 and the .NET 4.0 Platform* (издательство Apress, 2010 г.; в русском переводе книга вышла под названием "Язык программирования C# 2010 и платформа .NET 4.0" в ИД "Вильямс", 2011 г.; ISBN 978-5-8459-1682-2).

О научном рецензенте

Энди Ольсен работает независимым консультантом и инструктором в Великобритании. Энди работает на платформе .NET, начиная с ее первой бета-версии, а также на платформах WPF и Silverlight с тех пор, как они впервые стали доступными на настольных компьютерах и в браузерах. Энди проживает вместе со своей женой и детьми в приморском городе Суонси. Он любит прогулки по морскому берегу (с регулярными перерывами на чашечку кофе), увлекается катанием на лыжах и с интересом наблюдает за лебедями и орликами. С ним можно связаться по адресу `andyo@olsensoft.com`.

Посвящение

Будущему маленькому Троелю.

Благодарности

Мне посчастливилось работать с сотрудниками издательства Apress почти десять лет, и должен сказать, что каждая написанная мною книга выгодно отличалась своими полиграфическими качествами благодаря их труду.

Огромная благодарность выражается Энди Ольсену, Эвану Бакингему, Дебре Келли и всей коллегии литературных редакторов за приведение моих беспорядочных мыслей, запечатленных в рукописи, в связный и удобочитаемый текст. С глубокой признательностью и большой надеждой уповаю на дальнейшее сотрудничество со всеми ними.

Введение

Признания знатока XAML

Возможно, моя история покажется многим очень знакомой. Начав программировать сначала на платформе Windows Presentation Foundation (WPF), а затем на платформе Silverlight 2.0, я был одним из тех программирующих, которые считали, что для построения собственных приложений им придется набирать всю необходимую разметку веб-страниц вручную[1]. Ведь набор исходного кода составляет немалую долю всего объема работы, выполняемого любым разработчиком программного обеспечения. Все было хорошо, когда я компоновал страницы, разбивая их на сетки, блочные панели и элементы управления. Но как только я вник в этот предмет глубже, начав работать с анимацией, шаблонами данных, шаблонами элементов управления и специальной графикой, то сразу же осознал, насколько непродуктивно работал.

Мне было известно о том, что корпорация Microsoft разработала инструментальное средство Expression Blend, но я считал, что оно предназначено исключительно для *художников-оформителей*. А поскольку я совсем не относил себя к числу художников-оформителей, то избегал интегрированной среды разработки Expression Blend как чумы. Эта среда была совершенно не похожа на Visual Studio, а организация редактора свойств в ней не представляла для меня особого интереса. К тому же в первых версиях Expression Blend отсутствовали редакторы кода, но теперь это положение уже исправлено.

И вот в один прекрасный день я все же решил заняться Expression Blend, чтобы реально оценить ее возможности как инструментального средства разработчика. Как ни странно, она мне понравилась. И чем больше времени я с ней работал, тем больше она мне нравилось. Ныне я просто не могу себе представить разработку проектов на платформе WPF, Silverlight или Windows Phone 7 без Expression Blend. Это все равно, что набирать исходный код одной рукой, держа другую за спиной.

Сделанным открытием я решил поделиться с коллегами-программистами, но обнаружил их в том же положении, в каком сам находился первоначально. Они считали среду Expression Blend IDE слишком сложной, а кроме того, в Visual Studio 2010 теперь основательно поддерживается разработка XAML-ориентированных приложений. Так стоит ли вообще обращать внимание на другие инструментальные средства вроде Expression Blend? Разумеется, в Visual Studio 2010 сделан шаг в правильном направлении, но не следует забывать о том, что Expression Blend заметно превосходит Visual Studio 2010 по производительности. В частности, средствами Expression Blend можно делать следующее.

- Формировать шаблон элемента управления из векторной графики с помощью единственной команды, выбираемой из меню.
- Создавать сложные виды анимации в интегрированном редакторе.

[1] Следует напомнить, что в Visual Studio 2005 реальная поддержка WPF вообще отсутствовала, не считая экспериментальной технологии для предварительного просмотра поверхности визуального конструктора.

- Визуально редактировать шаблоны данных.
- Внедрять визуальные подсказки в специальные шаблоны, используя интегрированные графические редакторы.
- Составлять представления данных с перечислением подробностей, выполняя лишь две операции мышью.
- Создавать прототипы приложений на платформах WPF или Silverlight и регистрировать отзывы клиентов в реальном масштабе времени[2].

Я написал эту книгу с тем, чтобы помочь своим коллегам по разработке программного обеспечения освоить интегрированную среду Expression Blend IDE. Откровенно говоря, я считаю это инструментальное средство очень важной составляющей любого процесса разработки приложений WPF, Silverlight или Windows Phone 7 на промышленном уровне. И если вы освоите эту среду, то будете считать точно так же. Разумеется, никто не оспаривает важность и насущную необходимость среды Visual Studio 2010 для отладки, тестирования и расширения фрагментов исходного кода современных приложений под Windows. Как будет показано в этой книге, Expression Blend служит в качестве *дополнения*, а не *замены* интегрированной среды разработки Visual Studio 2010.

Эта книга не для программирующих

Все написанные мною ранее книги были посвящены программированию и содержали немало примеров исходного кода. Я всегда писал книги по информационным технологиям именно таким образом, уделяя в них меньше всего места описанию команд меню в интегрированной среде разработки, мастеров и прочих инструментальных средств. На мой взгляд, при изучении нового языка программирования или платформы так или иначе приходится разрабатывать и набирать исходный код. И если читатель легко понимает исходный код, то разобраться с инструментальными средствами визуальной разработки для него не составит большого труда.

В этой книге я придерживаюсь совершенно иного подхода, хотя для меня лично он не совсем обычен. В частности, в этой книге основное внимание уделяется описанию команд меню, интегрированных редакторов, диалоговых окон и вариантов настройки интегрированной среды разработки. И поэтому материал, излагаемый в книге, обильно дополняется рисунками в виде моментальных снимков экранов, а не примерами кода.

На самом деле, если попытаться собрать воедино весь исходный код C#, приведенный в этой книге, то он занял бы не больше десяти печатных страниц. В то же время примеры разметки в коде XAML заняли бы много больше места.

Таким образом, читатель не найдет подробное описание программных средств для построения приложений WPF, Silverlight или Windows Phone 7, как, впрочем, и для создания свойств специальных зависимостей или формирования всплывающих перенаправленных событий. В ней не рассматриваются принципы переопределения виртуальных методов при построении классов, расширяющих класс `UIElement`. Все эти и многие другие вопросы оставлены без должного внимания в данной книге, несмотря на их важность[3].

---

[2] Средствами SketchFlow при наличии копии Expression Studio Ultimate (см. главу 8).

[3] Детальное рассмотрение подобных вопросов можно найти в последних изданиях моих книг по C# или VB, где немало глав посвящено программированию на платформе WPF, а во многих случаях — на платформах Silverlight и Windows Phone 7.

Но эта книга для художников-оформителей

В этой книге отсутствуют примеры сложного кода, а следовательно, она адресована главным образом художникам-оформителям, чтобы помочь им научиться применять Expression Blend при разработке пользовательских интерфейсов на высоком профессиональном уровне для проектов, создаваемых на платформах WPF, Silverlight или Windows Phone 7. Как будет показано далее, средствами Expression Blend можно создавать сложную векторную графику и экспортировать данные изображений в формате, пригодном для использования в среде Expression Blend. Такой подход позволяет убить сразу двух зайцев, существенно упрощая задачу программистов и дизайнеров по созданию интерактивной графики.

Если вы занимаетесь графическим оформлением, то обнаружите в этой книге очень мало несложного кода C#, который совсем не обязательно набирать, поскольку его можно загрузить с веб-сайта издательства Apress, как поясняется далее. И наконец, если графическое оформление и дизайн являются вашей профессией, не судите слишком строго скромные познания автора в данной области.

Краткий обзор содержания книги

Книга состоит из восьми глав, которые было бы идеально прочитать по порядку, поскольку материал каждой последующей главы основывается на предыдущей. Ниже приводится краткое содержание книги по отдельным ее главам.

Глава 1. Общее представление о среде Expression Blend IDE

Эта глава служит основанием для остальных глав книги. В ней делается краткий обзор основных компонентов и особенностей интегрированной среды разработки Expression Blend IDE. Из нее вы узнаете о монтажном столе; интегрированных редакторах исходного кода, включая и XAML; назначении нескольких ключевых панелей, в том числе Objects and Timeline, Properties Tools, и пр. Кроме того, рассматриваются типы проектов, поддерживаемых в Expression Blend, а также назначение библиотеки ресурсов. Все эти важные компоненты будут рассмотрены на одном из характерных примеров проектов, входящих в состав Expression Blend.

Глава 2. Векторная графика и ресурсы объектов

Разработчики, имеющие опыт работы на платформе WPF или Silverlight, хорошо знают, что графика является важной составляющей любого проекта. Из этой главы вы узнаете, как пользоваться встроенными в Expression Blend инструментами графического оформления, в том числе Pen и Pencil, различными “ресурсами форм”, редактором кистей и другими средствами, представляющими интерес. Кроме того, поясняется, как пользоваться Expression Blend для графических преобразований элементов пользовательского интерфейса, создания трехмерной графики и манипулирования ею, определения и упаковки ресурсов объектов, которые, по существу, называются большими двоичными объектами XAML, неоднократно используемыми в проектах.

Глава 3. Редактор анимации

Анимация также является важной составляющей проектов на платформах WPF и Silverlight и широко используется при построении шаблонов элементов управления, шаблонов данных и других “визуально привлекательных” форм. Из этой главы вы узнаете о редакторе анимации, интегрированном в Expression Blend, а также о том, как определяются и настраиваются раскадровки и ключевые кадры анимации, каким образом регулируется темп анимации и применяются различные физические эффекты (упругос-

ти, отскакивания, притягивания и т.п.) в цикле анимации с помощью эффектов инерционности движения. Кроме того, дается первоначальное представление о встроенных в Expression Blend объектах поведения, более подробно рассматриваемых в главе 4.

Глава 4. Элементы управления, виды компоновки и объекты поведения

Цель данной главы — продемонстрировать ряд способов и средств, применяемых для работы с элементами управления пользовательского интерфейса в среде Expression Blend IDE. Вы узнаете, что собой представляет модель содержимого элементов управления, как настраиваются и заполняются сложными списками элементы управления типа `ListBox` и как фиксируются данные, вводимые пользователем в элементе управления типа `InkCanvas`, а также ознакомитесь с назначением элементов управления из прикладного интерфейса WPF Document API. Кроме того, будут представлены различные объекты поведения. Как следует из этой и других глав, объект поведения позволяет наделить элементы пользовательского интерфейса сложными функциями, проявляющимися во время выполнения, причем сделать это визуально, а не программно.

Глава 5. Стили, шаблоны и классы `UserControl`

При создании приложений WPF и Silverlight механизм стилевого оформления обеспечивает сходный внешний вид связанных вместе элементов управления пользовательского интерфейса. Из этой главы вы сначала узнаете, каким образом в среде Expression Blend IDE упрощается создание стилей и манипулирование ими, а затем ознакомитесь с назначением шаблонов элементов управления, в которых понятие стиля применяется на более высоком уровне. Как следует из этой главы, при определении специального шаблона для элемента управления можно полностью заменить исходный внешний вид этого элемента на собственный, задав специальные инструкции для его визуализации. И в заключение главы рассматривается быстрый способ создания в среде Expression Blend IDE новых специальных объектов класса `UserControl` одним щелчком кнопкой мыши.

Глава 6. Способы привязки данных в Expression Blend

В данной главе рассматриваются различные инструментальные средства Expression Blend IDE, упрощающие выполнение операций привязки данных. Из этой главы вы узнаете, каким образом настраивается привязка данных одних элементов управления к другим, как привязываются коллекции специальных бизнес-объектов и как привязываются данные, содержащиеся в XML-документах. Кроме того, будет показано, как в среде Expression Blend IDE создаются специальные шаблоны данных, позволяющие организовать стилевое оформление операций привязки данных для отображения результатов выполнения этих операций в приложении.

Глава 7. Разработка приложений на платформе Windows Phone 7

Когда писались эти строки, корпорация Microsoft официально объявила о выпуске серии мобильных устройств, работающих под ОС Windows Phone 7, и поддерживающего их набора инструментальных средств разработки программного обеспечения Windows Phone 7 Software Development Kit (SDK). Из этой главы вы сначала узнаете, каким образом загружаются и устанавливаются инструментальные средства, необходимые для построения приложений на платформе Windows Phone 7 в среде Expression Blend (а также Visual Studio 2010), а затем ознакомитесь с рядом новых средств, включая панель Device, интегрированный эмулятор Windows Phone и средства построения приложений панорамного и сводного типа. Вам будет приятно узнать, что все способы и средства,

усвоенные вами из материала шести предыдущих глав, можно применять непосредственно при создании проектов на платформе Windows Phone 7.

Глава 8. Создание прототипов средствами SketchFlow

По иронии судьбы завершающая глава посвящена тому, с чего обычно начинается разработка нового проекта! В частности, из этой главы вы узнаете, как создаются прототипы приложений на платформах WPF и Silverlight средствами SketchFlow и каким образом с помощью компонента SketchFlow из пакета Expression Studio Ultimate Edition можно оперативно сымитировать прототипы в реальном масштабе времени, фиксируя по ходу дела отзывы заинтересованных лиц. Кроме того, вы ознакомитесь с различными средствами Expression Blend, связанными с компонентом SketchFlow, включая панель Map, стили эскизов, средство просмотра пользовательских аннотаций и упрощенный редактор анимации. В этой главе показывается также, каким образом созданный в SketchFlow прототип преобразуется в реальный проект на платформе WPF или Silverlight.

Получение примеров проектов

При изучении материала каждой главы вам предоставляется возможность освоить на практике различные средства Expression Blend IDE, прорабатывая разнообразные примеры проектов. Трудно даже переоценить важность построения (и расширения) этих примеров проектов по ходу чтения книги. В пользу такого подхода к изучению материала этой книги свидетельствует тот факт, что осваивать среду Expression Blend IDE лучше всего, *делая* что-нибудь конкретное, а не просто *читая* книгу и рассматривая иллюстрации в виде моментальных снимков экрана.

Каждый пример проекта можно загрузить с веб-сайта издательства Apress по адресу `http://www.apress.com/9781430233770`, где находится архивный файл с расширением `*.zip` со всеми упоминаемыми в этой книге проектами. После распаковки содержимого этого архива отдельные проекты можно найти в папках соответствующих глав.

В тексте книги ссылки на примеры проектов указываются в примечаниях "Исходный код", где напоминается о том, что рассматриваемые примеры проектов могут быть загружены в среду Expression Blend (или в Visual Studio 2010) для дальнейшего изучения и видоизменения, как показано ниже.

Исходный код. Это примечание, в котором делается ссылка на конкретную папку в архиве примеров проектов к данной книге.

Для того чтобы открыть упоминаемый проект в среде Expression Blend, выберите команду File⇨Open Project/Solution (Файл⇨Открыть проект/решение) из главного меню и перейдите к соответствующему файлу с расширением `*.sln` в указанной папке архива примеров проектов к данной книге.

Получение обновлений этой книги

Читая эту книгу, вы можете обнаружить в ней случайные ошибки или опечатки, хотя я надеюсь, что они в ней все же отсутствуют. В этом случае будьте снисходительны к автору, поскольку он тоже человек и может ошибаться. На веб-сайте издательства Apress по адресу `http://www.apress.com/9781430233770` доступен перечень обнаруженных ошибок, опечаток и соответствующих исправлений.[4] Кроме того, вы можете сообщить автору о тех ошибках, которые обнаружили сами и которые не были исправлены.

---

[4] Только в исходном варианте на английском языке. — *Примеч. ред.*

Как связаться с автором

Если при чтении книги у вас возникнут вопросы относительно примеров рассматриваемых в ней проектов с целью прояснить их ли же просто поделиться своими соображениями по поводу среды Expression Blend IDE, напишите автору по приведенному ниже адресу электронной почты. Но для того чтобы ваше сообщение не попало в почтовый ящик с "макулатурной почтой", непременно укажите `Blend Book` (Книга по Expression Blend) в теме своего сообщения.

`atroelsen@intertech.com`

Я постараюсь ответить вам, но имейте в виду, что это будет не сразу, поскольку я такой же занятой человек, как и все остальные. Если вы не получите от меня ответ в течение одной или двух недель, знайте, что я не ответил вам не из-за нерадивости или нежелания общаться, а просто в силу большой занятости, а возможно, и отсутствия на месте.

Так или иначе благодарю вас за то, что вы приобрели эту книгу или хотя бы бегло просмотрели ее в книжном магазине, решая, стоит ли вообще покупать ее. Надеюсь, чтение этой книги будет для вас интересным и полезным с точки зрения практического применения полученных знаний.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш веб-сайт и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши координаты:

E-mail: `info@williamspublishing.com`

WWW: `http://www.williamspublishing.com`

Информация для писем из:

России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1

Украины: 03150, Киев, а/я 152

ГЛАВА 1

Общее представление о среде Expression Blend IDE

Эта глава посвящена исследованию основных компонентов интегрированной среды разработки Expression Blend IDE. Сначала в ней будет сделан краткий обзор каждого члена семейства программных продуктов Microsoft Expression и показано их место в системе Windows Presentation Foundation (WPF) и программной среде разработки Silverlight. Затем будут рассмотрены различные шаблоны проектов в Expression Blend, главные рабочие области (монтажный стол, панели Objects and Timeline и Properties и прочее), а также особенности взаимодействия Expression Blend и Microsoft Visual Studio 2010. И в заключение будет показано, каким образом данная интегрированная среда разработки настраивается с учетом личных предпочтений пользователя.

Семейство программных продуктов Microsoft Expression

Microsoft Expression как семейство программных продуктов было впервые продемонстрировано на Конференции профессиональных разработчиков (PDC — Professional Developers Conference) в 2005 году, но лишь в 2007 году корпорация Microsoft сделала общедоступным первый выпуск инструментальных средств из этого семейства. Семейство программных продуктов Expression представляет собой ряд приложений, предназначенных для тех пользователей, которые профессионально занимаются графическим оформлением, но этими продуктами все чаще стали пользоваться и разработчики программного обеспечения.

На момент написания этой книги семейство Expression состояло из четырех программных продуктов (Expression Web, Expression Encoder, Expression Design и Expression Blend), которые можно было приобрести вместе с пакетом программ Microsoft Expression Studio Ultimate[1]. Вам, вероятно, будет приятно узнать, что если у вас или вашей организации имеется подписка на MSDN (собрание документов корпорации Microsoft, со-

[1] Этих продуктов получается пять, если считать и Expression Media, права на который недавно были приобретены компанией Phase One A/S. Программный продукт Expression Media представляет собой коммерчески доступную программу каталогизации цифрового управления ресурсами (DAM) в операционных системах Microsoft Windows и Mac OS X.

держащее сведения обо всех ее разработках), то Expression Studio Ultimate входит в ваш текущий пакет программ. Но даже если у вас нет законной подписки на MSDN, вам все равно будет приятно узнать, что вы можете загрузить пробную версию Expression Studio Ultimate, действующую в течение 60 дней, по следующему адресу[2]:

www.microsoft.com/expression/try-it

Строго говоря, для изучения материала этой книги требуется только копия Expression Blend. Но если вы стремитесь к тому, чтобы научиться внедрять сложную векторную графику в приложения WPF или Silverlight (подробнее об этом — в главе 2), настоятельно рекомендуется установить также копию Expression Design. Что же касается остальных членов семейства программных продуктов Microsoft Expression, то в настоящей книге они не упоминаются. Тем не менее вы сможете изучить их самостоятельно, а приведенный ниже краткий обзор поможет сориентироваться в назначении и функциональных возможностях каждого члена семейства Microsoft Expression.

Назначение Expression Web

Инструментальное средство Expression Web позволяет создавать готовые для эксплуатации и стандартизированные веб-сайты в режиме визуальной разработки. Несмотря на то что это инструментальное средство веб-разработки от корпорации Microsoft, оно *не* накладывает никаких ограничений на применение только на платформе ASP.NET или ASP.NET AJAX, хотя поддержка платформы .NET реализована в Expression Web отлично. По желанию можете воспользоваться интегрированными редакторами страниц и исходного кода для создания веб-сайтов средствами PHP, HTML/XHTML, XML/XSLT, CSS, JavaScript, а также с помощью компонентов Adobe Flash и Windows Media.

В состав Expression Web входит также сопутствующий программный продукт SuperPreview. Этот компонент Expression Web существенно упрощает тестирование создаваемых веб-сайтов в нескольких наиболее распространенных браузерах, работающих как в Windows, так и в Mac OS. Если у вас имеется некоторый опыт веб-разработки, то вам, вероятно, известно, каких нервов стоит обеспечение правильного функционирования веб-страниц в разных программных средах. Применяя Expression Web и SuperPreview, вы получаете в свое распоряжение солидный набор инструментальных средств, помогающих благополучно справиться с подобной задачей и сберечь свои нервы.

Назначение Expression Encoder

В примерах, рассматриваемых в этой книге, Expression Encoder не применяется, тем не менее это инструментальное средство следует иметь в виду, поскольку оно предоставляет удобную платформу для импорта, редактирования и усовершенствования видеоматериалов, кодированных в самых разных форматах файлов, включая AVI, WMV, WMA, QuickTime MOV (если установлен проигрыватель QuickTime), MPEG, VC-1 и H.264.

Так, например, с помощью Expression Encoder можно создать на профессиональном уровне учебный видеоматериал, настроенный на воспроизведение в потоковом режиме в приложении Silverlight или WPF. Кроме того, Expression Encoder можно использовать для создания мультимедийных средств, плавно интегрируемых в приложения Silverlight или WPF посредством закладок и специально настраиваемых обложек.

---

[2] На начальной странице веб-сайта, посвященного семейству программных продуктов Microsoft Expression (`www.microsoft.com/expression`), предоставляются ссылки на богатые ресурсы, доступные в Интернете. По этим ссылками вы сможете найти немало учебных видеоматериалов, подробно разбираемых примеров применения, предварительных обзоров новых технологий и много другой полезной информации. Непременно посетите этот сайт, заслуживающий того, чтобы сделать на него закладку в браузере.

Назначение Expression Design

Инструментальное средство Expression Design разработано корпорацией Microsoft с целью составить конкуренцию таким программным продуктам компании Adobe Systems, как Illustrator and Photoshop. (На самом деле в Expression Design и Expression Blend можно импортировать файлы изображений в форматах Illustrator и Photoshop — этих двух основных приложений для графического оформления.) По существу, инструментальное средство Expression Design позволяет художникам-оформителям создавать изысканные образцы векторной графики.

Как и следовало ожидать, Expression Design дает художникам-оформителям возможность сохранять результаты своих трудов в самых разных стандартных форматах файлов, включая PNG, JPEG, GIF, TIFF и пр. Но самое интересное, что Expression Design позволяет также сохранять графические данные в формате XAML для приложений WPF и Silverlight.

Вам, вероятно, известно, что расширяемый язык разметки приложений (XAML) основывается на синтаксисе языка XML для описания состояния графического или другого объекта на платформе .NET. Например, в приведенном ниже фрагменте кода разметки на языке XAML описывается внешний вид векторного изображения, представленного на рис. 1.1, а также на цветной вклейке. Данный пример показывает, как с помощью всего лишь нескольких строк кода XAML можно создать привлекательные эффекты падающей тени и сложного радиального градиента, заполняющего участок внутри круга.

```
<Ellipse x:Name="shadowCircle" HorizontalAlignment="Left"
       Height="117.5" Margin="99,58,0,0" Stroke="#FF0B17D6"
       VerticalAlignment="Top" Width="117.5"
       StrokeThickness="7">
  <Ellipse.Effect>
    <DropShadowEffect Color="#FFB7B8E0" BlurRadius="6" ShadowDepth="11"/>
  </Ellipse.Effect>
  <Ellipse.Fill>
    <RadialGradientBrush GradientOrigin="0.38,0.304">
      <GradientStop Color="#FF111EE0" Offset="0"/>
      <GradientStop Color="#FFC5C7E0" Offset="1"/>
      <GradientStop Color="#FF4750DA" Offset="0.526"/>
    </RadialGradientBrush>
  </Ellipse.Fill>
</Ellipse>
```

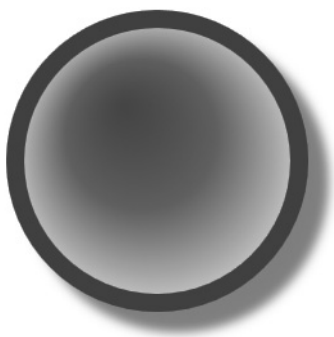


Рис. 1.1. Результат выполнения фрагмента кода XAML из примера графического оформления окружности

Предоставляя возможность сохранять векторную графику в формате XAML, Expression Design существенно упрощает разработчикам задачу внедрения профессионально оформленной графики в существующее приложение и ее взаимодействия с данными посредством кода. В частности, художник-оформитель может создать стилизованный двухмерный лабиринт для видеоигры. Сохранив эти графические данные в формате XAML, он может затем импортировать их в проект Expression Blend (или Visual Studio 2010) и дополнить стилизованной анимацией, поддержкой проверки местоположения курсора мыши и прочими средствами. О том, как это делается, речь пойдет в главе 2.

Назначение Expression Blend

А теперь о главном предмете этой книги! Expression Blend представляет собой компонент, предназначенный для разработки приложений WPF или Silverlight на промышленном уровне. Это инструментальное средство генерирует большой объем кода XAML, требующегося для прикладных программ. И хотя аналогичную разметку *можно* выполнить вручную, используя разнообразные средства разработки, начиная с текстового редактора WordPad и кончая интегрированной средой Visual Studio 2010, вы можете избавить себя от хронических судорог в кистях рук, воспользовавшись многословным характером синтаксиса XAML, основанного на языке XML.

Возможности Expression Blend выходят далеко за рамки относительно простой поддержки редактирования кода XAML в Visual Studio 2010, предоставляя развитые инструментальные средства для компоновки и настройки элементов управления, создания анимационных последовательностей, специальных стилей оформления и шаблонов, построения новых классов `UserControl`[3] из имеющейся векторной графики, визуальной разработки шаблонов данных, назначения различных режимов работы и визуальных состояний для элементов пользовательского интерфейса и выполнения многих других полезных операций.

Читая эту книгу, вы постепенно ознакомитесь с различными функциональными возможностями Expression Blend по созданию весьма выразительных пользовательских интерфейсов для приложений, разрабатываемых для веб (средствами Silverlight), настольных систем (средствами WPF) и мобильных устройств, работающих под Windows Phone 7 (как ни странно, но средствами Silverlight).

Expression Blend — это лишь одна сторона медали

Несмотря на то что в состав Expression Blend входит упрощенный редактор кода C# и VB, рассматриваемый далее в главе, вы вряд ли будете разрабатывать весь свой код на платформе .NET в подобной интегрированной среде в силу ее довольно ограниченных в этом отношении возможностей. В частности, в ней не поддерживается отладка кода. Правда, проект в Expression Blend имеет *тот же самый* формат, что и родственный ему проект в Visual Studio 2010. Следовательно, работу над новым проектом можно начать в Expression Blend с разработки пользовательского интерфейса, а затем открыть этот проект в Visual Studio 2010 для реализации, отладки, тестирования и компоновки сложного кода приложения.

[3] `UserControl` — это класс .NET, связанный с разметкой в формате XAML и представляющий ряд неоднократно применяемых элементов пользовательского интерфейса. О том, как элементы управления пользовательского интерфейса создаются в Expression Blend, речь пойдет в главе 5.

Как правило, подавляющая часть приложений WPF или Silverlight строится в каждой из упомянутых выше интегрированных сред в цикле их разработки[4]. Встроенный редактор кода и порядок взаимодействия Expression Blend с Visual Studio 2010 будут рассмотрены более подробно далее в главе.

Назначение SketchFlow

Помимо инструментальных средств, предоставляемых для построения изящно оформленных пользовательских интерфейсов, в состав Expression Blend входит набор инструментов, позволяющих оперативно создавать прототипы приложений средствами SketchFlow[5]. Этот компонент Expression Blend дает возможность быстро и эффективно имитировать и определять последовательность операций в пользовательском интерфейсе приложения, компоновку экранов и переход из одного состояния приложения в другое. На рис. 1.2 приведен проект SketchFlow, загруженный в интегрированную среду разработки Expression Blend IDE.

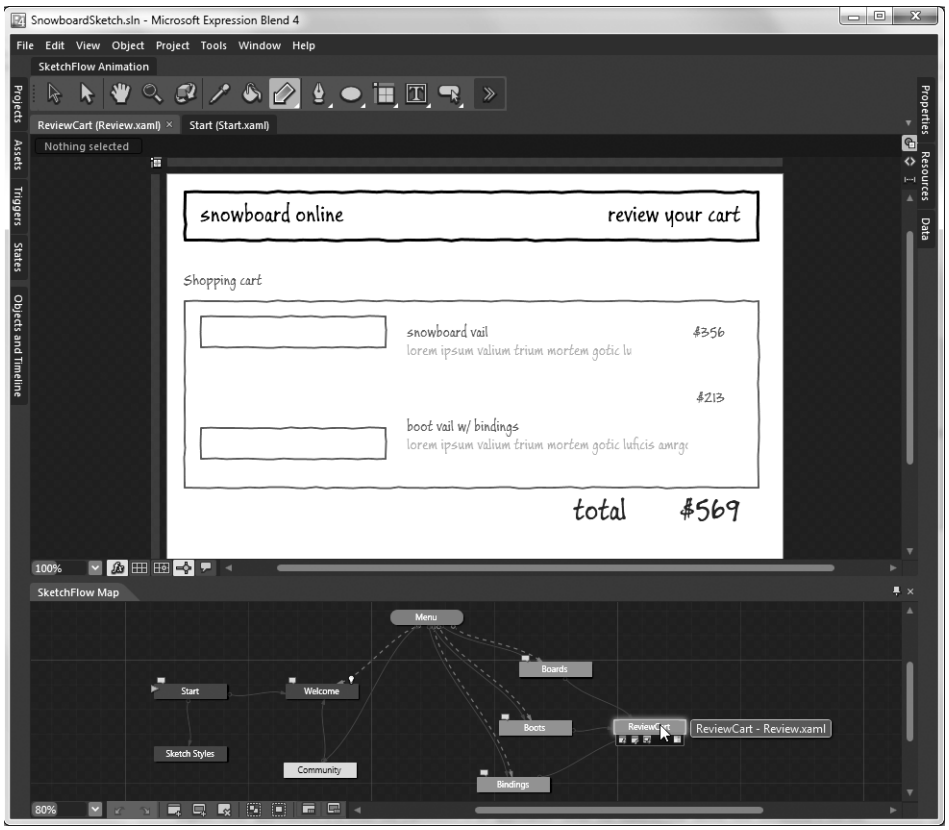


Рис. 1.2. Компонент SketchFlow позволяет оперативно создавать прототипы приложений WPF и Silverlight

[4] Проект, разрабатываемый в Expression Blend, можно интегрировать в систему управления версиями исходного кода на сервере Visual Studio Team Foundation Server.

[5] Компонент SketchFlow входит только в состав версии Expression Studio Ultimate Edition.

Компонент SketchFlow предназначен для применения на стадии создания прототипов приложений, и поэтому у вас появляется возможность построения имитаций пользовательского интерфейса, не особенно заботясь о мелких деталях, а также быстрой адаптации прототипов в реальном масштабе времени, учитывая отзывы клиента. Этот процесс упрощается в еще большей степени с помощью свободно распространяемого проигрывателя SketchFlow Player, который дает возможность оперативно и эффективно показывать прототипы клиенту. Но самое главное, что проект SketchFlow может стать отправной точкой для построения "реального" приложения на платформе WPF или Silverlight. О том, как обращаться с этим компонентом Expression Blend, речь пойдет в главе 8.

Шаблоны проектов Expression Blend

А теперь, когда у вас сложилось общее представление об отдельных членах семейства программных продуктов Microsoft Expression, перейдем к рассмотрению различных типов проектов, поддерживаемых в среде Expression Blend IDE. При запуске Expression Blend появляется экран приветствия. На рис. 1.3 показано содержимое вкладки Help на этом экране.

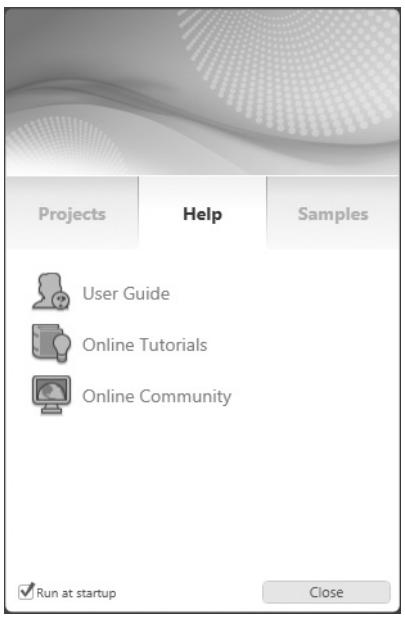


Рис. 1.3. Экран приветствия, появляющийся при запуске Expression Blend

Как видите, экран приветствия разделен на три вкладки: Projects (Проекты), Help (Справка) и Samples (Примеры). Выберите вкладку Projects и щелкните на варианте New Project (Новый проект). Откроется диалоговое окно New Project, приведенное на рис. 1.4[6].

[6] На рис. 1.4 курсор мыши располагается над вертикальным элементом пользовательского интерфейса, что дает возможность показывать и скрывать отображаемое слева иерархическое представление шаблонов проектов.

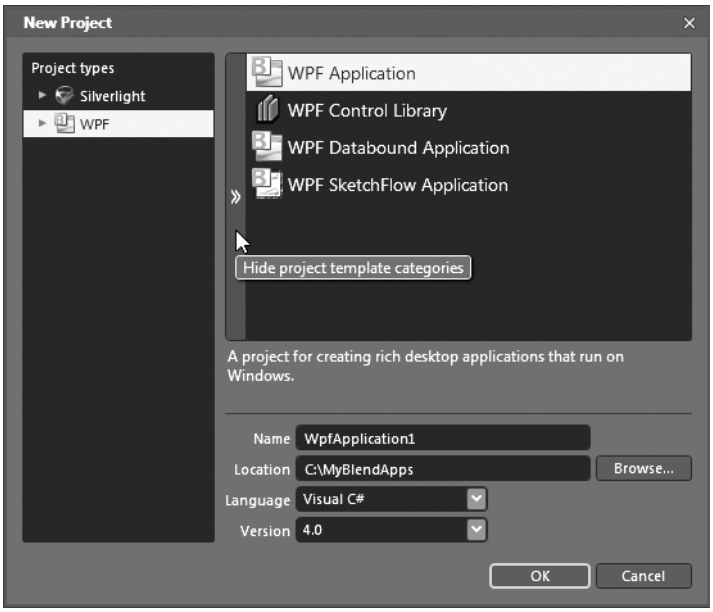


Рис. 1.4. Диалоговое окно New Project в среде Expression Blend

Какой бы шаблон проекта вы ни выбрали, в диалоговом окне New Project можете указать имя и местоположение нового проекта, а также сделать выбор между базовым кодом C# и VB. И последнее, но не менее важное: вы можете настроить новый проект на конкретную версию .NET (для проектов на платформе WPF) или же на платформу Silverlight.

Примечание. В этой книге предполагается, что при проработке приведенных в ней примеров проектов используется язык C#. Если же вы пользуетесь языком VB, вам не составит большого труда привести минимальный код C# в соответствие с синтаксисом VB.

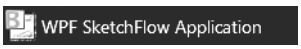
Шаблоны проектов на платформе WPF

Если вас интересует создание нового, наполненного различными возможностями настольного приложения для операционной системы Microsoft Windows, работу над ним вы, скорее всего, начнете с нового проекта WPF. В узле `WPF` иерархического представления шаблонов проектов в диалоговом окне New Project среды Expression Blend 4 для этой цели доступны четыре исходных шаблона, назначение которых вкратце описывается в табл.1.1.

Таблица 1.1. Шаблоны проектов WPF, доступные в Expression Blend

Шаблон проекта WPF	Назначение
WPF Application	Предназначен для построения традиционных настольных приложений на платформе WPF в виде исполняемых файлов
WPF Control Library	Предназначен для создания неоднократно используемых библиотек классов .NET, содержащих специальные элементы управления на платформе WPF. Подробнее о построении специальных элементов управления пользовательского интерфейса, а также стилей и шаблонов для них речь пойдет в главе 5

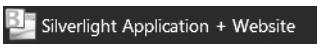
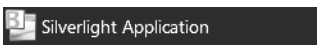
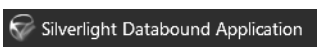
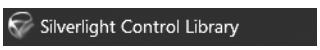
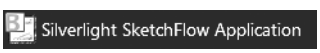
Окончание табл. 1.1

Шаблон проекта WPF	Назначение
	Также предназначен для построения традиционных настольных приложений, но в нем используются объекты типа `View` и `ViewModel`, а также шаблон проектирования MVVM для установления слабой связи между создаваемым представлением и логикой данных[7]
	Позволяет создать прототип нового настольного приложения на платформе WPF, используя компонент SketchFlow, как поясняется в главе 8

Шаблоны проектов на платформе Silverlight

В табл. 1.2 вкратце описывается назначение шаблонов веб-ориентированных проектов Silverlight, доступных для просмотра щелчком кнопкой мыши на узле `Silverlight` иерархического представления шаблонов проектов в диалоговом окне New Project среды Expression Blend 4. Как видите, они очень похожи на родственные им шаблоны проектов WPF.

Таблица 1.2. Шаблоны проектов Silverlight, доступные в Expression Blend

Шаблон проекта Silverlight	Назначение
	Предназначен для создания нового приложения на платформе Silverlight и проекта соответствующего веб-сайта, на котором оно должно размещаться
	Предназначен для создания нового приложения на платформе Silverlight. С его помощью нельзя построить полноценный проект веб-сайта, но при выполнении данного проекта в Expression Blend будет автоматически сформирована простая тестовая HTML-страница (`Default.html`)
	Также предназначен для создания нового приложения на платформе Silverlight, но в нем используются объекты типа `View` и `ViewModel`, а также шаблон проектирования MVVM для установления слабой связи между создаваемым представлением и логикой данных
	Предназначен для создания неоднократно используемых библиотек классов .NET, содержащих специальные элементы управления на платформе Silverlight. Подробнее о создании специальных элементов управления пользовательского интерфейса, а также стилей и шаблонов рассказывается в главе 5
	Позволяет создать прототип нового приложения на платформе Silverlight с помощью компонента SketchFlow, как поясняется в главе 8

А теперь самое приятное: большинство компонентов Expression Blend остается без изменения независимо от выбранного типа шаблона проекта. Не следует, однако, забывать, что проекты на платформах WPF и Silverlight не являются точными копиями друг друга. Несмотря на то что они создаются на основе одних и тех же технологий (языка XAML, модели содержимого элементов управления и пр.), их базовый код не является полностью совместимым.

Даже самое поверхностное сравнение прикладных интерфейсов API показывает, что на платформе WPF полностью поддерживается трехмерная векторная графика, тог-

[7] Этот тип проекта вкратце рассматривается в главе 6, а подробное описание шаблона проектирования MVVM выходит за рамки данной книги.

да как на платформе Silverlight — лишь самая простая, но все же полезная графика в трехмерной перспективе. Кроме того, на платформе WPF поддерживается более сложная модель обработки визуальных подсказок пользовательского интерфейса в разметке XAML с помощью триггеров или диспетчера визуальных состояний (VSM — Visual State Manager), тогда как на платформе Silverlight обработка визуальных подсказок выполняется в разметке XAML исключительно с помощью диспетчера VSM.

Шаблоны проектов на платформе Windows Phone 7

На момент написания этой книги корпорация Microsoft выпустила семейство программных продуктов Windows Phone 7 в качестве платформы для мобильных устройств[8]. Вам, вероятно, уже известно, что собственной средой для разработки приложений под Windows Phone 7 на самом деле служит Silverlight! С учетом этого обстоятельства следует иметь в виду, что среда Expression Blend IDE (а также Visual Studio 2010) может быть обновлена для поддержки шаблонов самых разных проектов под Windows Phone 7 путем их свободной загрузки через Интернет.

Можно с уверенностью сказать, что в следующую версию Expression Blend эти шаблоны войдут как предварительно установленные. Но как бы там ни было, в главе 7 рассматриваются вопросы построения в Expression Blend пользовательских интерфейсов для приложений на платформе Windows Phone 7, а также показывается, каким образом загружается и устанавливается требующийся для этой цели набор инструментов SDK и соответствующие шаблоны проектов, если они еще не доступны в среде разработки.

Примечание. Вам, вероятно, будет приятно узнать, что практически все, что вы узнаете из этой книги или других источников о среде Expression Blend для построения проектов на платформах WPF или Silverlight, распространяется непосредственно и на проекты приложений на платформе Windows Phone 7.

Основы работы в среде Expression Blend IDE

Построение собственных проектов в Expression Blend вы начнете с главы 2, а до тех пор рассмотрим вкратце основы работы в этой интегрированной среде разработки на примере загрузки одного из доступных в ней примеров проектов. Это будет полезно для вас в том отношении, что вы сможете убедиться воочию, как создается реальный проект, а по ходу дела — получить более полное представление о тех видах приложений, которые можно создавать на платформах WPF и Silverlight.

Если диалоговое окно New Project все еще открыто в среде Expression Blend IDE, нажмите клавишу <Esc>, чтобы вернуться к экрану приветствия. (Если вы уже закрыли диалоговое оно New Project, то вернуться к экрану приветствия вы сможете по команде меню Help⇨Welcome Screen (Справка⇨Экран приветствия).) Но в любом случае щелкните на вкладке Samples (рис. 1.5), где находится ряд встроенных в Expression Blend примеров проектов.

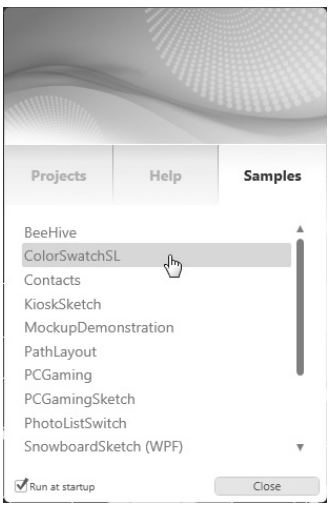


Рис. 1.5. На экране приветствия вы сможете открыть один из ряда встроенных в Expression Blend примеров проектов

[8] Платформа Windows Phone 7 была официально выпущена в октябре 2010 года. Подробнее о ней можно узнать по адресу `www.microsoft.com/windowsphone`.

Конкретный перечень примеров проектов зависит от версии Expression Blend, поэтому не особенно тревожьтесь, если этот перечень не совпадет у вас с тем, что приведен на рис. 1.5. Здесь и далее будет рассматриваться пример проекта Silverlight типа `ColorSwatchSL`, но вы вольны загрузить любой другой пример проекта для дальнейшего изучения.

Загрузка примера проекта в Expression Blend

Прежде чем углубляться в изучение интегрированной среды разработки Expression Blend, поясним, каким образом проект выполняется в Expression Blend, чтобы вы могли сами тестировать разрабатываемое в нем приложение. При нажатии функциональной клавиши <F5> или комбинации клавиш <Ctlr+F5> в Expression Blend создается и выполняется текущий проект. Если вы создаете приложение на платформе WPF, конечный результат, как правило, представляет собой новое окно, появляющееся на рабочем столе операционной системы вашего компьютера. А если создаете новое приложение на платформе Silverlight, то в окне используемого вами по умолчанию браузера загружается и отображается веб-страница, на которой размещается данное приложение.

Итак, выполните проект `ColorSwatchSL`. Обратите внимание на то, что при перемещении курсора мыши по цветным полоскам они выдвигаются на передний план благодаря специальной анимации. Если вы щелкнете на одной из цветных полосок, появится область “подробностей” благодаря еще одной анимационной последовательности. На рис. 1.6 показан один из возможных результатов пробного выполнения данного проекта.

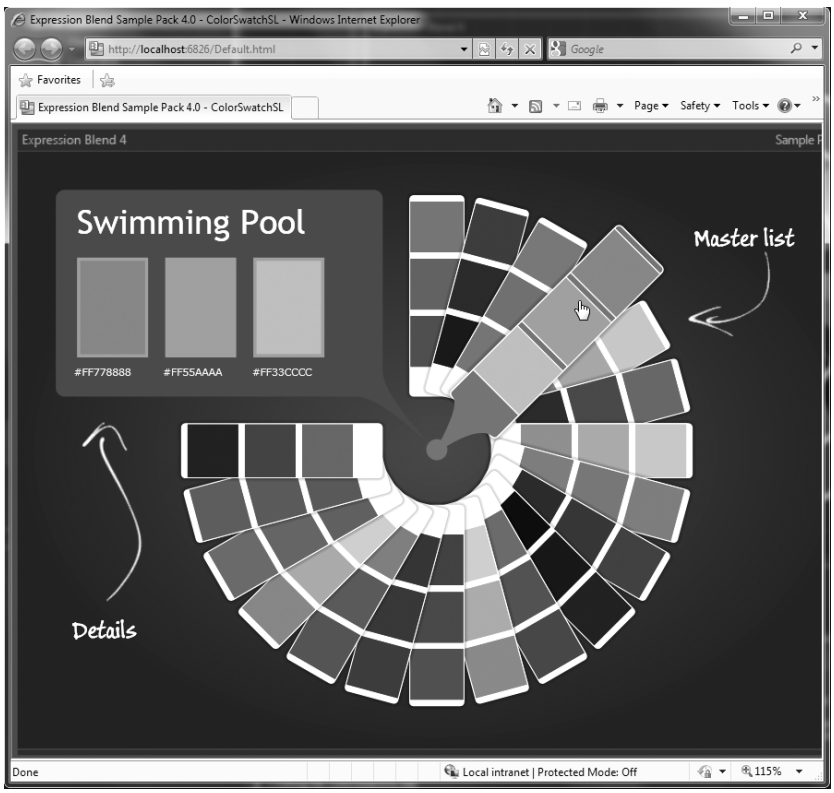


Рис. 1.6. Выполнение проекта `ColorSwatchSL`

По завершении пробного выполнения примера проекта `ColorSwatchSL` закройте окно браузера и вернитесь к Expression Blend.

Монтажный стол и элементы его управления

Монтажный стол располагается в самом центре окна Expression Blend и является едва ли не самым первым компонентом этой интегрированной среды разработки, который сразу же находит применение. Ведь этот визуальный конструктор служит для создания внешнего вида любого окна в приложении WPF или элемента пользовательского интерфейса в приложении Silverlight. На рис. 1.7 показан вид монтажного стола для содержимого файла `MainControl.xaml` в текущем примере проекта.

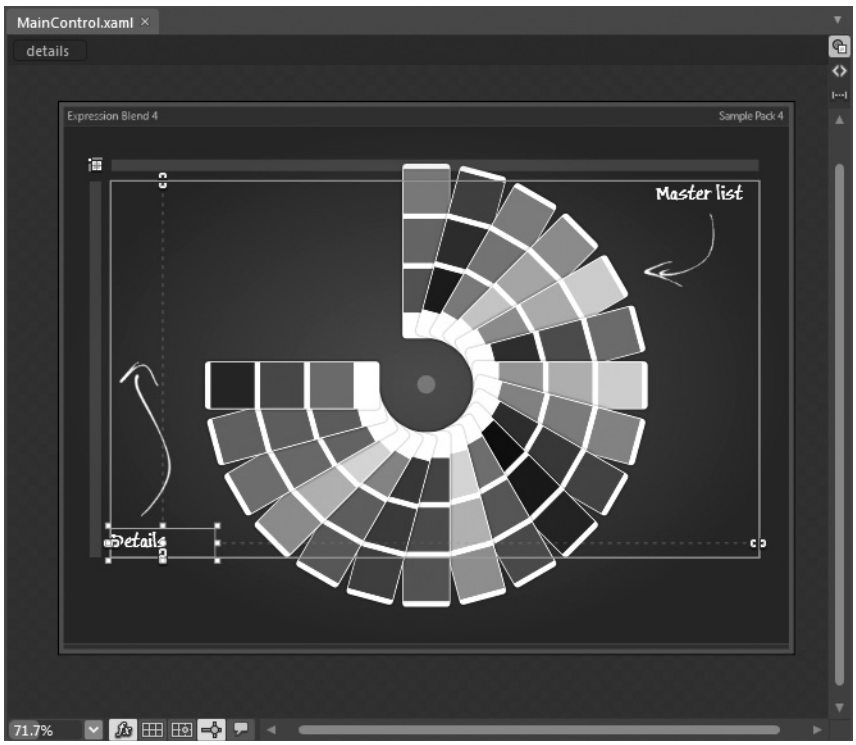


Рис. 1.7. На монтажном столе можно визуально конструировать пользовательский интерфейс приложения WPF или Silverlight

Слева внизу на монтажном столе расположен ряд элементов управления, которые, как и следовало ожидать, называются *элементами управления монтажного стола* (рис. 1.8). С их помощью вы можете изменять общий вид поверхности этого визуального конструктора.

Рис. 1.8. С помощью элементов управления монтажного стола можно настраивать внешний вид этого визуального конструктора

В последующих подразделах приводится краткое описание функций, доступных в области элементов управления монтажного стола, а также ряда других важных инструментов, предоставляемых на монтажном столе.

Изменение масштаба отображения монтажного стола

Слева в области элементов управления монтажного стола находится *элемент управления масштабированием*, с помощью которого можно изменять размеры поверхности визуального конструктора. Манипулируя этим элементом управления, вы обнаружите, что он представляет собой комбинированный элемент управления, в котором можно ввести конкретное значение, выбрать предварительно заданное значение из списка и задать масштаб, щелкнув левой кнопкой мыши и удерживая ее до тех пор, пока курсор не окажется на нужной величине масштаба, как при перемещении ползунка по полосе прокрутки. Возможность изменять размеры текущего монтажного стола оказывается очень полезной при создании, среди прочего, элементов управления со специальным содержимым, шаблонов привязки данных и специальных стилей оформления.

Отображение и сокрытие эффектов визуализации

Далее по порядку следует элемент управления, обозначенный знаком математической функции (`fx`) и служащий для включения или отключения любых эффектов визуализации, накладываемых на элемент пользовательского интерфейса в визуальном конструкторе. Как будет показано далее в книге, в состав Expression Blend входит целый ряд предварительно заданных визуальных эффектов, в том числе эффект падающей тени (см. рис. 1.1). При построении изысканных пользовательских интерфейсов у вас может периодически возникать потребность скрывать подобные визуальные эффекты на время конструирования, чтобы было удобнее настраивать основные элементы пользовательского интерфейса. В рассматриваемом здесь примере проекта визуальные эффекты не применяются, поэтому если вы щелкнете на кнопке `fx`, то ничего особенного не произойдет. Но далее в этой главе вам предстоит ввести визуальный эффект в свой проект.

Корректировка положения элементов пользовательского интерфейса по сетке для привязки

Далее следуют три элемента управления, с помощью которых можно задавать реакцию монтажного стола на расположение на нем элемента пользовательского интерфейса. Если щелкнуть на кнопке Show snap grid (Показать сетку для привязки), на поверхность визуального конструктора будет наложена сетка. После этого можно активизировать одну из двух других кнопок: Turn on snapping to gridlines (Включить привязку к линиям сетки) или Turn on snapping to snaplines (Включить привязку к линиям привязки).

Если включен режим привязки к линиям сетки, то при перемещении объекта по поверхности монтажного стола он будет привязываться или отвязываться от ближайших горизонтальной и вертикальной линий сетки. Такой режим привязки может оказаться полезным при выравнивании положения нескольких элементов управления пользовательского интерфейса по горизонтали или по вертикали.

Режим привязки к линиям привязки оказывается полезным в том случае, если требуется, чтобы два или три элемента пользовательского интерфейса были расположены относительно друг друга вполне определенным образом. Так, активизировав режим привязки к линиям привязки, вы сможете добиться расположения текста в обоих элементах управления на одной и той же горизонтальной линии. В этом режиме можно также привязывать элементы управления к ячейкам диспетчера компоновки по сетке или же по указанной величине заполнения или промежутка между связанными элементами.

На рис. 1.9 приведен произвольный пример работы с представлением привязки к сетке и режимом привязки к линиям привязки. Для опробования этих средств привязки выделите сначала текстовую область `Master list` (Главный список) на поверхности визуального конструктора и переместите ее по этой поверхности. Обратите внимание, насколько изменяется режим работы визуального конструктора при включении и выключении различных средств привязки.

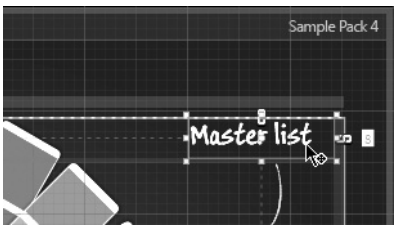


Рис. 1.9. Элементы управления привязкой на монтажном столе позволяют задавать положение содержимого в пределах диспетчера компоновки

Просмотр аннотаций

И последняя кнопка в области элементов управления монтажного стола служит для просмотра или сокрытия аннотаций в Expression Blend. Подобные аннотации можно рассматривать как своего рода записки программирующего в среде Expression Blend. С помощью аннотаций можете вводить текстовые заметки в текущий проект. А это может быть особенно полезно на стадии создания прототипа приложения, как поясняется в главе 8.

В настоящий момент пример проекта `ColorSwatchSL` не содержит никаких аннотаций, поэтому введите в него одну из них по команде меню Tools⇨Create Annotation (Сервис⇨Создать аннотацию). После этого можете набрать в качестве аннотации любые текстовые данные (рис. 1.10). Введя в текущий проект одну или две аннотации, попробуйте активизировать кнопку Annotations, чтобы просмотреть полученный результат.

Рис. 1.10. Аннотации в среде Expression Blend позволяют документировать заметки непосредственно в разрабатываемом проекте

Следует, однако, иметь в виду, что аннотации вообще не видны при выполнении прикладной программы. Кроме того, аннотации, вводимые в проект Expression Blend, не отображаются в конструкторах проектов WPF или Silverlight, разрабатываемых в среде Visual Studio 2010, хотя, проявив настойчивость, можно все же увидеть исходную разметку XAML, в которой представлены данные самой аннотации. И это вполне возможно, поскольку даже аннотации хранятся в формате XAML. Так, аннотация, приведенная на рис. 1.10, размечается в коде XAML следующим образом:

```
<Anno:AnnotationManager.Annotations>
  <Anno:Annotation AuthorInitials="AT" Author="Andrew Troelsen"
    Left="765.43" SerialNumber="1" Top="262.179"
    Timestamp="08/10/2010 15:30:55"
    Text="Trust me, if I can become comfortable working with Blend,
         anyone can do it!"/>
</Anno:AnnotationManager.Annotations>
```

И последнее замечание: не следует забывать, что если в конструкторе выбран какой-нибудь элемент пользовательского интерфейса *до* ввода новой аннотации, то она будет связана с выбранным элементом. Этим обстоятельством можно воспользоваться, например, для комментирования отдельного элемента управления пользовательского интерфейса или же элемента графических данных. Если же вы не выбрали ни одного из элементов до ввода аннотации, а это означает, что вы просто щелкнули на пустом участке монтажного стола, новая аннотация послужит в качестве "общей" заметки, не связанной с каким-нибудь конкретным элементом пользовательского интерфейса.

Просмотр и правка исходной разметки XAML

Если вы занимаетесь только графическим оформлением, вас, возможно, мало интересует просмотр разметки в коде XAML, автоматически формируемой в Expression Blend. Но если вы занимаетесь разработкой программного обеспечения и пользуетесь для этой цели средой Expression Blend, вам будет приятно узнать, что в Expression Blend предоставляется довольно развитый редактор разметки в коде XAML. Но прежде чем просматривать и править подобную разметку, вам придется перейти от кнопки Design (Конструировать) к кнопке XAML или Split (Разделить). Все эти кнопки находятся в правом верхнем углу монтажного стола, как показано на рис. 1.11.

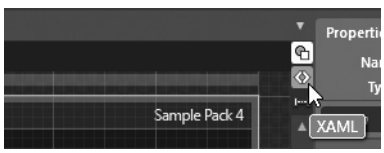


Рис. 1.11. Данные на монтажном столе можно просматривать в режиме конструирования, разметки в коде XAML или разделения того и другого

Активизируйте режим просмотра с разделением, выбрав кнопку Split. В нижней части монтажного стола появится интегрированный в Expression Blend редактор XAML. Как показано на рис. 1.12, в редакторе XAML поддерживается автоматическое завершение кода и система автоматической подсказки IntelliSense, а также предоставляется полезная справочная информация для ввода и правки кода.

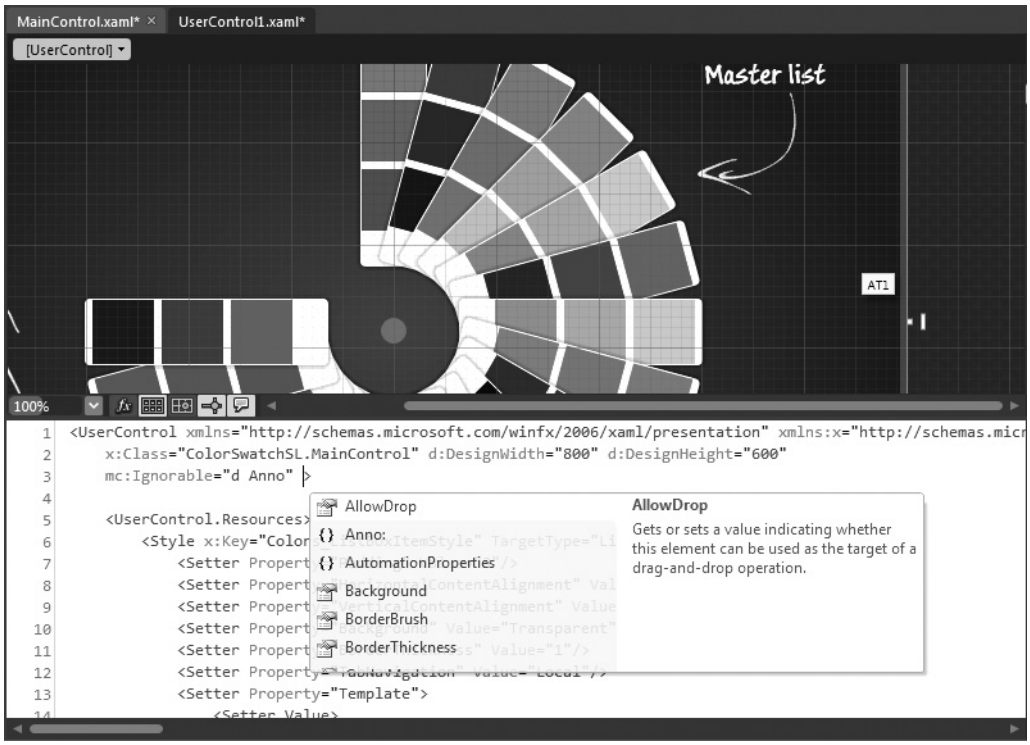


Рис. 1.12. Интегрированный в Expression Blend редактор XAML обладает довольно развитыми функциональными возможностями

На этом краткий обзор монтажного стола в Expression Blend завершается. Далее будут рассмотрены способы просмотра и настройки элементов пользовательского интерфейса, которыми заполняется рабочая область визуального конструктора.

Панель Objects and Timeline

Следующим важным компонентом среды Expression Blend IDE является панель Objects and Timeline (Объекты и временная шкала), располагаемая по умолчанию с левой стороны в рабочем окне Expression Blend. Эта панель служит двум целям. Прежде всего, и это самое главное, на панели Objects and Timeline отображается иерархическое представление разметки XAML для данного конкретного окна или элемента управления пользовательского интерфейса, оформляемого в визуальном конструкторе. Верхний узел этого иерархического представления обозначает все окно в целом (посредством элемента `<Window>`) или же отдельный элемент управления пользовательского интерфейса (посредством элемента `<UserControl>`), тогда как подчиненный ему непосредственно узел — корневой диспетчер компоновки, называемый по умолчанию `LayoutRoot`. Этот диспетчер компоновки может содержать любое количество элементов управления пользовательского интерфейса, отражающих текущее художественное представление автора проекта о том, как должен выглядеть пользовательский интерфейс (подробнее о диспетчерах компоновки и элементах управления речь пойдет в главе 4).

Как показано на рис. 1.13, справа от каждого узла, отображаемого на панели Objects and Timeline, находится пиктограмма с изображением глаза, а еще дальше — кружок. Пиктограмма с изображением глаза служит для отображения и сокрытия отдельных

элементов пользовательского интерфейса в визуальном конструкторе. Это удобно для просмотра отдельных элементов в сложной вложенной системе компоновки пользовательского интерфейса. Следует, однако, иметь в виду, что если скрыть часть пользовательского интерфейса, это будет сделано *только* в визуальном конструкторе! А полностью воспроизведенную разметку пользовательского интерфейса можно всегда посмотреть, запустив текущий проект на выполнение.

С помощью кружка, расположенного справа от пиктограммы с изображением глаза, можно блокировать отдельные объекты (и любые содержащиеся в них объекты), чтобы их нельзя было править в визуальном конструкторе. Нетрудно догадаться, что подобная возможность оказывается полезной в том случае, если оформление отдельного элемента пользовательского интерфейса доведено по такого совершенства, что его случайное изменение в дальнейшем было бы нежелательным.

Рис. 1.13. Элементы пользовательского интерфейса можно выборочно скрывать и блокировать

Выбор объектов для правки

Помимо обычного просмотра иерархического представления разметки, его узлы на панели Objects and Timeline позволяют быстро и просто выбирать отдельные элементы в визуальном конструкторе для последующей правки. Попробуйте щелкнуть на разных узлах иерархического представления разметки, чтобы посмотреть, какая именно часть монтажного стола выбирается при этом для правки. Это очень важное свойство данной панели, и поэтому вы должны вполне освоиться с ним.

Назначение временной шкалы

Итак, мы рассмотрели одно назначение панели Objects and Timeline, связанное с объектами, а теперь перейдем к другому ее назначению, касающемуся временной шкалы анимации. Как оказывается, на панели Objects and Timeline можно также создавать *объекты раскадровки*, содержащие инструкции для анимации. Используя это назначение данной панели, можно выбрать отдельный узел в иерархическом представлении разметки и видоизменить его самыми разными способами: изменить местоположение, цвет и прочие свойства. По ходу выполнения этих действий они регистрируются инструментами анимации в Expression Blend. Редактор анимации представляет собой довольно развитый компонент Expression Blend и более подробно рассматривается в главе 3. А до тех пор можете воспользоваться панелью Objects and Timeline лишь для просмотра текущей разметки и выбора отдельных элементов с целью их правки на монтажном столе.

Панель Properties

Теперь, когда вы знаете, каким образом отдельные элементы пользовательского интерфейса выбираются на панели Objects and Timeline, перейдем к рассмотрению панели Properties (Свойства), располагаемой по умолчанию с правой стороны в рабочем окне Expression Blend. Аналогично окну Properties в среде Visual Studio 2010, этот компонент Expression Blend позволяет видоизменять выбранный элемент самыми разными способами. Однако, если видоизменить выбранный элемент на монтажном столе, например, перетащить его в другое место мышью, связанные с ним свойства на панели Properties будут обновлены. Но в любом случае исходная разметка в коде XAML видоизменяется автоматически в среде Expression Blend.

Панель Properties разделяется на различные категории свойств, каждая из которых может быть развернута или свернута независимо от остальных. Конкретные категории свойств будут динамически изменяться на панели Properties в зависимости от того, что именно выбрано в настоящий момент в визуальном конструкторе[9]. Так, если выбрать все окно или только отдельный элемент управления пользовательского интерфейса, на панели Properties появится целый ряд категорий свойств (в основном в свернутом виде), в том числе Brushes (Кисти), Appearance (Внешний вид), Layout (Компоновка) и Common Properties (Общие свойства). На рис. 1.14 представлены те категории свойств, которые обычно появляются на панели Properties при выборе самого верхнего объекта типа `UserControl` на панели Objects and Timeline.

Рис. 1.14. На панели Properties предоставляется возможность изменить свойства выбранного в настоящий момент элемента

[9] Следует также иметь в виду, что конкретные категории свойств отображаются на панели Properties в зависимости от типа разрабатываемого проекта: WPF, Silverlight или Windows Phone 7.

Именование и поиск объектов

Рассмотрим вкратце самую верхнюю часть панели Properties, где находится текстовое поле Name. Нетрудно догадаться, что в этом поле указывается значение для свойства `Name` (Имя) конкретного элемента XAML, что дает возможность манипулировать им непосредственно в исходном коде проекта. Непосредственно под этим полем находится текстовое поле Search (Поиск), помогающее быстро найти нужное свойство по имени, вместо того, чтобы искать его вручную по категориям. Для проверки такой возможности введите в текстовом поле Search значение **`height`** (высота). По мере ввода этого значения на панели Properties будут появляться те свойства, которые полностью или частично совпадают с заданным критерием поиска, как показано на рис. 1.15. Если же очистить текстовое поле Search, все категории свойств появятся снова[10].

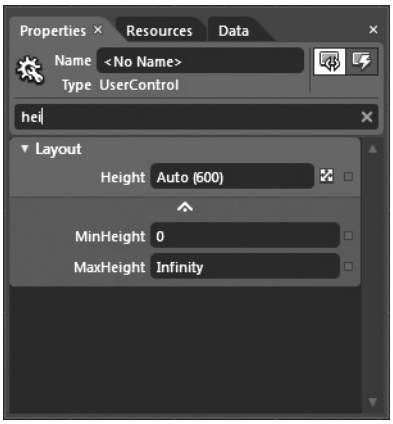


Рис. 1.15. Найти нужное свойство на панели Properties не составляет большого труда

Краткий обзор категорий свойств

Прорабатывая материал данной книги, вы раскроете для себя целый ряд важных свойств панели Properties в контексте конкретной темы, будь то графика, компоновка, элементы управления, анимация и т.д. Но для начала ознакомьтесь с кратким описанием назначения наиболее часто используемых категорий свойств, приведенных в табл. 1.3 в алфавитном порядке.

Таблица 1.3. Категории свойств, чаще всего используемые на панели Properties

Категория свойств	Назначение
Appearance	Определяются общие свойства визуализации, в том числе непрозрачность, видимость и применяемые графические эффекты (размывание, падающая тень и пр.)
Brushes	Предоставляется доступ к визуальному редактору кистей
Common Properties	К этой категории относятся свойства, общие для большинства элементов пользовательского интерфейса, в том числе всплывающие подсказки, индексы перехода по табуляции, местоположение контекста данных (для операций привязки данных)

[10] По завершении поиска свойства не забывайте очистить текстовое поле Search. Трудно даже подсчитать, сколько раз я забывал сделать это, недоумевая впоследствии, почему свойства, отображаемые на панели Properties, не соответствуют элементу, выбранному на монтажном столе.

Окончание табл. 1.3

Категория свойств	Назначение
Layout	Обычно правятся свойства, определяющие физические размеры элементов управления, в том числе высоту, ширину, поля и пр.
Miscellaneous	К этой категории, по существу, относятся все остальные свойства, доступные на панели Properties. Но самое главное, что в ней можно задать конкретный стиль оформления или шаблон для выбранного элемента
Text	Настраиваются текстовые свойства выбранного элемента, в том числе параметры шрифтового оформления, разбиение на абзацы и отступы
Transform	Допускается выполнение графических преобразований над выбранным элементом, в том числе вращение, поворот на заданный угол, смещение и пр.

Доступ к дополнительным свойствам

Продолжая исследовать панель Properties, вы непременно обнаружите, что у некоторых категорий свойств имеются развертываемые вниз области дополнительных свойств. Так, если щелкнуть на такой области, соответствующая категория развернется для отображения ее дополнительных свойств, которые расширяют возможности данной категории, хотя используются нечасто. В качестве примера на рис. 1.16 показана в развернутом виде категория свойств Text.

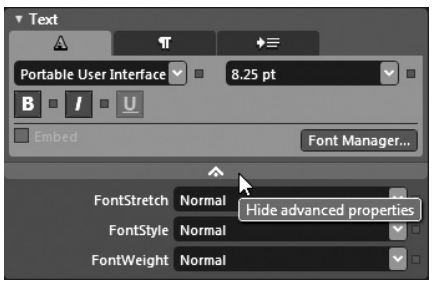


Рис. 1.16. У некоторых категорий свойств имеются развертываемые области дополнительных свойств

Что касается дополнительной настройки свойств, то справа от некоторых свойств отдельной категории указывается квадратик, как показано на рис. 1.17. Щелкнув на этом квадратике, обозначающем кнопку Advanced options (Дополнительные параметры), вы откроете еще одно окно редактора с дополнительными параметрами настройки *отдельного свойства*. Такая возможность оказывается полезной при выполнении операций привязки данных и обращении с ресурсами объектов, как поясняется в последующих главах.

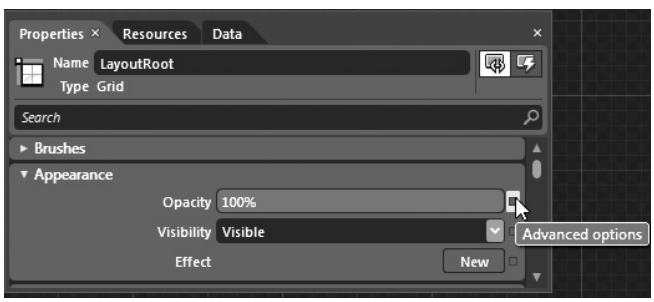


Рис. 1.17. У некоторых свойств имеются дополнительные параметры настройки

Всего сказанного выше должно быть достаточно для того, чтобы вы могли свободно пользоваться панелью Properties. А в последующих главах возможности панели Properties будут рассмотрены более подробно.

Панель Project

Панель Project (Проект), располагаемая по умолчанию слева в рабочем окне Expression Blend, должна быть хорошо знакома тем, у кого имеется опыт работы в среде Microsoft Visual Studio IDE. Всякий раз, когда вы создаете в Expression Blend новый проект, вместе с ним автоматически создается целый ряд первоначальных файлов (разметки XAML и исходного кода), а также делаются ссылки на нужные библиотеки .NET, или так называемые сборки. При создании проектов вы вольны вводить в них дополнительные типы файлов и добавлять ссылки на другие библиотеки .NET. Более того, можете добавлять в текущий проект новые папки, содержащие связанные с ним ресурсы, в том числе файлы изображений, видео- и аудиозаписей, а также XML-документы.

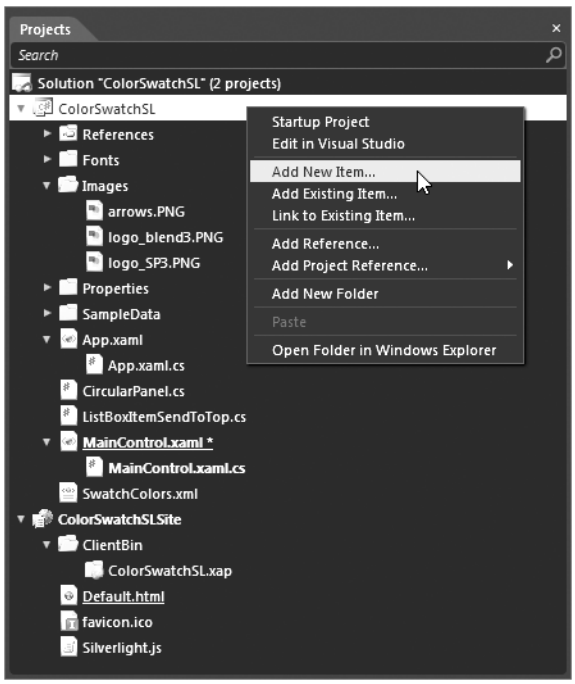


Рис. 1.18. На панели Project текущее решение отображается с разбиением на отдельные проекты или файлы

Если вы загрузили пример проекта `ColorSwatchSL`, то обнаружите на панели Project не один, а два проекта. Первый проект содержит файлы исходного кода и библиотеки для приложения Silverlight, тогда как второй проект — файлы, связанные с веб-страницей, на которой размещается данное приложение. Как и в среде Visual Studio, в Expression Blend используются понятия *решения* и *проекта*. Отдельное решение может содержать несколько проектов, которые совместно представляют разрабатываемое приложение. Как показано на рис. 1.18, в один из двух проектов в рассматриваемом здесь примере добавляются новые элементы из контекстного меню, вызываемого щелчком правой кнопкой мыши. Это делается лишь для того, чтобы показать, каким образом в проект вводятся новые элементы. Хотя в рассматриваемом здесь примере этого пока что не требуется.

Интегрированный редактор исходного кода

Несмотря на то что XAML позволяет сделать немало полезного и замечательного, не написав ни единой строки кода C# или VB, в реальные проекты в конечном итоге приходится вводить определенный объем кода для приведения в действие функций разрабатываемого приложения. В предыдущих версиях среды Expression Blend IDE отсутствовало какое-либо подобие редактора исходного кода. Так, если бы вы дважды щелкнули на файле исходного кода C# (с расширением `.cs`) или VB (с расширением `.vb`) на панели Projects, произошел бы автоматический запуск среды Visual Studio (или текстового редактора Notepad либо WordPad, если среда Visual Studio не установлена).

В состав текущей версии Expression Blend включен полезный редактор исходного кода. Для того чтобы убедиться в этом, дважды щелкните на файле исходного кода `ListBoxItemSendToTop.cs` (или любом выбранном вами исходном файле), доступном на панели Project. Интегрированный в Expression Blend редактор исходного кода оказывается полезным в том случае, если требуется ввести код простой "заглушки" для обработчиков событий или написать несложный тестовый код в ходе разработки или создания прототипа приложения (и то и другое вам предстоит еще сделать, прорабатывая материал остальной части книги).

Несмотря на всю свою полезность, интегрированный в Expression Blend редактор исходного кода не идет ни в какое сравнение по своим функциональным возможностям с аналогичным редактором в Visual Studio, хотя от него этого и не требуется. В частности, в нем отсутствует встроенный отладчик, поддержка фрагментов кода C# или VB, средств реорганизации кода и визуального конструирования иерархий классов. Но, с другой стороны, в интегрированном в Expression Blend редакторе исходного кода поддерживается система автоматической подсказки IntelliSense и автоматическое завершение кода (рис. 1.19).

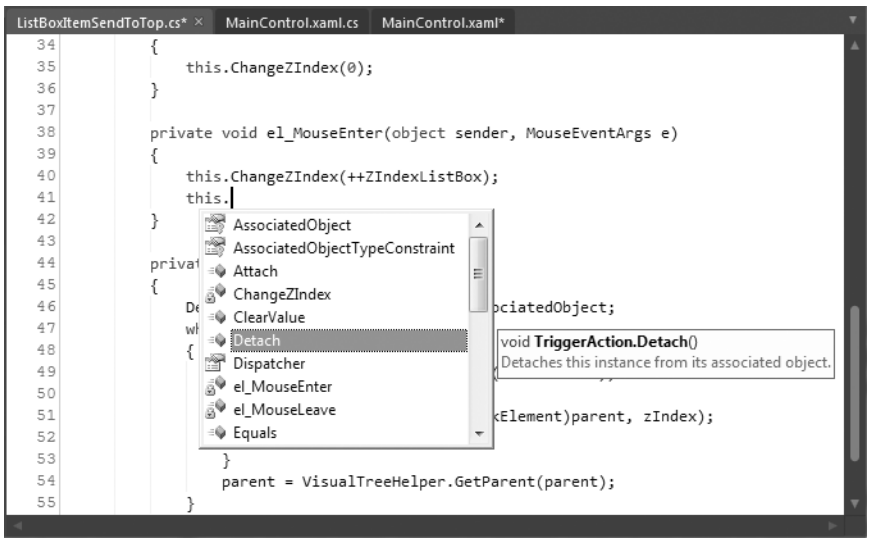


Рис. 1.19. Интегрированный в Expression Blend редактор исходного кода позволяет вводить код в процессе создания прикладной программы

Данная книга не посвящена построению полноценных приложений на платформе WPF или Silverlight, тем не менее, вам придется периодически писать код, прорабатывая представленные далее примеры. Этот код не особенно сложный, но если вам потребуется дополнительная помощь, например, в отладке кода, не забывайте, что в Expression Blend и Visual Studio применяется один и тот же формат решений и проектов. Это означает, что вы сможете без особого труда открыть в Visual Studio проект, созданный вами в Expression Blend, чтобы задать точки прерывания для отладки уже имеющегося кода или написать более сложный код.

Панель Results

При наборе кода или разметке документа вручную вы, конечно, можете допустить ошибки или опечатки, неверно набрав или выделив прописными буквами ключевое слово, не закрыв скобкой кодовый блок или не завершив элемент XAML соответствующим

дескриптором, и т.п. Если вы попытаетесь выполнить приложение, построенное таким образом в Expression Blend, нажав функциональную клавишу <F5> или комбинацию клавиш <Ctlr+F5>, на панели Results (Результаты) появится перечень обнаруженных ошибок. А если вы дважды щелкнете на любой обнаруженной ошибке, то в Expression Blend автоматически откроется соответствующий исходный файл, а курсор установится на том месте, где была обнаружена ошибочная строка кода или разметки документа. В качестве примера на рис. 1.20 показано, каким образом на панели Results выводятся обнаруженные ошибки, намеренно внесенные в проект `ColorSwatchSL` в целях демонстрации.

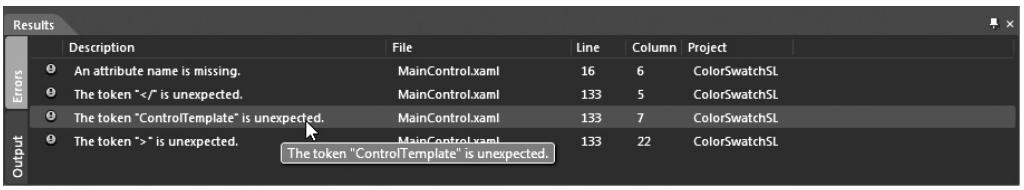


Рис. 1.20. На панели Results отображаются ошибки, обнаруженные при выполнении текущего проекта

Панель Tools

Закройте текущий файл исходного кода, если открыли его, следуя приведенным выше указаниям, а затем откройте XAML-файл, например `MainWindow.xaml`. По умолчанию слева в рабочем окне Expression Blend появляется вертикальная полоса кнопок, напоминающая панель инструментов в Visual Studio тем, у кого имеется опыт работы в этой среде. Это и есть панель Tools (Инструменты), на которой доступны самые разные и чаще всего используемые элементы пользовательского интерфейса, в том числе элементы управления, диспетчеры компоновки, простые геометрические формы и пр. Эти элементы можно выбирать на панели Tools для создания собственных пользовательских интерфейсов.

Справа от отдельных пиктограмм на панели Tools можно обнаружить мелкие белые треугольники. Если щелкнуть на таком треугольнике, не отпуская кнопку мыши, появится список элементов, связанных с данным элементом. В качестве примера на рис. 1.21 показан список дополнительных инструментов Ellipse (Эллипс) и Line (Линия), появляющихся в том случае, если щелкнуть на белом треугольнике инструмента Rectangle (Прямоугольник), не отпуская кнопку мыши. Однако внешний вид панели Tools может изменять в зависимости от типа текущего проекта: WPF или Silverlight. Кроме того, на рис. 1.21 панель Tools показана расположенной у верхнего, а не у левого края рабочего окна Expression Blend в результате специальной настройки данной среды.

Рис. 1.21. Некоторые элементы на панели Tools сгруппированы вместе со связанными с ними элементами

Изучая материал последующих глав, вы получите возможность поработать с самыми разными инструментами, доступными на панели Tools, при рассмотрении отдельных тем. А до тех пор рассмотрим следующие элементы панели Tools.

- Инструменты обычного и прямого выделения.
- Инструменты масштабирования и панорамирования.
- Библиотека ресурсов.

Ниже будут рассмотрены элементы панели Tools, представленные на рис. 1.22, где остальные элементы, оставляемые пока что без внимания, размыты. Прежде всего следует обратить внимание на инструменты обычного выделения. А для того чтобы стали более понятными отличия инструментов обычного выделения от инструментов прямого выделения, в рассматриваемый здесь пример проекта `ColorSwatchSL` придется добавить для целей демонстрации немного нового содержимого.

Рис. 1.22. Эти элементы панели Tools будут подробнее рассмотрены в ряде последующих подразделов

Добавление специального содержимого в пример проекта

Для того чтобы добавить специальное содержимое в рассматриваемый здесь пример проекта, выберите узел `LayoutRoot` в иерархическом представлении на панели Objects and Timeline, поскольку далее вам предстоит ввести новый объект в главном диспетчере компоновки. Затем выберите инструмент Pencil (Карандаш) на панели Tools (он находится в группе инструмента Pen (Перо), как показано на рис. 1.23). Для того чтобы выбрать инструмент Pencil, можно также нажать клавишу <Y>.

Воспользуйтесь мышью, чтобы нарисовать произвольную, замкнутую геометрическую форму на свободном участке монтажного стола. Эта форма может быть любой конфигурации и размера, а ее местоположение не имеет особого значения. Убедитесь в том, что вновь созданная геометрическая форма выделена на панели Objects and Timeline (в иерархическом представлении она должна быть обозначена узлом `[Path]`), а затем воспользуйтесь панелью Properties, чтобы присвоить вновь созданной форме подходящее имя, например `myPolygon` (мой многоугольник), в текстовом поле Name, расположенном в верхней части данной панели. И наконец, воспользуйтесь редактором кистей, доступном в категории свойств Brushes, чтобы выбрать сплошной цвет для заполнения формы. С этой целью щелкните непосредственно на образце цвета заполнения (более подробно редактор кистей рассматривается в главе 2). На рис. 1.24 приведен один из возможных вариантов визуализации произвольной формы.

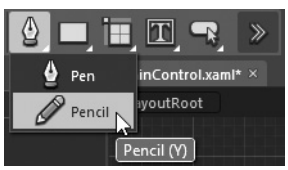


Рис. 1.23. Выберите инструмент Pencil на панели Tools

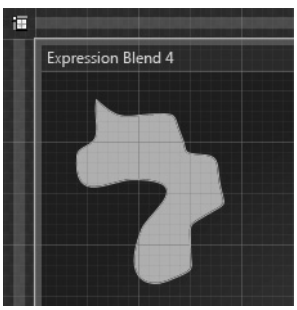


Рис. 1.24. Форма простого многоугольника, созданная инструментом Pencil

Далее вам предстоит организовать взаимодействие с этим объектом посредством кода, а до тех пор следует поближе ознакомиться с самой панелью Tools.

Инструменты обычного и прямого выделения

На панели Tools доступны два разных инструмента выделения элементов на монтажном столе: Selection (Выделение) и Direct Selection (Прямое выделение)[11]. В связи с этим невольно возникает вопрос: а зачем вообще нужны два инструмента для выделения элементов? Дело в том, что если попытаться просто выделить элемент на монтажном столе, чтобы переместить его в пределах окна или отдельного элемента управления или же изменить его размеры маркерами, вытягиваемыми из ограничивающей рамки в визуальном конструкторе, то для этого потребуется *обычное* выделение. С этой целью активизируйте сначала инструмент Selection (с изображением черной стрелки на пиктограмме), нажав оперативную клавишу <V>, а затем проверьте, сможете ли вы изменить размеры или переместить специальный графический объект (созданную ранее произвольную геометрическую форму) на монтажном столе.

С другой стороны, если требуется выделить *отдельные отрезки*, составляющие геометрическую форму, или же поправить сложное внутреннее содержимое класса, производного от класса `ContentControl` (подробнее об этом — в главе 4), то для этого потребуется *прямое* выделение. С этой целью активизируйте инструмент Direct Selection (с изображением белой стрелки на пиктограмме), нажав оперативную клавишу <A>, и обратите внимание на то, что теперь вы сможете выделять отдельные отрезки многоугольной формы (рис. 1.25).

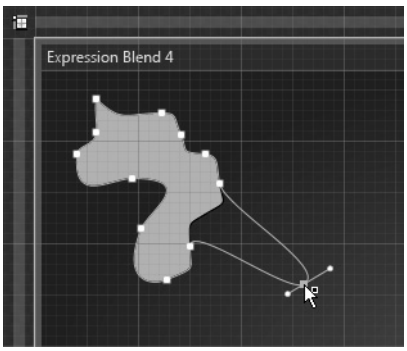


Рис. 1.25. Благодаря прямому выделению можно править отдельные отрезки геометрической формы или сложного содержимого элемента управления

Видоизменив произвольную геометрическую форму, вновь щелкните на пиктограмме инструмента Selection, чтобы еще раз переместить эту форму по поверхности визуального конструктора. Затем перейдите в режим прямого выделения, выбрав инструмент Direct Selection, измените любой отрезок данной формы и опять вернитесь к режиму обычного выделения. Вам придется привыкнуть к частой смене режимов выделения во время работы над своим проектом, поэтому постарайтесь запомнить оперативные клавиши <V> и <A> для быстрого выбора инструментов Selection и Direct Selection соответственно.

Инструменты масштабирования и панорамирования

Изменить масштаб участка монтажного стола с графическими данными можно, перемещая колесико мыши или же настраивая соответствующий элемент управления в

[11] Следует признать, что когда я впервые начал работать с Expression Blend, то потратил немало времени и нервов, чтобы выяснить отличия между двумя этими инструментами выделения!

левом нижнем углу монтажного стола, как пояснялось ранее в этой главе. Но имеется и третий способ: воспользоваться инструментом Zoom (Масштаб), который можно выбрать как на панели Tools, так и нажатием оперативной клавиши <Z>.

Итак, выберите инструмент Zoom на панели Tools (с изображением увеличительного стекла на пиктограмме), а затем щелкните на любом участке монтажного стола, чтобы увеличить его масштаб. Если вы дополнительно нажмете клавишу <Alt>, щелкая на выбранном участке монтажного стола, его масштаб уменьшится. И наконец, если вы дважды щелкнете на пиктограмме инструмента Zoom на панели Tools, участок монтажного стола с графическими данными вернется к исходному масштабу, определяемому в разметке XAML.

Инструмент Pan (Панорамирование), с изображением руки на пиктограмме, служит альтернативой комбинации клавиши <Ctrl> и колесика мыши в том отношении, что если выбрать сначала этот инструмент, нажав оперативную клавишу <H>, а затем щелкнуть и перетащить курсор мыши по монтажному столу, то произойдет переход к соответствующему участку, визуализируемому на монтажном столе. Инструментом Pan удобнее всего пользоваться в тех случаях, когда масштаб сложного графического объекта слишком сильно изменен и требуется перейти к конкретному участку, чтобы поправить встроенное содержимое.

Библиотека ресурсов и панель Assets

На панели Tools отображаются далеко не все элементы, которые могут быть использованы для построения пользовательского интерфейса приложения на платформе WPF или Silverlight. Когда потребуются дополнительные элементы пользовательского интерфейса, на помощь может прийти библиотека ресурсов (`Assets`), которую нетрудно открыть, щелкнув на крайней справа кнопке на панели Tools (на этой кнопке изображен знак >>). Как показано на рис. 1.26, содержимое библиотеки ресурсов разделено на несколько категорий высокого уровня.

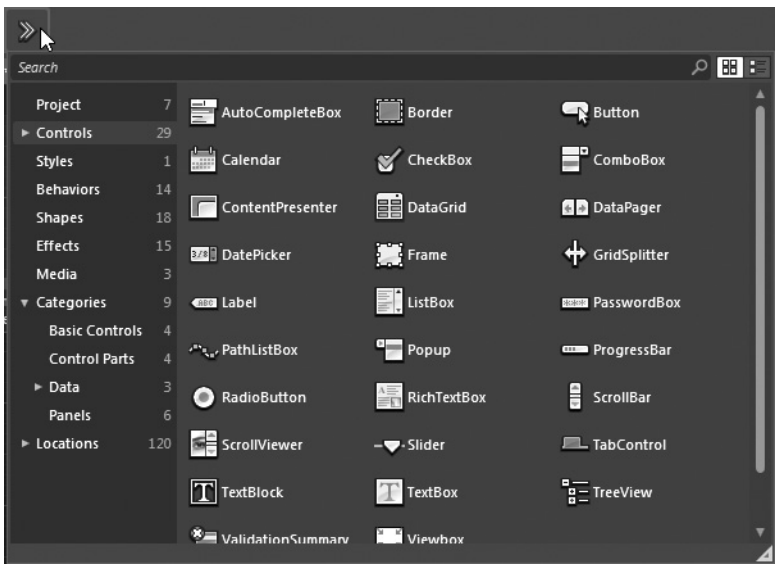


Рис. 1.26. В библиотеке ресурсов имеется целый ряд дополнительных ресурсов для построения приложений на платформах WPF и Silverlight

В табл. 1.4 вкратце описывается общее назначение дополнительных ресурсов по отдельным категориям.

Таблица 1.4. Категории библиотеки ресурсов в Expression Blend

Категория библиотеки ресурсов	Назначение
Project	К этой категории относятся все специальные ресурсы, добавленные в текущий проект, в том числе файлы изображений, видео- и аудиозаписей, специальные стили оформления и пр.
Controls	К этой категории относятся все элементы управления, используемые для построения пользовательского интерфейса приложений на платформе WPF или Silverlight
Styles	К этой категории относятся любые стили оформления, специально созданные для текущего проекта
Behaviors	К этой категории относятся виды поведения, которые представляют собой объекты, позволяющие фиксировать типичные события непосредственно в разметке, не прибегая к необходимости писать для этой цели специальный код C# или VB. Более подробно объекты поведения будут рассматриваться с главы 3
Shapes	К этой категории относятся предварительно визуализированные геометрические формы (шестиугольники, выноски, звезды и т.д.), которые можно добавлять в текущий проект. Это дает возможность вводить стандартные формы более оперативно, чем с помощью инструментов Pen и Pencil
Effects	К этой категории относятся эффекты, с помощью которых можно изменять внешний вид элементов пользовательского интерфейса самыми разными способами. Напомним, что эффекты можно включать и выключать с помощью кнопки **fx** из области элементов управления монтажного стола
Media	Эта категория подобна категории Project в том отношении, что в ней доступны специальные ресурсы текущего проекта, но к категории Media относятся только файлы изображений, видео- и аудиозаписей
Categories	К этой категории относятся все ресурсы текущего проекта, разделенные на подчиненные категории. Это дает возможность, например, быстро просматривать все элементы управления, применяемые в текущем проекте; все элементы управления, в которых используется привязка данных, и т.д.
Locations	К этой категории относятся все библиотеки .NET (или так называемые сборки), содержащие различные ресурсы, применяемые в проектах на платформах WPF и Silverlight

Во время работы в Expression Blend IDE приходится нередко обращаться к дополнительным ресурсам, и поэтому в этой среде предоставляется еще один способ доступа к дополнительным ресурсам через панель Assets (Ресурсы), которая показана на рис. 1.27. Я лично пользуюсь панелью Assets намного чаще, чем библиотекой ресурсов, поскольку она постоянно отображается на экране и не исчезает из виду, как библиотека ресурсов, но вы вольны выбирать тот способ доступа к дополнительным ресурсам, который больше всего вам подходит.

Попробуйте воспользоваться панелью Assets (или библиотекой ресурсов) на практике. Сначала перейдите к панели Objects and Timeline и убедитесь в том, что объект `LayoutRoot` выбран в иерархическом представлении. Затем выберите объект `Star` в категории Shapes (Формы) и нарисуйте небольшую звезду рядом с созданной ранее произвольной геометрической формой, как показано на рис. 1.28, после чего присвойте ей имя `myStar` (моя звезда) в текстовом поле на панели Properties.

Еще раз выберите объект `LayoutRoot` на панели Objects and Timeline и найдите эффект `Ripple` (Волнистость) на панели Assets (напомним, что имя нужного элемента можно ввести в текстовом поле Search для его быстрого поиска). Перетащите эффект `Ripple` на узел `LayoutRoot` стандартным методом перетаскивания объектов мышью (рис. 1.29).

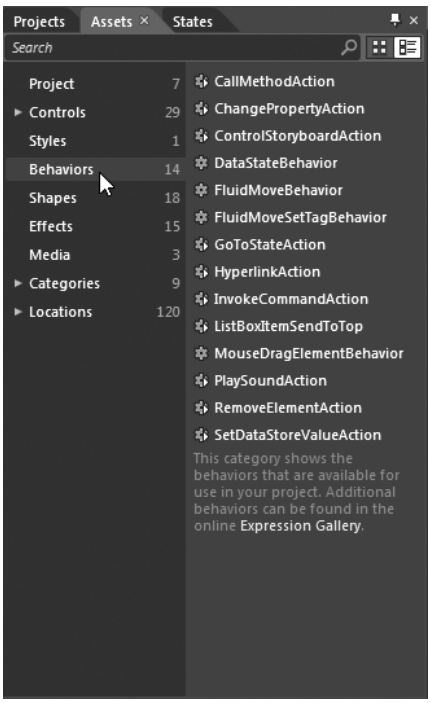


Рис. 1.27. Панель Assets предоставляет еще один способ поиска нужных элементов в библиотеке ресурсов

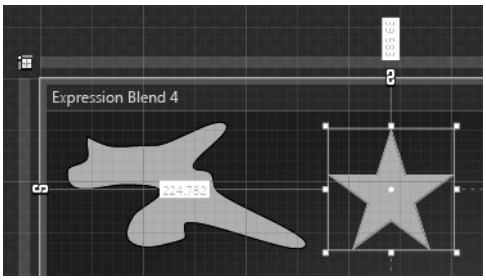


Рис. 1.28. Добавление второй геометрической формы в диспетчер компоновки

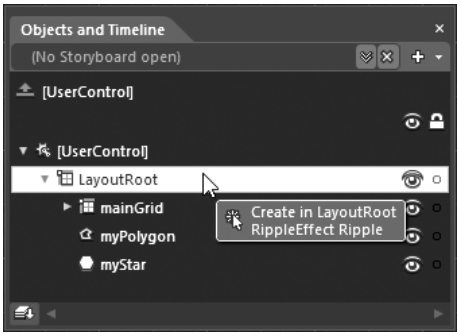


Рис. 1.29. Добавление эффекта волнистости в диспетчер компоновки вообще и во все его подчиненные объекты в частности

Как только вы добавите данный эффект, компоновка пользовательского интерфейса на монтажном столе примет совершенно другой вид, как показано на рис. 1.30!

Рис. 1.30. Показ и сокрытие визуальных эффектов на монтажном столе

Прорабатывая материал этой книги, вам придется еще не раз обращаться к библиотеке ресурсов. Поэтому не старайтесь запомнить каждую категорию дополнительных ресурсов. А теперь рассмотрим, как пользоваться интегрированным в Expression Blend редактором исходного кода, чтобы добавить в рассматриваемый здесь пример проекта немного интерактивности.

Обработка и реализация событий

В качестве заключительного видоизменения рассматриваемого здесь примера проекта `ColorSwatchSL` найдите объект `myPolygon` на панели Objects and Timeline. Затем перейдите к панели Properties и в ее правом верхнем углу щелкните на кнопке Events (События), чтобы открыть редактор событий. Пиктограмма кнопки Events обозначена знаком молнии, как показано на рис. 1.31.

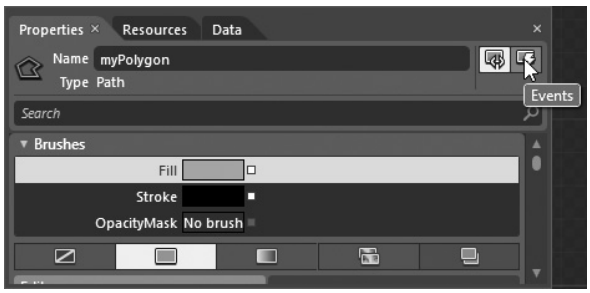


Рис. 1.31. На панели Properties имеются средства, позволяющие обрабатывать события, наступающие для объектов

Щелкнув на кнопке Events, найдите далее событие `MouseLeftButtonDown`. Введите имя метода, который должен вызываться, когда пользователь щелкнет левой кнопкой мыши на специальной многоугольной форме (рис. 1.32). Именно поэтому обрабатываемое этим методом событие называется `MouseLeftButtonDown`.

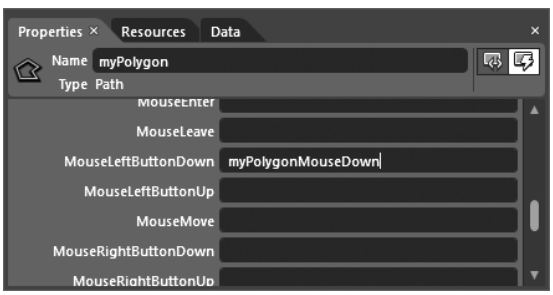


Рис. 1.32. Ввод имени обработчика событий

Введя имя метода обработки упомянутого выше события, нажмите клавишу <Enter>. В итоге откроется окно редактора исходного кода. Обновите первоначальный код рассматриваемого здесь проекта приведенной ниже логикой, отменяющей визуальный эффект волнистости в диспетчере компоновки.

```
private void myPolygonMouseDown(object sender,
  System.Windows.Input.MouseButtonEventArgs e)
{
    // Удалить визуальный эффект волнистости.
    LayoutRoot.Effect = null;
}
```

А теперь нужно обработать аналогичным образом событие `LeftMouseButtonDown`, наступающее при нажатии левой кнопки мыши на звездообразной геометрической форме, но на этот раз должен вызываться метод `myStarMouseDown`. С этой целью обновите первоначальный код рассматриваемого здесь проекта следующей логикой.

```
private void myStarMouseDown(object sender,
  System.Windows.Input.MouseButtonEventArgs e)
{
    // Попытаться ввести другой эффект!
    LayoutRoot.Effect = new System.Windows.Media.Effects.BlurEffect();
}
```

Далее запустите приложение на выполнение, нажав функциональную клавишу <F5> или комбинацию клавиш <Ctrl+F5>. Сначала при выполнении данного приложения проявится эффект волнистости. Но если вы щелкнете левой кнопкой мыши на любой из геометрических форм, то произойдет одно из двух: либо полная отмена визуальных эффектов, либо появление эффекта размывания. Не такой уж и плохой результат всего для двух строк кода! Разумеется, волнистость или размывание всего пользовательского интерфейса требуется далеко не во всех приложениях на платформе WPF или Silverlight. Впрочем, о том, насколько полезными могут оказаться визуальные эффекты при построении реальных проектов, речь пойдет далее в этой книге.

Исходный код. Проект `ColorSwatchSL _ Modified` находится в папке `Ch 1 Code` загружаемого архива примеров проектов к данной книге.

Настройка параметров и режимов работы Expression Blend IDE

А теперь перейдем к рассмотрению способов специальной настройки среды Expression Blend IDE. Прежде всего, по команде меню Tools⇨Options (Сервис⇨Параметры) можно установить общие глобальные параметры настройки и режимы работы Expression Blend IDE, в том числе шрифт выделения исходного текста, вводимого в редакторе исходного кода; устанавливаемые по умолчанию параметры настройки нового монтажного стола; выбираемые по умолчанию имена и инициалы для аннотаций и прочие основные настройки. Особый интерес представляет одна настройка, доступная в области Workspace (Рабочее пространство) диалогового окна Options. С ее помощью можно выбирать светлый или темный фон темы для среды Expression Blend IDE, как показано на рис. 1.33.

Создание специальной компоновки рабочего пространства

Помимо настройки параметров в диалоговом окне Options, следует также иметь в виду, что каждая панель Expression Blend IDE (Tools, Properties, Objects and Timeline и пр.) может быть расположена где угодно в пределах рабочего пространства данной интегрированной среды разработки. Так, если требуется расположить панель Tools у верхнего, а не у левого края рабочего окна Expression Blend IDE, достаточно поместить курсор на области "захвата" над выбранной панелью и пристыковать ее к верхнему краю рабочего окна Expression Blend IDE стандартным методом перетаскивания мышью (попробуйте сделать это теперь же).

Область "захвата" имеется не у всех панелей, но и их можно перемещать в пределах рабочего пространства Expression Blend IDE, щелкнув на вкладке панели и перетащив ее на нужное место. В качестве примера попробуйте щелкнуть, не отпуская кнопку мыши, на вкладке Properties панели Properties и перетащить ее к одной из боковых сторон рабочего окна Expression Blend IDE или же на другую панель, чтобы добавить ее в текущий набор панелей.

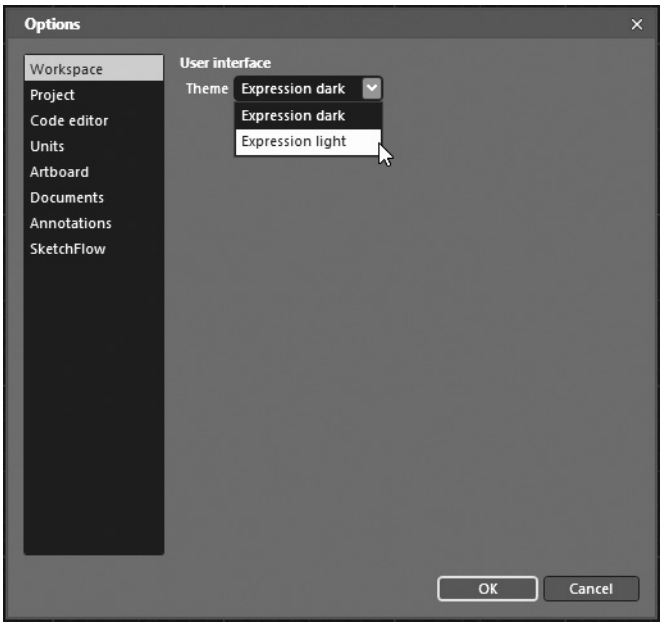


Рис. 1.33. В диалоговом окне Options можно настраивать различные параметры и режимы работы среды Expression Blend IDE

Расположив панели наиболее удобным для вас способом, можете по желанию сохранить свою компоновку среды Expression Blend IDE в качестве специального рабочего пространства по команде меню Window⇨Save as New Workspace (Окно⇨Сохранить как новое рабочее пространство). Выбрав эту команду из главного меню, можете далее присвоить сохраняемому рабочему пространству имя в открывшемся диалоговом окне, чтобы легко найти его в списке рабочих пространств, доступном по команде меню Window⇨Workspaces (Окно⇨Рабочие пространства), как показано на рис. 1.34.

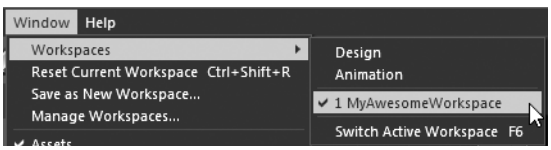


Рис. 1.34. Обнаружение и загрузка специального рабочего пространства

Как видите, два первых варианта выбора в списке рабочих пространств, доступном по команде меню Window⇨Workspaces, представляют собой два стандартных рабочих пространства Expression Blend IDE: Design (Конструирование) и Animation (Анимация). Если вы вдруг запутаетесь и потеряетесь в среде Expression Blend IDE, то можете всегда вернуться к ее исходной компоновке, выбрав стандартное рабочее пространство Design.

Система документации по Expression Blend

В заключение следует отметить, что вместе с программным продуктом Expression Blend поставляется специально составленное для него руководство пользователя, доступное по команде меню Help⇨User Guide (Справка⇨Руководство пользователя). К этой

системе документации полезно обращаться во время чтения данной книги. Так, в руководстве пользователя можно найти раздел, в котором описывается каждый клавиатурный эквивалент команды (этот раздел выделен на рис. 1.35), прочитать учебный материал по различным компонентам Expression Blend, подробно изучить особенности функционирования компонента SketchFlow и найти много другой информации, полезной для разработки приложений.

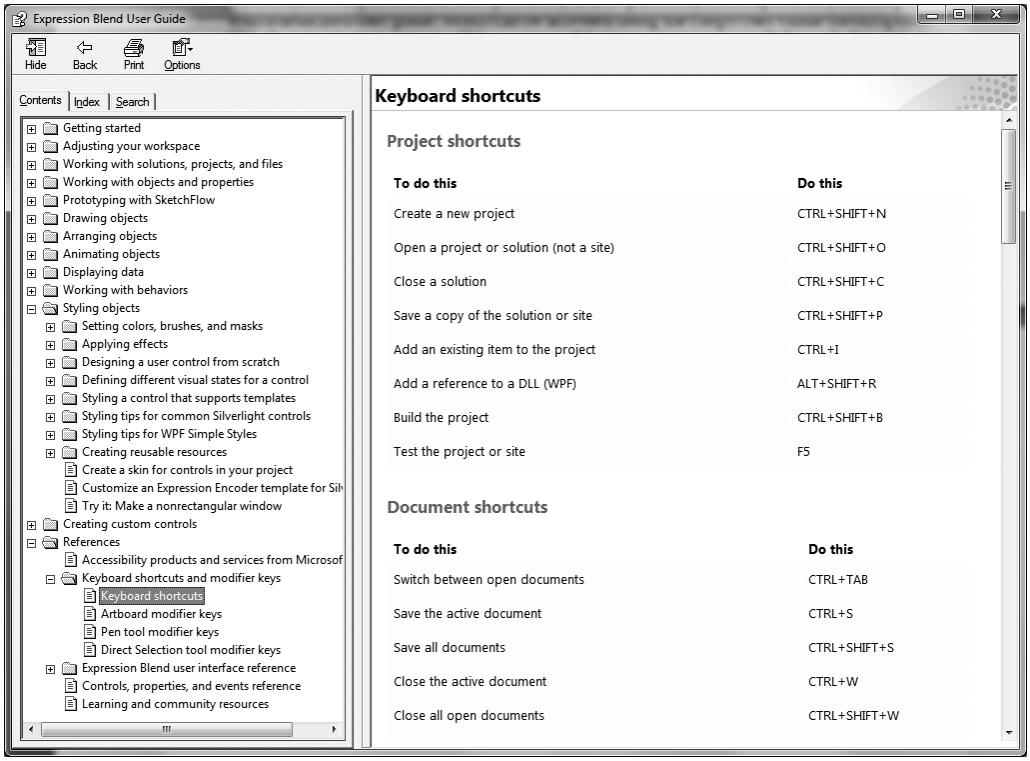


Рис. 1.35. Документация по Expression Blend очень полезна и заслуживает того, чтобы часто обращаться к ней за справкой!

Различные свойства справочной системы Expression Blend будут еще не раз упоминаться в этой книге, но вы должны самостоятельно изучить документацию по Expression Blend. Время, потраченное на ее чтение, окупится впоследствии сторицей!

Резюме

На этом вводная глава, дающая общее представление о среде Expression Blend IDE, завершается. В этой главе преследовалась цель ознакомить вас с основными компонентами Expression Blend IDE по следующему принципу: дальнее путешествие всегда начинается с первого шага. Сначала вы ознакомились с назначением каждого члена семейства программных продуктов Microsoft Expression: Web, Encoder, Design и Blend, а затем сразу же перешли к первоначальному освоению среды Expression Blend IDE на готовом примере проекта.

Монтажный стол служит основным местом для разработки любого нового проекта WPF или Silverlight в среде Expression Blend IDE. С помощью панели Tools, библиотеки

ресурсов и связанной с ней панели Assets, а также панели Properties вы сможете добавлять элементы пользовательского интерфейса в корневой диспетчер компоновки, настраивать и оформлять их графически по собственному усмотрению.

Панель Objects and Timeline служит главным местом для работы в среде Expression Blend IDE, поскольку исходная разметка в коде XAML отображается на ней в виде привычного для разработчиков иерархического визуального представления. Напомним, что для быстрого доступа к отдельным объектам с целью их правки на монтажном столе достаточно выбрать соответствующий узел в иерархическом представлении. Далее в книге будет показано, каким образом с помощью панели Objects and Timeline создаются сложные анимационные последовательности.

В среду Expression Blend IDE встроены два редактора исходного кода. Перейдя к редактору XAML, вы сможете набрать вручную разметку, описывающую создаваемый вами пользовательский интерфейс, или же поправить разметку, автоматически формируемую в коде XAML непосредственно в Expression Blend. А дважды щелкнув на файле исходного кода C# или VB на панели Projects, вы откроете редактор исходного кода, который оказывается очень полезным для ввода простого первоначального кода в кодовых блоках автоматически генерируемых обработчиков событий. Но не забывайте, что большую часть кода C# или VB вам, скорее всего, придется писать в среде Visual Studio 2010, где формат разрабатываемого проекта точно такой же, как и в среде Expression Blend.

В следующей главе будет рассмотрено назначение интерактивной векторной графики — главной составляющей любого приложения на платформе WPF или Silverlight.

ГЛАВА 2

Векторная графика и ресурсы объектов

В этой главе рассматриваются различные инструментальные средства, предоставляемые в Expression Blend в помощь разработчикам для создания интерактивной векторной графики. Сначала будут представлены основные инструменты рисования, доступные в среде Expression Blend IDE, в том числе Pen, Pencil и редактор кистей. А по ходу чтения вы сможете научиться стилизовать "перо", используемое для обрамления границ визуализируемых геометрических форм, применять различные визуальные эффекты и внедрять графические данные, формируемые средствами Expression Design.

Далее будут рассмотрены различные инструменты и способы работы с графическими преобразованиями и трехмерной графикой в среде Expression Blend IDE. При этом вы сразу же обнаружите, что на платформах WPF и Silverlight поддерживаются различные степени обработки трехмерной графики, и сможете по достоинству оценить каждую из них.

И в заключение будет рассмотрено, на первый взгляд, не связанное с основной ее темой понятие *ресурсов объектов*. Как станет ясно из описания ресурсов объектов, они представляют собой именованные большие двоичные объекты разметки, которые можно неоднократно использовать в приложениях. В среде Expression Blend IDE предоставляется целый ряд способов манипулирования собственными ресурсами, правки уже имеющихся и создания новых ресурсов. Несмотря на то что векторная графика относится к числу наиболее распространенных видов ресурсов объектов, при проработке материала остальных глав этой книги вы научитесь также фиксировать другие виды графического содержимого, в том числе стили и шаблоны, в качестве неоднократно используемых ресурсов объектов.

Царство векторной графики

В приложениях на платформах WPF и Silverlight все графические данные визуализируются с использованием *векторной графики*. Но применение векторной графики не ограничивается только специальными геометрическими формами, создаваемыми инструментом Pen или Pencil. Оказывается, в прикладных интерфейсах WPF API и Silverlight API элементы управления пользовательского интерфейса (кнопки, текстовые поля, меню, таблицы данных и прочее) также визуализируются с использованием векторной графики.

Векторная графика предоставляет целый ряд преимуществ, первым из которых является независимость от разрешения. При разработке приложения на платформе WPF или Silverlight можете быть полностью уверены в том, что данные пользовательского ин-

терфейса, а также специальные графические данные будут четко отображаться независимо от размеров просмотрового экрана. Так, если приложение Silverlight выполняется на мобильном телефоне под управлением ОС Windows Mobile 7, его пользовательский интерфейс автоматически масштабируется в соответствии с размерами маленького экрана мобильного телефона. А если то же самое приложение Silverlight выполняется в окне браузера на настольном компьютере пользователя, его пользовательский интерфейс соответственно масштабируется и четко отображается на крупном экране монитора независимо от разрешения последнего.

Применение векторной графики оказывается особенно эффективным при конструировании элементов управления пользовательского интерфейса (и в диспетчерах компоновки, которые содержат подобные элементы). В частности, всего лишь в нескольких строках разметки можно определить блочную панель (объект типа `StackPanel`) с элементам управления, которые визуализируются под углом 45°, вверх дном или на трехмерной плоскости. Кроме того, в приложениях на платформах WPF и Silverlight поддерживаются многочисленные визуальные эффекты (падающие тени, завихрения, размывания и пр.), которые можно применять к векторным графическим данным, не опасаясь ухудшения окончательно визуализируемого результата.

Читая эту главу, вы ознакомитесь с целым рядом инструментов, доступных в Expression Blend для создания, видоизменения и преобразования векторной графики. Несмотря на то что для разработки большинства приложений WPF и Silverlight вполне пригодны инструменты рисования, предоставляемые в Expression Blend, пользоваться только ими для создания графических данных с высокой степенью детализации, возможно, будет неудобно и затруднительно. Правда, в инструментальном средстве Expression Design, рассматриваемом далее в главе, для создания элементов векторной графики предоставляется полный набор эффективных инструментов, аналогичных по своим возможностям инструментам из приложения Adobe Illustrator. Как будет показано далее, художник-оформитель может экспортировать графические данные из Expression Design в формате XAML и внедрять их, а также взаимодействовать с их разметкой непосредственно в проекте, разрабатываемом в Expression Blend (или в Visual Studio 2010).

Повсеместное применение графических данных

Если вы перешли на WPF или Silverlight с другой платформы для разработки пользовательского интерфейса приложений, например Windows Forms, ASP.NET или иной среды, у вас может возникнуть невольный вопрос: а нужно ли вообще иметь дело с графическими данными? В частности, у вас может сложиться мнение о том, что типичное коммерческое приложение, полностью оснащенное системами меню, таблицами данных и специальными диалоговыми окнами, не требует особого графического оформления. И хотя такая точка зрения вполне правомерна, следует все же иметь в виду, что применение векторной графики глубоко укоренено на платформах WPF и Silverlight и обнаруживается в самых неожиданных местах.

В частности, ясное представление о том, как следует манипулировать графикой, имеет решающее значение для создания особого стиля оформления ряда элементов управления. Графика и анимация применяется также при создании шаблонов элементов управления, специальной настройке операций привязки данных (с помощью соответствующего шаблона) или внедрении визуальных подсказок для конечного пользователя (например, эффекта свечения активной области ввода текста). Более того, как будет показано далее в главе, в Expression Blend предоставляется целый ряд способов оперативного формирования специальных элементов управления с использованием векторной графики в качестве отправной точки.

Таким образом, ясное представление о том, как следует манипулировать графическими данными, служит прочным основанием для разработки многих реальных проектов, несмотря на то, что вам, возможно, придется не так уж и часто оформлять свои приложения WPF и Silverlight такими графическим элементами, как, например, произвольный зеленый кружок. Поэтому рассмотрение данной темы начнем с применения основных инструментов рисования, доступных в Expression Blend, для формирования специальных векторных геометрических форм.

Исследование возможностей основных инструментов рисования

В среде Expression Blend IDE определено пять основных инструментов, с помощью которых можно создавать простейшие геометрические формы. К их числу относятся инструменты Pen, Pencil, Ellipse, Rectangle и Line, и все они доступны на панели Tools (см. главу 1). Для ознакомления с этими инструментами (и связанными с ними средствами, включая редактор кистей) запустите Expression Blend на выполнение и создайте новый проект приложения WPF или Silverlight, присвоив ему имя `BlendDrawingTools`. Тип создаваемого проекта на самом деле не имеет особого значения, поскольку основные инструменты рисования действуют одинаково как в проектах WPF, так и Silverlight. Но в данном случае рекомендуется выбрать для примера проект приложения на платформе WPF, как показано на рис. 2.1.

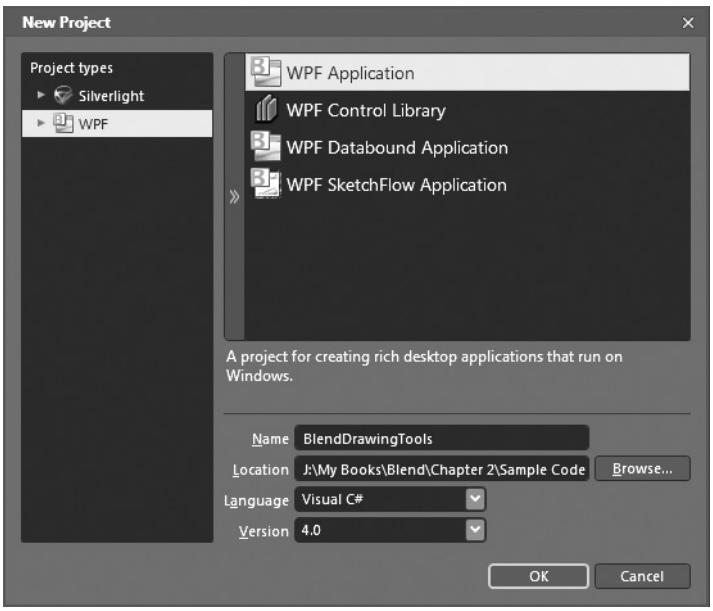


Рис. 2.1. Создание нового проекта приложения на платформе WPF

Работа с инструментом Pencil

В предыдущей главе инструменты Pen и Pencil уже рассматривались в самых общих чертах. Напомним, что инструментом Pencil можно рисовать от руки штриховые рисунки, как карандашом, а инструментом Pen — дуги и соединяемые точками прямые и кривые линии, как пером. Итак, найдите эти инструменты на панели Tools

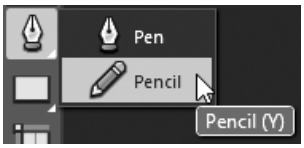


Рис. 2.2. Инструменты Pen и Pencil, доступные на панели Tools

(рис. 2.2) и выберите инструмент Pencil. С другой стороны, инструмент Pencil можно быстро выбрать, нажав оперативную клавишу <Y>.

Инструментом Pencil можно рисовать произвольные геометрические формы на монтажном столе, удерживая нажатой левую кнопку мыши и перемещая курсор данного инструмента. Выполняя эту операцию, вы непременно заметите, что в том месте, где помещается курсор инструмента Pencil, визуализируется отдельный элемент изображения (так называемый *пиксель*). С помощью инструмента Pencil можно определить произвольную разомкнутую геометрическую форму или же соединить начальную и конечную точки для создания замкнутой многоугольной формы.

Завершив рисование нескольких геометрических форм по собственному усмотрению, активизируйте инструмент Direct Selection, нажав оперативную клавишу <A>. Напомним, что прямое выделение формы инструментом Direct Selection дает возможность видоизменять отдельные участки, из которых состоит данная форма, по точкам контура, как показано на рис. 2.3. С другой стороны, обычное выделение инструментом Selection, который активизируется с помощью оперативной клавиши <V>, дает возможность выделить весь графический элемент для изменения его положения, размеров или иного вида преобразования.

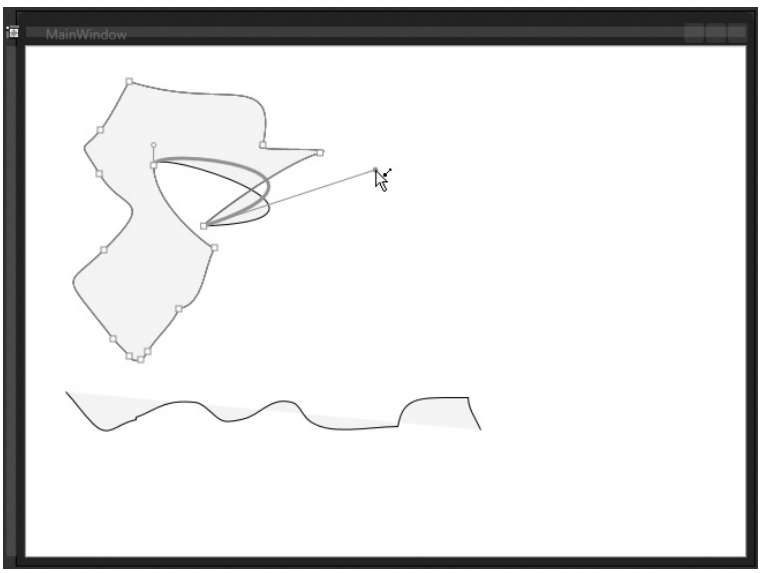


Рис. 2.3. Прямое выделение формы инструментом Direct Selection дает возможность видоизменять отдельные участки геометрической формы по точкам контура

Работа с инструментом Pen

Если работать с инструментом Pencil относительно просто, то работать с инструментом Pen, как ни странно, немного труднее. Итак, выберите инструмент Pen, нажав оперативную клавишу <P>. Главное отличие инструмента Pen от родственного ему инструмента Pencil заключается в том, что при каждом перемещении мыши данные отдельных пикселей *не* регистрируются инструментом Pen. Вместо этого инструментом Pen создается ряд соединяемых отрезков после каждого щелчка кнопкой мыши.

В качестве упражнения активизируйте инструмент Pen, если вы этого еще не сделали, перейдите к пустому участку монтажного стола и щелкните пять или шесть раз левой кнопкой мыши. Создав подобным образом несколько отрезков линий, можете затем видоизменить их контур разными способами, используя инструмент Pen. Так, если вы нажмете клавишу <Alt>, пользуясь инструментом Pen, то можете выделить (и захватить) уже существующую активную точку, чтобы преобразовать отрезок линии в отрезок дуги при перемещении курсора данного инструмента, как показано на рис. 2.4.

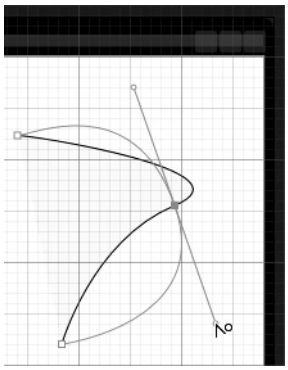


Рис. 2.4. Превращение отрезка линии в отрезок дуги инструментом Pen при одновременно нажатой клавише <Alt>

Следует также иметь в виду, что если вы выберете инструмент Pen и поместите его курсор на уже существующей активной точке, курсор изменит свой вид на знак “минус”. Если вы далее щелкнете на этой точке, она будет удалена из контура. Если же вы хотите добавить к геометрической форме дополнительную активную точку, щелкните на нужном месте текущего отрезка линии.

Инструмент Pen реагирует на целый ряд операций, инициируемых нажатием клавиш и допускающих дальнейшее видоизменение контура геометрической формы. Подробнее об этом можно узнать, обратившись за справкой к разделу “Инструмент Pen, модифицирующие клавиши” (Pen tool, modifiers) по предметному указателю руководства пользователя Expression Blend (рис. 2.5).

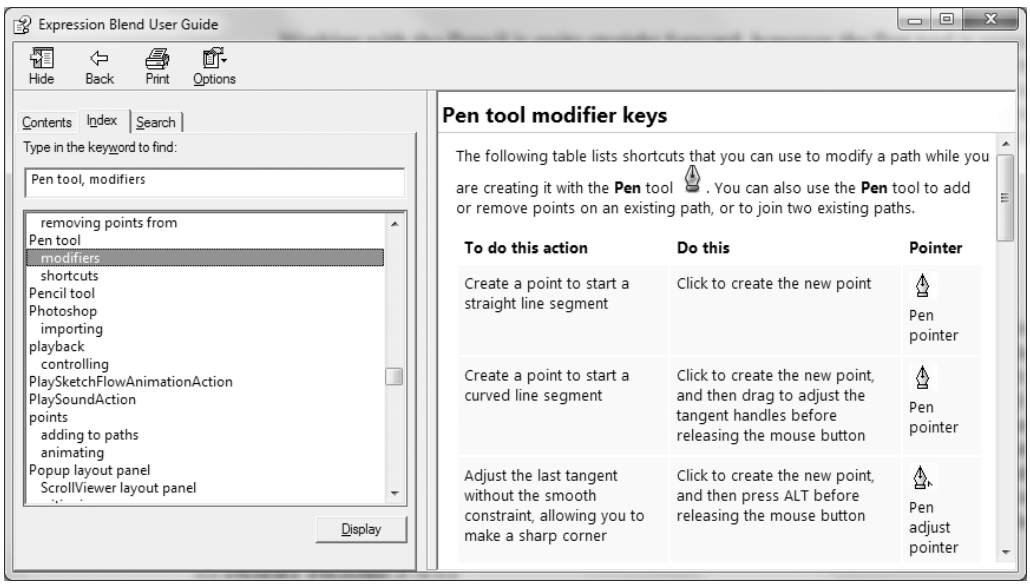


Рис. 2.5. Полный перечень операций, инициируемых нажатием модифицирующих клавиш во время работы с инструментом Pen

Результаты применения инструментов Pen и Pencil

При создании новых геометрических форм инструментами Pen и Pencil в Expression Blend фиксируются операции, инициируемые перемещением курсора, нажатием кнопок мыши и клавиш для создания нового объекта класса `Path`. А на платформах WPF и

Silverlight предоставляются классы, представляющие наиболее распространенные геометрические формы, причем все эти классы находятся в пространстве имен `System.Windows.Shapes`. В классе `Path` определяется свойство `Data`, где в конечном итоге сохраняется коллекция различных объектов геометрической формы, определяющих размеры и форму отдельного графического элемента. Для подобных целей на платформах WPF и Silverlight предоставляются такие классы, как `RectangleGeometry`, `EllipseGeometry`, `PathGeometry`.

Но если вы посмотрите исходную разметку XAML объектов типа `Path`, автоматически создаваемых в среде Expression Blend, то ожидаемых объектов геометрической формы вы в ней *не* обнаружите. Вместо них вы увидите длинное строковое значение, присвоенное свойству `Data` и состоящее из целого ряда чисел и буквенных знаков вроде `M`, `C` и `Z`, как показано ниже.

```
<Path Data="M254,101 C297.83333,110.83333 330.5,191.50008 385.5,130.5
   440.5,69.499921 456.83333,142.5 492.5,148.5 419.5,159.83333
   229.5,236.5 273.5,182.5 317.5,128.5 242.83333,164.5 227.5,155.5 z"
   Fill="#FFF4F4F5" Margin="227.5,101,130.5,0"
   Stretch="Fill" Stroke="Black" Height="101.57"
   VerticalAlignment="Top"/>
```

Это длинное строковое значение представляет собой компактную форму описания объектов геометрической формы, используемых для визуализации объекта типа `Path`, на так называемом мини-языке описания контуров. К счастью, вам не придется обрабатывать эти строковые данные вручную. Но в то же время вам не помешает знать, что в столь сжатой строковой форме представлен целый ряд инструкций по выводу результатов визуализации объекта типа `Path`[1].

Примечание. Когда вы экспортируете специальные графические данные средствами Expression Design, язык моделирования контуров в XAML не применяется, а вместо него используется более крупная объектная модель. Но, как правило, именно это и требуется, поскольку объектами геометрической формы можно легко манипулировать в коде.

Работа с инструментами Rectangle, Ellipse и Line

Разумеется, любую геометрическую форму можно создать и с помощью инструментов Pen и Pencil. Тем не менее в Expression Blend предоставляется ряд дополнительных инструментов для создания стандартных геометрических форм. Так, с помощью инструмента Rectangle, выбираемого нажатием оперативной клавиши <M>, нетрудно создать прямоугольник; с помощью инструмента Ellipse, выбираемого нажатием оперативной клавиши <L>, — эллипс, а с помощью инструмента Line, выбираемого нажатием оперативной клавиши <\>, — линию. Работая с этими инструментами, необходимо иметь в виду следующие их особенности.

- Если вы нажмете клавишу <Alt> после выбора одного из этих инструментов, центральной точкой геометрической формы окажется первая точка, на которой вы щелкнете, а не левый верхний угол данной формы.
- Если вы нажмете клавишу <Shift>, перетаскивая курсор инструмента Rectangle или Ellipse, полученная в итоге геометрическая форма будет иметь одинаковые размеры по ширине и по высоте.

---

[1] Если вас интересуют более подробные сведения о мини-языке описания контуров, обратитесь к документации по .NET 4.0 SDK, выполнив поиск по критерию *path markup syntax* (синтаксис разметки контуров).

- Если вы нажмете клавишу <Shift>, перетаскивая курсор инструмента Line, угол наклона рисуемой линии будет ограничиваться величиной, кратной 15. Благодаря этому упрощается рисование идеально прямых линий под разным углом.

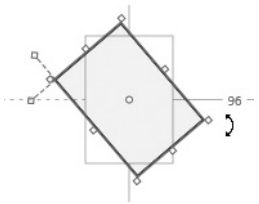


Рис. 2.6. На монтажном столе можно выполнять различные преобразования над выделенными графическими элементами

Создав геометрическую форму любым из рассматриваемых здесь инструментов, можете далее воспользоваться инструментами Selection и Direct Selection для видоизменения ее размеров и положения. А если вы поместите курсор мыши за пределами вытягиваемого маркера ограничивающей рамки, то сможете выполнить ряд простых преобразований геометрической формы, в том числе скашивание и вращение. Подробнее о графических преобразованиях речь пойдет далее в этой главе, а пока рассмотрите рис. 2.6, на котором показано, каким образом осуществляется вращение графического объекта типа `Rectangle` за пределами вытягиваемого углового маркера ограничивающей рамки.

Применение категории Shapes библиотеки ресурсов

Помимо инструментов Rectangle, Line и Ellipse, для создания соответствующих стандартных геометрических форм можно также воспользоваться категорией Shapes библиотеки ресурсов, рассматривавшейся в главе 1. В этой категории определен целый ряд полезных предварительно заданных геометрических форм[2]. Как показано на рис. 2.7, в категории Shapes библиотеки ресурсов можно выбрать различные виды стрелок, выносок и других распространенных форм, включая треугольники, пятиугольники и пр.

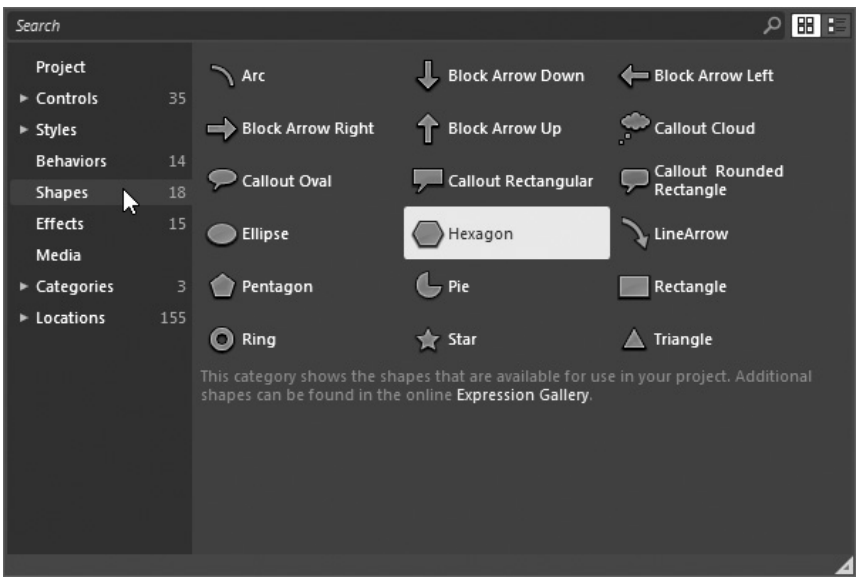


Рис. 2.7. В библиотеке ресурсов предоставляется целый ряд дополнительных готовых геометрических форм

[2] Это новое для Expression Blend 4 средство. В предыдущих версиях этой интегрированной среды разработки наборы типичных геометрических форм отсутствовали.

В качестве простого упражнения выберите графический элемент `Star` из библиотеки ресурсов. Соответствующий инструмент рисования формы звезды появится на панели Tools, как показано на рис. 2.8.

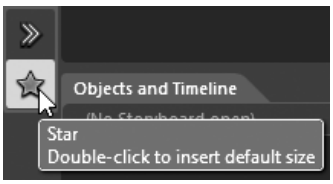


Рис. 2.8. Графические элементы, выбираемые из библиотеки ресурсов, появляются в виде инструментов на выбранном в последний раз участке панели Tools

Примечание. Назначение библиотеки ресурсов, выбираемой на панели Tools с помощью кнопки, обозначенной пиктограммой со знаком >>, уже пояснялось в главе 1. Но и теперь не грех повторить следующее: когда вы выбираете элемент из библиотеки ресурсов, ранее выбранный элемент появится рядом с пиктограммой со знаком >>. Это удобно на тот случай, если данный элемент потребуется использовать впоследствии.

Используя эти дополнительные геометрические формы, вы косвенным образом добавляете в свой проект ссылку на новую библиотеку `Microsoft.Expression.Drawing.dll`. В этой библиотеке определяется ряд классов для упомянутых выше дополнительных специальных форм. Например, для рисования формы звезды инструментом Star используется класс `RegularPolygon`, а различные формы выносок представлены классом `Callout`. Так, если ввести формы звезды и прямоугольной выноски на монтажном столе, эти новые графические элементы, выделенные на рис. 2.9, появятся на панели Objects and Timeline.

Примечание. Ссылки на сборки Expression Blend SDK можно делать и в проекте, разрабатываемом в Visual Studio, на вкладке .NET диалогового окна Add References (Добавление ресурсов). Кроме того, сборки Expression Blend SDK будут разворачиваться по умолчанию как закрытые, и поэтому в указанном выходном каталоге будут содержаться только локальные копии этих библиотек.

Видоизменение формы в редакторе внешнего вида

Все рассматривавшиеся до сих пор формы (прямоугольника, звезды, линии и пр.) могут быть выбраны и настроены на панели Properties. Разумеется, конкретные свойства отображаются на этой панели в зависимости от типа графического элемента, выбранного на монтажном столе. Тем не менее на панели Properties доступен ряд настраиваемых свойств, общих для всех геометрических форм.

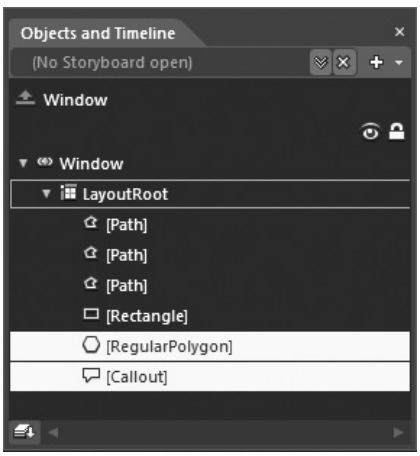


Рис. 2.9. Геометрические формы из библиотеки ресурсов представлены особыми классами

Допустим, вы нарисовали форму звезды инструментом Star. Если вы выберете этот графический элемент на монтажном столе и перейдете к области Appearance (Внешний вид) на панели Properties, то обнаружите там ряд свойств, с помощью которых можно, в частности, настроить толщину обводки границы данной формы, а также степень ее непрозрачности. Кроме того, на панели Properties доступен ряд свойств, специфических для геометрической формы звезды, в том числе количество точек (`PointCount`) и внутренний радиус (`InnerRadius`), как показано на рис. 2.10.

Рис. 2.10. В редакторе внешнего вида из области Appearance на панели Properties доступен целый ряд свойств, как общих, так и специфических для выбранной геометрической формы

Если бы вы выбрали форму прямоугольной выноски на монтажном столе, то в редакторе внешнего вида из области Appearance на панели Properties появились бы другие свойства, характерные для данной геометрической формы и соответствующего ей объекта класса `CalloutStyle`. На рис. 2.11 показаны настроенные геометрические формы звезды и выноски после изменения их различных свойств в редакторе внешнего вида.

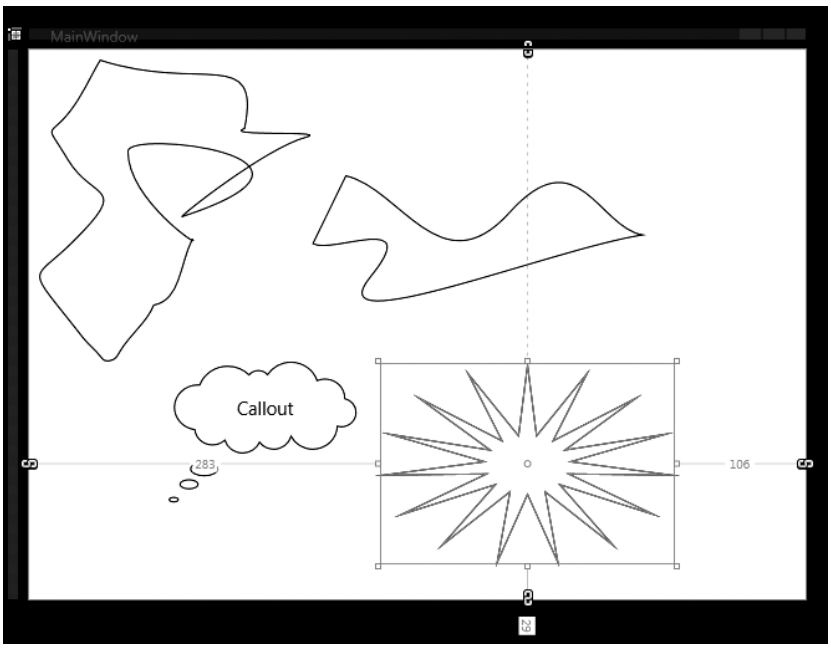


Рис. 2.11. Некоторые примеры специально настроенных геометрических форм

Следует, однако, иметь в виду, что содержимое редактора внешнего вида будет изменяться в зависимости от того, какой именно графический элемент выбран на монтажном столе. Это справедливо для каждого редактора, доступного на панели Properties. Читая эту книгу, вы еще не раз встретитесь с целым рядом свойств, которые могут быть настроены на панели Properties. Но самое лучшее, что можно посоветовать в отношении свойств графических элементов, — это выбрать интересующий вас элемент на монтажном столе и тщательно исследовать его настраиваемые свойства на панели Properties.

Примечание. Если вы выберете на монтажном столе несколько графических элементов, нажав клавишу <Shift> и щелкнув сначала на первом, а затем на последнем выбираемом элементе, то на панели Properties вы увидите только свойства, общие для всех выбранных элементов. Это оказывается удобным в том случае, если требуется, например, задать одинаковую высоту и ширину сразу для нескольких графических элементов.

Раскраска форм в редакторе кистей

В редакторе кистей, доступном в области Brushes на панели Properties, вы можете подбирать цвета для рисования границ элементов пользовательского интерфейса, а также для заполнения их внутреннего пространства. Этот редактор пригоден для работы с любым элементом пользовательского интерфейса, включая формы, элементы управления, диспетчеры компоновки или целые объекты типа `Window` на платформе WPF либо объекты типа `UserControl` на платформе Silverlight. Независимо от типа настраиваемого графического элемента, общие операции, выполняемые в редакторе кистей, остаются неизменными. В качестве примера, допустим, что вас интересует видоизменение цветов раскраски шестиугольника, введенного на монтажном столе из категории Shapes библиотеки ресурсов.

Просмотр свойств раскраски кистью

В самой верхней области редактора кистей находится список свойств раскраски кистью графического элемента, выделенного на монтажном столе. Как показано на рис. 2.12, для шестиугольника, представленного базовым объектом типа `RegularPolygon`, доступны три таких свойства: `Fill` (Заливка), `Stroke` (Обводка) и `OpacityMask` (Маска непрозрачности). Щелкнув на любом из этих свойств, вы сможете настроить соответствующую кисть, используемую для раскраски.

Рис. 2.12. В самой верхней области редактора кистей находится список свойств раскраски кистью выделенного графического элемента

Выбор режима раскраски No brush

Непосредственно под этой областью находятся пять выделяемых вкладок, как показано на рис. 2.13, где можно выбрать общий вид кисти. Если выбрать крайнюю слева вкладку No brush (Без кисти), то свойства раскраски кистью можно настроить на использование, по существу, прозрачной кисти. В этом случае сквозь выделенный на монтажном столе графический элемент будет проявляться все, что находится под ним.

Выбор кисти для раскраски сплошным цветом

Далее по порядку следует вкладка Solid color brush (Кисть для раскраски сплошным цветом), позволяющая подобрать единственный сплошной цвет раскраски кистью для выбранного свойства, используя интуитивный селектор цвета. На рис. 2.13 (а также на цветной вклейке) показано, каким образом в этом редакторе кисти для раскраски сплошным цветом выбирается оттенок сплошного цвета морской волны для свойства `Fill` шестиугольника.

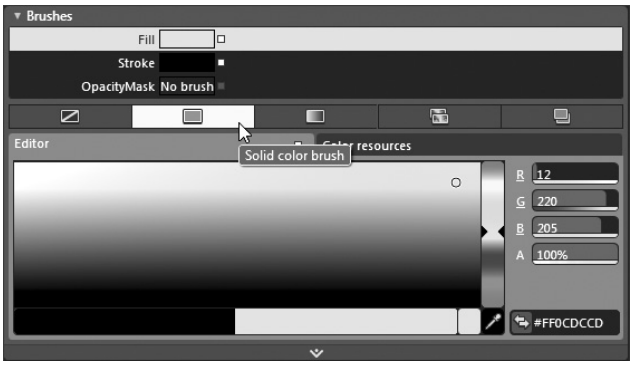


Рис. 2.13. Выбор кисти для раскраски сплошным цветом

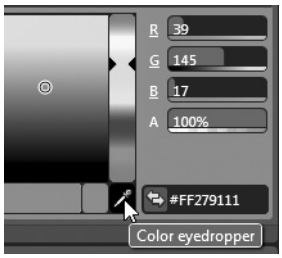


Рис. 2.14. С помощью инструмента Color Eyedropper можно отобрать цвет из любого элемента, на котором достаточно щелкнуть кнопкой мыши

Непременно уделите хотя бы немного времени исследованию различных функциональных возможностей редактора кисти для раскраски сплошным цветом. В частности, в правом нижнем углу окна этого редактора находится инструмент Color Eyedropper (Пипетка для отбора цвета) (рис. 2.14, а также цветная вклейка), с помощью которого вы сможете отбирать цвет из любого элемента, на котором щелкнули мышью, — даже из тех элементов, которые находятся за пределами рабочего окна Expression Blend IDE! Допустим, вам требуется отобрать цвет отдельной папки на рабочем столе Windows. Для этого достаточно выбрать инструмент Color Eyedropper и щелкнуть на интересующем вас графическом элементе непосредственно на рабочем столе Windows.

Выбор кисти для раскраски градиентом

Далее по порядку следует вкладка Gradient brush (Кисть для раскраски градиентом), позволяющая выбрать кисть, в которой для раскраски поверхности используется ряд смешивающихся цветов. С этой целью выберите сначала одну из геометрических форм на монтажном столе, затем свойство `Fill` в редакторе кистей и, наконец, щелкните на вкладке Gradient brush, как показано на рис. 2.15, а также на цветной вклейке.

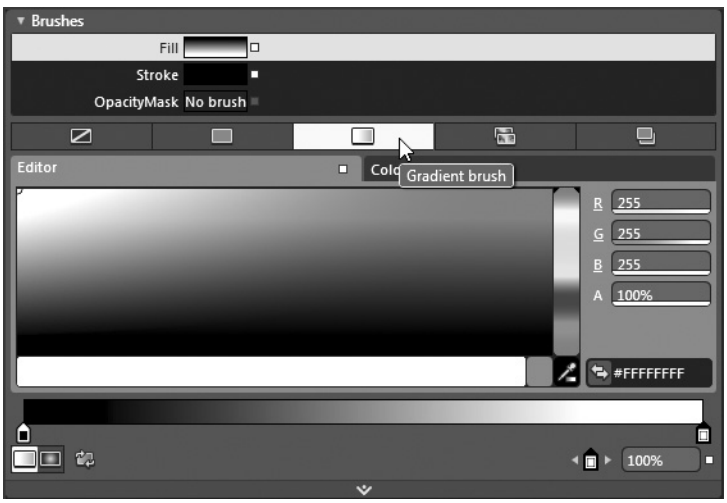


Рис. 2.15. Редактор кисти для раскраски градиентом

По умолчанию в редакторе кисти для раскраски градиентом предоставляются два маркера выбора цветов градиента, обозначающих два первых цвета градиента (исходно черный и белый) для любой кисти и представленных в виде ползунковых элементов управления, расположенных по краям редактируемой полосы градиента. Щелкнув на любом из ползунков на полосе градиента, можете изменить с помощью селектора цвет, используемый в данной части градиента. Кроме того, можете переместить ползунок вдоль редактируемой полосы градиента, определяя начальные и конечные значения цветов градиента. На рис. 2.16 (а также на цветной вклейке) в качестве примера показан один из возможных вариантов настройки градиента с помощью маркеров выбора первых двух его цветов.

Рис. 2.16. Настройка градиента с помощью маркеров выбора его цветов

По желанию можете добавить дополнительные маркеры выбора цветов в любом месте редактируемой полосы градиента. Для примера на рис. 2.17 (а также на цветной вклейке) показано применение четырех маркеров выбора цветов градиента для раскраски кистью, причем каждый из них установлен на отдельно подобранный цвет.

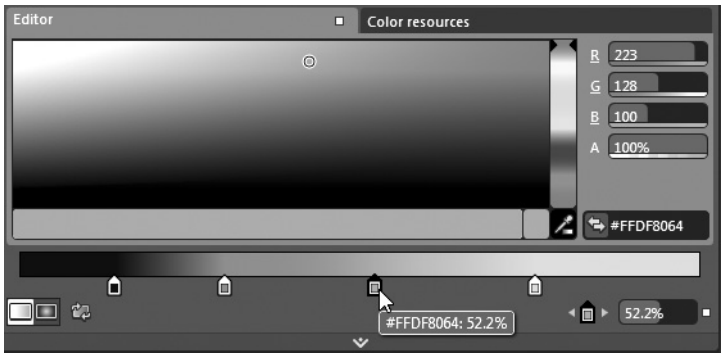


Рис. 2.17. Для того чтобы добавить дополнительные маркеры выбора цветов градиента, достаточно щелкнуть кнопкой мыши на редактируемой полосе градиента

Примечание. Если требуется удалить отдельный маркер выбора цвета градиента, щелкните на нем левой кнопкой мыши и, не отпуская ее, перетащите удаляемый маркер за пределы редактируемой полосы градиента.

В левом нижнем углу редактора кисти для раскраски градиентом находятся две кнопки, с помощью которых можно выбрать радиальный или линейный градиент для

раскраски кистью. Нетрудно догадаться, что эти кнопки определяют порядок смешения цветов в градиенте круговым или линейным способом. А прямо под этими кнопками находится еще одна кнопка, позволяющая изменить на обратное расположение всех маркеров выбора цветов на редактируемой полосе градиента, что очень удобно, как показано на рис. 2.18.

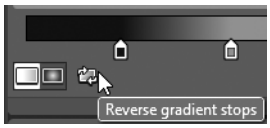


Рис. 2.18. Дополнительные средства редактора кисти для раскраски градиентом

И наконец, в редакторе кисти для раскраски градиентом используется инструмент Gradient, находящийся на панели Tools и выбираемый нажатием оперативной клавиши <G>. В качестве упражнения выберите на монтажном столе геометрическую форму со свойством `Fill`, в котором используется кисть для раскраски градиентом, а затем нажмите оперативную клавишу <G>. С этого момента вы можете определить точку начала отсчета градиента и переставить по своему усмотрению маркеры выбора цветов с помощью инструмента Gradient. В качестве примера на рис. 2.19 приведен один из возможных вариантов окончательной настройки кисти для раскраски градиентом.

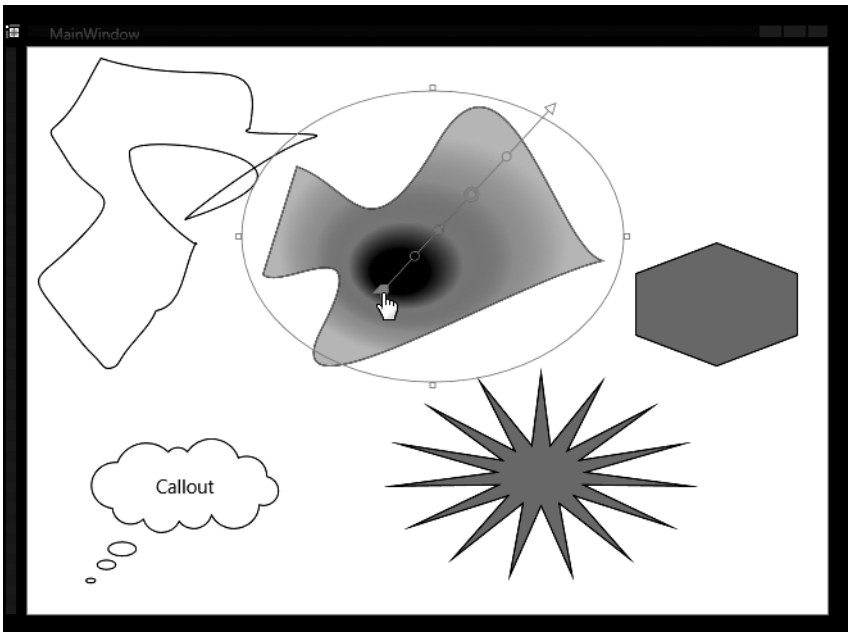


Рис. 2.19. С помощью инструмента Gradient можно задать точку начала отсчета градиента для раскраски кистью

Если вы посмотрите на разметку специально настроенной вами кисти в коде XAML, то обнаружите в ней много общего с приведенной ниже разметкой, где показана настройка кисти (объекта `RadialGradientBrush`) для раскраски радиальным градиентом. Если же вы выберете для раскраски линейный градиент, то соответствующая кисть будет представлена объектом `LinearGradientBrush`.

```
<RadialGradientBrush RadiusY="0.587" RadiusX="0.587"
                     GradientOrigin="0.386,0.662">
   <GradientStop Color="#FFCE1DD8" Offset="0.458"/>
   <GradientStop Color="#FFB0BBE9" Offset="0.747"/>
   <GradientStop Color="#FF2B022D" Offset="0.153"/>
   <GradientStop Color="#FFDF8064" Offset="0.522"/>
</RadialGradientBrush>
```

Выбор кисти для мозаичной раскраски

Последней в редакторе кистей и рассматриваемой здесь является вкладка Tile brush (Кисть для мозаичной раскраски). Этот тип кисти позволяет раскрашивать поверхность, исходя из данных изображения, находящегося в специально указанном внешнем графическом файле с расширением `.bmp`, `.tif`, `.jpeg` и т.д. Итак, выберите какую-нибудь другую геометрическую форму (или создайте новую) на монтажном столе, выделите ее и щелкните на вкладке Tile brush. А затем настройте свойство `ImageSource` этой формы, используя кнопку Choose an image (Выбрать изображение), как показано на рис. 2.20.

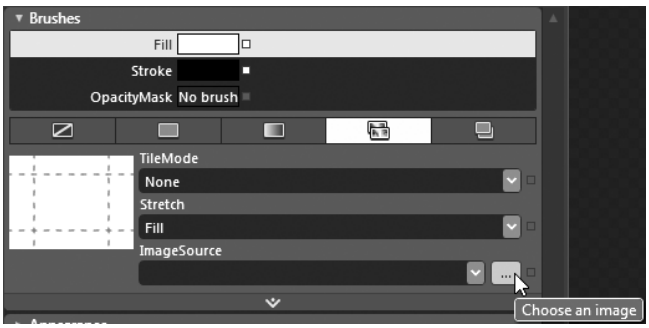


Рис. 2.20. Выбор источника для мозаичной раскраски кистью

Теперь найдите в открывшемся диалоговом окне файл любого исходного изображения на своем компьютере. Как только вы выберете подходящий файл изображения, он будет автоматически добавлен в текущий проект, а данные этого изображения будут использованы для создания новой кисти. Для примера на рис. 2.21 показана одна из геометрических форм, заполненная графическими данными из выбранного файла изображения формата BMP.

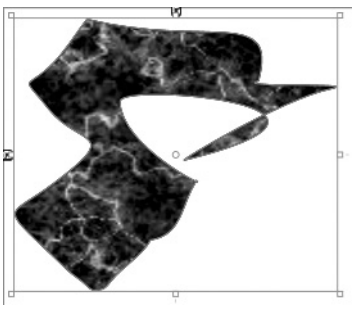


Рис. 2.21. Кистью для мозаичной раскраски можно заполнить геометрическую форму, используя данные исходного изображения

Надеюсь, теперь вы чувствуете себя достаточно уверенно, создавая кисти, задаваемые для различных свойств раскраски кистью (`Fill`, `Stroke` и пр.) в среде Expression Blend IDE. Далее будут рассмотрены способы объединения геометрических форм для образования новых объектов типа `Path`.

Объединение геометрических форм и извлечение контуров

В Expression Blend предоставляется возможность выполнять целый ряд операций, с помощью которых можно создавать новые объекты типа `Path`, объединяя уже существующие формы самыми разными способами. Для активизации подобных инструмен-

тальных средств необходимо, прежде всего, выбрать несколько графических элементов на монтажном столе, а еще лучше — несколько перекрывающихся так или иначе элементов. Для расположения двух или нескольких объединяемых графических элементов на монтажном столе таким образом, чтобы они действительно перекрывались, воспользуйтесь инструментом Selection. После этого выделите все эти элементы, нажав клавишу <Shift> и щелкнув сначала на первом из них, а затем на последнем.

Примечание. При формировании нового объекта типа `Path` в Expression Blend используется кисть, назначенная для графического элемента, выбранного *последним*. Поэтому убедитесь в том, что у выбранной последней геометрической формы имеется именно та кисть, которой вы хотели бы воспользоваться. Разумеется, вы всегда сможете обратиться к редактору кистей, чтобы внести изменения в кисть, которой предполагаете пользоваться после объединения графических элементов.

Далее щелкните правой кнопкой мыши на выделенных элементах и перейдите к пункту Combine (Объединить) всплывающего контекстного меню. Под этим пунктом доступен целый ряд полезных операций объединения геометрических форм (рис. 2.22).

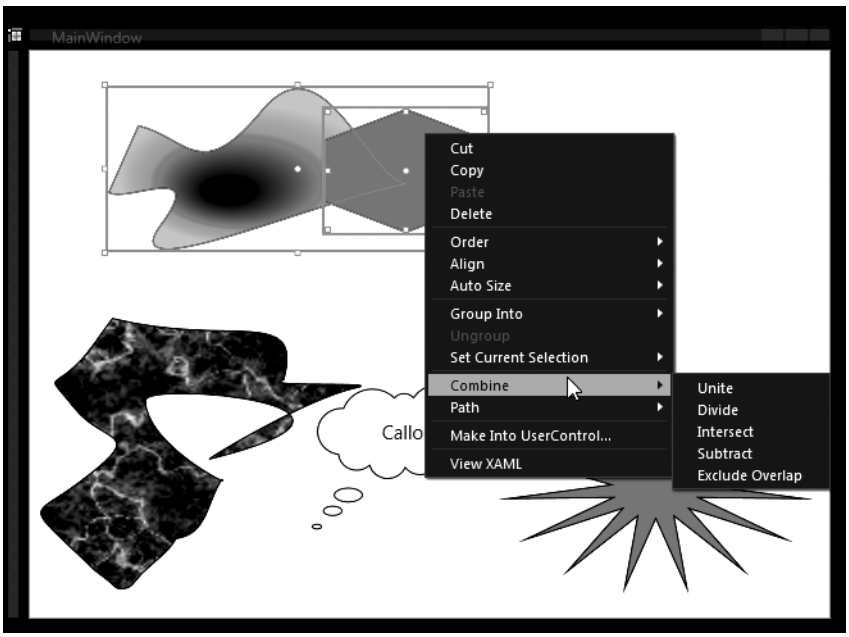


Рис. 2.22. Операции, выбираемые из пункта Combine всплывающего контекстного меню, позволяют создать новый объект `Path` из объединяемых разными способами существующих геометрических форм

Ниже приведено краткое описание операций объединения геометрических форм, выбираемых из пункта Combine всплывающего контекстного меню.

- Unite (Объединить). Соединяет все формы или контуры в единый объект.
- Divide (Разделить). Вырезает формы или контуры в месте их пересечения, оставляя все разделенные части без изменения.
- Intersect (Пересечь). Оставляет перекрывающиеся участки форм, удаляя неперекрывающиеся.

- Subtract (Вычесть). Вырезает все остальные выделенные формы из той, что выделена последней.
- Exclude Overlap (Исключить перекрытие). Оставляет неперекрывающиеся участки форм, удаляя перекрывающиеся.

Опробовав каждую из упомянутых выше операций объединения геометрических форм, вы сами убедитесь в том, что они достаточно интуитивны. Но имейте в виду, что в Expression Blend не поддерживаются стандартные для графических редакторов клавиатурные эквиваленты команд отмены последней операции (комбинация клавиш <Ctrl+Z>) и повтора последней операции (<Ctrl+Y>).

Преобразование формы в контур

В Expression Blend предоставляется еще один способ формирования нового объекта типа `Path` по команде Convert to Path. Это оказывается удобным в том случае, если геометрическая форма создана основными инструментами рисования, например Ellipse или Rectangle, или же построена на основании специальной формы, выбираемой из библиотеки ресурсов, и требует дальнейшего видоизменения отдельных ее линий или отрезков.

В качестве примера рассмотрим форму звезды, созданную ранее в этой главе. Если посмотреть на разметку этой геометрической формы в коде XAML, то она будет выглядеть аналогично приведенной ниже.

```
<ed:RegularPolygon Fill="#FF279111" InnerRadius="0.302"
    PointCount="15" Stretch="Fill" Stroke="Black"
    Margin="0,114,18,168"
    HorizontalAlignment="Right" Width="235"/>
```

Щелкните правой кнопкой мыши на этой (или аналогичной) форме на монтажном столе и выберите команду Path⇨Convert to Path (Контур⇨Преобразовать в контур) из всплывающего контекстного меню. В итоге объект типа `RegularPolygon` будет преобразован в объект типа `Path`, как показано в приведенной ниже разметке.

```
<Path Data="M117.5,0.5 L124.7254,57.65331 L165.35032,7.4490627
L137.92686,61.669108 L204.92688,27.094695 L147.59633,69.006338
L229.38655,56.039988 L152.06187,78.396325 L234.50001,89.280037
L150.55135,88.215455 L219.38311,121.06734 L143.32595,96.765911
L186.64969,145.90558 L131.63501,102.56924 L141.95966,159.49999
L117.5,104.622 L93.040342,159.49999 L103.36499,102.56924
L48.350318,145.90558 L91.674053,96.765911 L15.616902,121.06734
L84.448655,88.215455 L0.49999996,89.280037 L82.938135,78.396325
L5.6134615,56.039988 L87.403677,69.006338 L30.073123,27.094695
L97.073147,61.669108 L69.649686,7.4490627 L110.27461,57.65331 z"
Fill="#FF279111" HorizontalAlignment="Right"
Margin="0,114.25,18.25,168.25" Stretch="Fill" Stroke="Black"
Width="234.5"/>
```

Выбрав новый объект типа `Path`, вы можете далее выполнить его прямое выделение инструментом Direct Selection и видоизменить каждый элемент его геометрической формы. Для примера на рис. 2.23 показан результат существенного видоизменения формы звезды, преобразованной в контур.

Примечание. Такое применение инструмента Direct Selection было бы невозможным, если бы форма представляла собой правильный многоугольник. Ведь у правильного многоугольника отсутствуют отдельные точки контура, доступные для правки.

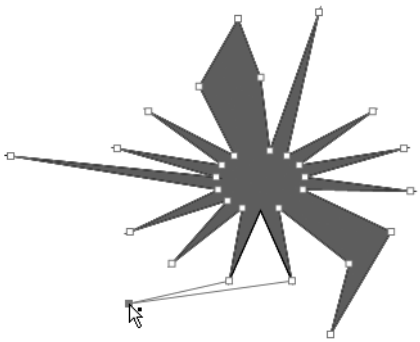


Рис. 2.23. Преобразование геометрической формы в контур оказывается полезным в том случае, если требуется видоизменить отрезки ее линий

Взаимодействие с формами

Не следует никогда забывать, что любая форма, создаваемая на монтажном столе средствами Expression Blend, на самом деле является объектом, с которым можно взаимодействовать в коде приложения. В частности, с геометрическими формами (прямоугольниками, эллипсами, линиями, выносками, контурами и пр.) связан целый ряд событий, которые могут быть обработаны в прикладных программах для организации взаимодействия с этими формами самыми разными способами.

Первым шагом на пути к взаимодействию с формами в коде должно стать присваивание им подходящих имен в текстовом поле Name на панели Properties. В качестве упражнения попробуйте сделать следующее: по очереди выделите формы инструментом Selection на монтажном столе, а затем присвойте каждой из них имя на панели Properties. Конкретные имена в данном случае особого значения не имеют, но в реальном проекте они должны стать удобными указателями на тип или иные особенности геометрической формы, как, например, `myCallout`, `myStar` и т.д. На рис. 2.24 в качестве примера демонстрируется присваивание имени `fancyShape` одной из геометрических форм.

Рис. 2.24. Именование объектов геометрических форм является первым шагом на пути к взаимодействию с ними в коде

Присвоив имя объекту геометрической формы, вы можете сразу же обнаружить, что в его определение в коде XAML добавлен атрибут `x:Name`, как показано в приведенной ниже строке кода.

```
<Path x:Name="fancyShape" ... />
```

Обработка событий

А теперь выделите любую форму на монтажном столе и щелкните на кнопке Events в правом верхнем углу панели Properties (пиктограмма этой кнопки обозначена знаком молнии). Найдите событие `MouseEnter` и введите имя `InsideAShape` метода обработки этого события, как показано на рис. 2.25.

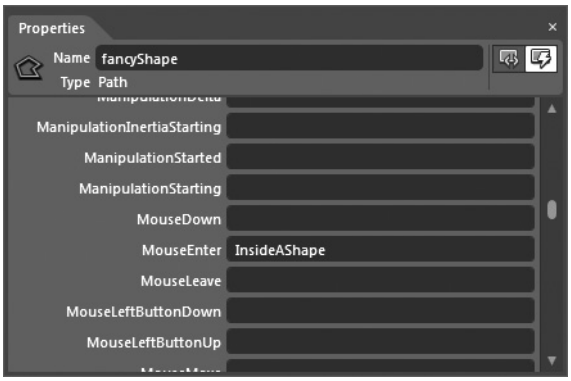


Рис. 2.25. Организация обработки события, связанного с выбранным графическим элементом

Как только вы нажмете клавишу <Enter>, в среде Expression Blend IDE будет сгенерирован приведенный ниже код обработчика событий, сохраняющийся в соответствующем файле исходного кода текущего проекта.

```
private void InsideAShape(object sender,
                    System.Windows.Input.MouseEventArgs e)
{
  // Что сделать: добавить здесь реализацию обработчика событий в коде.
}
```

В рассматриваемом здесь примере обрабатывается событие наведения курсора мыши (`MouseEnter`) на каждую из форм в текущем проекте, поэтому среди свойств каждой из них следует указать один и тот же метод `InsideAShape` обработки данного события. Этот метод будет вызываться независимо от того, на какую именно форму был наведен курсор мыши, а следовательно, первый входной параметр данного метода, называемый `sender` (отправитель) и имеющий тип `object`, можно использовать для указания конкретной геометрической формы, отправившей событие на обработку, т.е. той формы, в которой оказался курсор мыши.

Если вы занимаетесь программированием профессионально, то приведенный ниже код должен быть вам понятен. В этом коде входной параметр `sender` типа `object` сначала приводится к типу родительского класса `UIElement`, поскольку все геометрические формы и элементы управления на платформах WPF и Silverlight наследуют от класса `UIElement`, а затем значение свойства `Opacity` (Непрозрачность) изменяется до 50 процентов[3]. А если у вас нет особого опыта программирования, наберите приведенный ниже код, но помните, что в синтаксисе языка C# учитывается регистр!

```
private void InsideAShape(object sender,
                    System.Windows.Input.MouseEventArgs e)
{
  // Сделать выбранную в настоящий момент форму полупрозрачной.
```

[3] Если вы создаете текущий пример проекта на языке VB, придется воспользоваться оператором `DirectCast` вместо оператора приведения типов, применяемого в C#.

```
  ((UIElement)sender).Opacity = .5;
}
```

Если вы запустите теперь свое приложение на выполнение, нажав функциональную клавишу <F5>, то при наведении курсора на любую из форм она должна стать полупрозрачной.

Примечание. Если вы загрузили архив примеров проектов к этой книге с веб-сайта издательства Apress по адресу, указанному во введении, то обнаружите, что в коде рассматриваемого здесь примера проекта обрабатывается также событие покидания курсора мыши (`MouseLeave`) каждой геометрической формы. В общем для всех форм обработчике событий типа `MouseLeave` устанавливается исходное значение **`1,0`** свойства `Opacity`. Благодаря этому непрозрачность каждой формы восстанавливается, когда курсор мыши перемещается за ее пределы.

Настройка “перьев”

До сих пор вы пользовались редактором кистей для создания специально настраиваемых кистей, которыми внутреннее пространство геометрических форм заполняется с помощью свойства `Fill`. В Expression Blend имеется также возможность настраивать “перья”, которыми обводятся границы выбранных элементов пользовательского интерфейса. Когда на монтажном столе рисуется форма, ее граница по умолчанию обводится кистью черного цвета толщиной в один пиксель.

Если хотите изменить данную исходную настройку, то придется сначала внести изменения в кисть, выбранную для свойства `Stroke` в редакторе кистей, описанным ранее способом. Кроме того, можете изменить свойство `StrokeThickness` (Толщина обводки), доступное в области Appearance на панели Properties. И наконец, в вашем распоряжении имеется целый ряд свойств обводки, доступных в области дополнительных параметров настройки редактора внешнего вида (рис. 2.26).

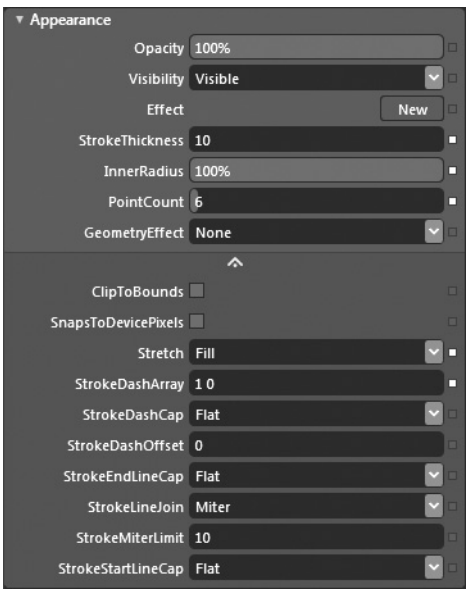


Рис. 2.26. В редакторе внешнего вида имеется целый ряд дополнительных свойств, с помощью которых можно определить порядок обводки границ форм и линий

Выбор окончаний “перьев”

С помощью упомянутых выше свойств обводки можно настроить любой элемент пользовательского интерфейса, но некоторые из них, например свойства `StrokeStartLineCap` (Окончание линии в начале) и `StrokeEndLineCap` (Окончание линии в конце), оказываются полезными только для рисования линий. Нарисуйте сначала линию инструментом Pen или Line на монтажном столе, а затем измените толщину обводки, установив значение **10** в свойстве `StrokeThickness` этого графического элемента. Затем попробуйте изменить свойства `StrokeStartLineCap` и `StrokeEndLineCap`, чтобы воспользоваться одним из следующих вариантов выбора окончаний “перьев”: `Flat` (Плоское), `Square` (Квадратное), `Round` (Круглое) или `Triangle` (Треугольное). На рис. 2.27 для примера показан результат применения круглого окончания “пера” в начале линии и треугольного окончания “пера” в конце линии.

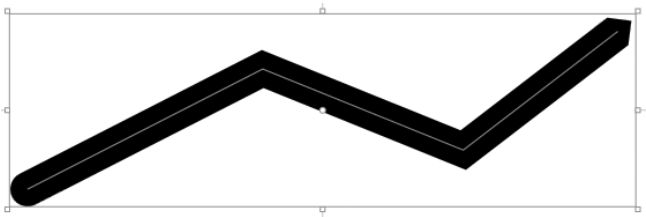


Рис. 2.27. С помощью окончаний “перьев” можно задать форму линии в ее начале и в конце

Примечание. Для того чтобы увидеть результат применения выбранного окончания “пера”, возможно, придется увеличить масштаб геометрической формы, что зависит от толщины линии. Как упоминалось в главе 1, масштаб отдельных участков монтажного стола можно изменять с помощью колесика мыши или соответствующих элементов управления в левом нижнем углу монтажного стола.

Выбор образца пунктира

А теперь найдите свойство `StrokeDashArray` (Массив обводок пунктира) в редакторе внешнего вида. По умолчанию в этом свойстве установлены значения **`1 0`**. С помощью свойства `StrokeDashArray` “перо” настраивается на рисование *пунктира* по любому *образцу*. Первое значение (**1**) этого свойства обозначает длину пунктира, а второе значение (**0**) — длину промежутка. Таким образом, значения **`1 0`**, по существу, определяют сплошную линию пунктира без промежутков. Если изменить эти значения на **`1 1`**, то тем самым будет определена пунктирная линия, приведенная на рис. 2.28.

Примечание. Числовые значения, задаваемые в свойстве `StrokeDashArray`, можно разделять запятой вместо пробела. Это означает, что можно ввести **`1, 1`** вместо `1 1` в поле данного свойства. Но как только вы введете эти значения через запятую и нажмете клавишу <Enter>, запятая будет автоматически исключена.

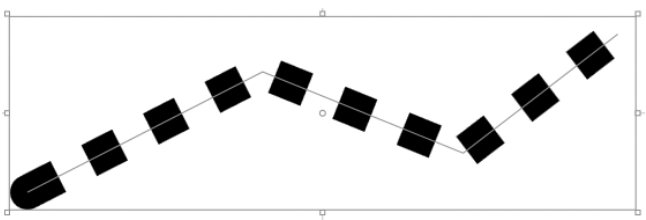


Рис. 2.28. Пунктирная линия

Значения, присваиваемые свойству `StrokeDashArray`, совсем не обязательно должны определяться лишь одной парой чисел. Вы можете создавать и более сложные сочетания пунктиров и промежутков между ними, определяя несколько пар числовых значений, каждая из которых должна обозначать часть пунктирной линии. В качестве примера ниже приведена строка кода, в которой образец новой пунктирной линии, рисуемой инструментом Line, присваивается свойству `StrokeDashArray`.

```
StrokeDashArray="0.5 1 2 1"
```

В данном случае линия начинается с рисования половинного пунктира, после чего следует пробел, затем двойной пунктир и снова пробел. Этот образец повторяется в остальной части рисуемой линии, в результате чего получается пунктирная линия, приведенная на рис. 2.29.

Рис. 2.29. Замысловатая пунктирная линия

Остальные свойства обводки, доступные в области `Appearance` на панели Properties, мы оставляем вам для самостоятельного изучения. Если же вам потребуются дополнительные сведения о конкретном свойстве, обращайтесь за справкой к руководству пользователя Expression Blend.

Еще раз о применении визуальных эффектов

В главе 1 у вас уже была возможность применить визуальный эффект в рассматривавшемся там примере проекта. Но нелишне будет напомнить еще раз, что в библиотеке ресурсов (или на панели Assets, где много проще выполнять операции перетаскивания) предоставляется целый ряд встроенных в Expression Blend визуальных эффектов, которые можно применять к любому элементу пользовательского интерфейса, доступному на панели Objects and Timeline. На рис. 2.30 приведена категория Effects библиотеки ресурсов Expression Blend, в которой доступны различные визуальные эффекты.

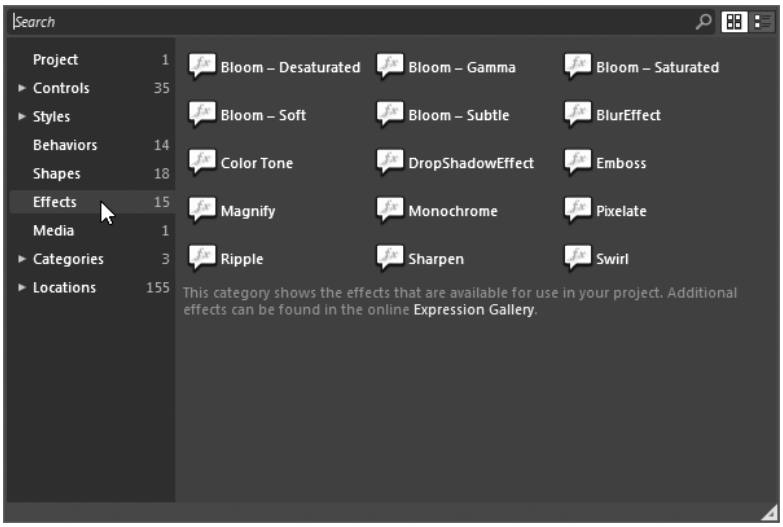


Рис. 2.30. Визуальные эффекты могут быть применены из библиотеки ресурсов к любому элементу пользовательского интерфейса

Для того чтобы применить визуальный эффект, перетащите его на выбранный узел в иерархическом представлении на панели Objects and Timeline или же прямо на целевой графический элемент, находящийся на монтажном столе. На рис. 2.31 показана панель Objects and Timeline после применения различных визуальных эффектов к выбранным геометрическим формам (для большей наглядности эти эффекты выделены).

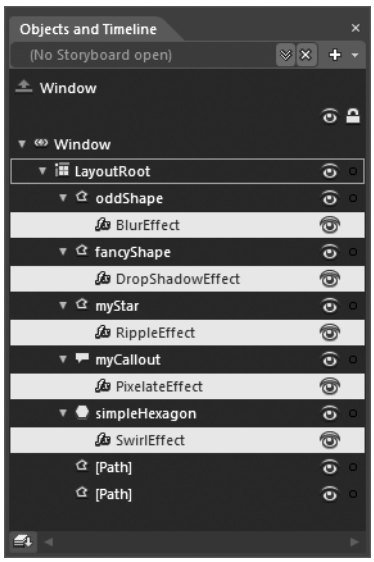


Рис. 2.31. Различные визуальные эффекты, применяемые к геометрическим формам

Примечание. Немало дополнительных визуальных эффектов (и других интересующих вас элементов графического оформления пользовательского интерфейса) можно свободно загрузить, посетив веб-сайт Microsoft Expression Gallery по адресу `http://gallery.expression.microsoft.com`.

Настройка визуального эффекта

Применив визуальный эффект к избранному элементу пользовательского интерфейса, можете затем выбрать его на панели Objects and Timeline и посмотреть любые настраиваемые свойства данного эффекта. У каждого визуального эффекта имеется свой определенный набор свойств. Поэкспериментировав немного с этими свойствами, вы сможете в дальнейшем создавать без особого труда весьма интересные и довольно привлекательные графические эффекты. В качестве примера на рис. 2.32 приведены свойства, которые могут быть заданы для эффекта падающей тени, определяемого объектом типа `DropShadowEffect`.

Однако подавляющее большинство визуальных эффектов, которые поддерживаются в Expression Blend, входят в состав сборки .NET под названием `Microsoft.Expression.Effects.dll`. Если вам придется разрабатывать новый проект WPF или Silverlight в среде Visual Studio 2010, имейте в виду, что у вас имеется возможность ссылаться вручную на эту библиотеку в диалоговом окне Add References, где она перечислена в алфавитном порядке на вкладке .NET. А если вам требуется ссылаться на упомянутые выше эффекты непосредственно в коде, то для этой цели придется импортировать пространство имен `Microsoft.Expression.Media.Effects`.

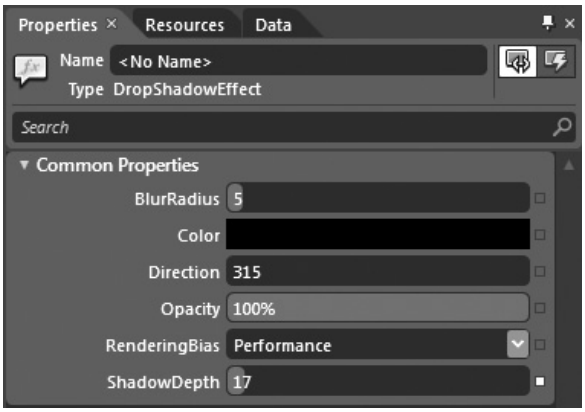


Рис. 2.32. Каждый визуальный эффект может быть настроен с помощью свойств, доступных на панели Properties

Примечание. Как упоминалось в главе 1, на монтажном столе Expression Blend имеется кнопка, с помощью которой визуальные эффекты можно отключать на время разработки. Для этого найдите кнопку **fx** среди элементов управления в левом нижнем углу монтажного стола.

На этом рассмотрение основных инструментов рисования в Expression Blend завершается. Далее речь пойдет о том, как импортируются графические данные, создаваемые средствами Expression Design, и для чего это может потребоваться.

Исходный код. Исходный код примера проекта `BlendDrawingTools` находится в папке `Ch 2 Code` загружаемого архива примеров проектов к данной книге.

Назначение инструментального средства Expression Design

Несмотря на всю пользу инструментов рисования в Expression Blend, художнику-оформителю было бы непросто создавать сложную векторную графику описанными выше способами и средствами. Правда, для создания более сложной графики можно воспользоваться инструментальным средством Expression Design, чтобы затем экспортировать полученные графические данные в файлах самых разных форматов, включая и XAML. Если графические данные экспортируются в формате XAML, их можно затем без особого труда импортировать в разрабатываемое приложение WPF или Silverlight и далее манипулировать полученными таким образом графическими объектами на панели Objects and Timeline, применяя все описанные ранее средства, в том числе визуальные эффекты, взаимодействие с формами в коде и пр.

Примечание. Подробное описание Expression Design выходит за рамки этой книги и, откровенно говоря, находится вне компетенции автора. Поэтому, если хотите поближе ознакомиться с применением Expression Design для создания собственной графики, обращайтесь к руководству пользователя Expression Design, доступному по команде меню Help этого инструментального средства.

Подготовка и экспорт данных из примера графического изображения

Для того чтобы приступить к работе над следующим проектом, запустите Expression Design на выполнение. Если у вас имеется некоторый опыт выполнения графических работ, нарисуйте какой-нибудь графический элемент специально для данного примера. А если у вас, как и у меня, нет особых художественных талантов, выберите команду меню Help⇨Samples (Справка⇨Примеры), чтобы найти среди целого ряда файлов с расширением `*.design` подходящий для загрузки пример готовой графики, например файл `bear _ paper.design` (рис. 2.33, а также цветная вклейка).

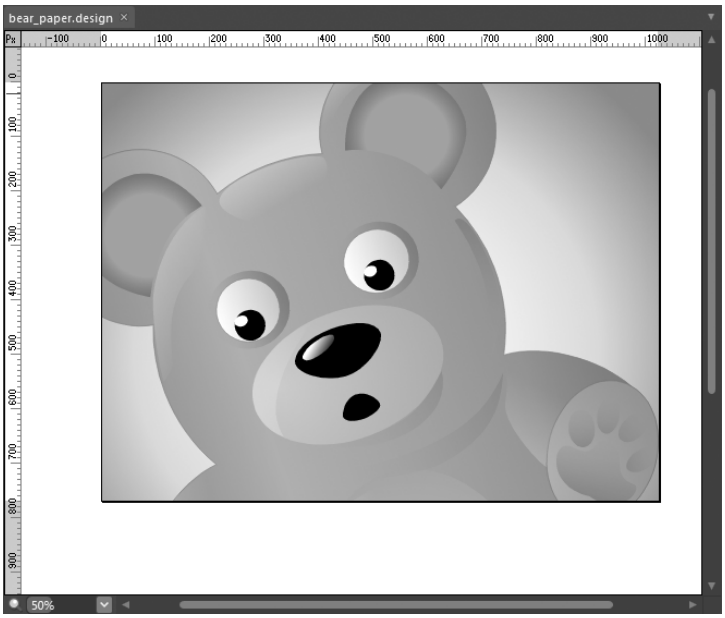


Рис. 2.33. Пример графического изображения плюшевого медвежонка

Примечание. Если у вас не установлена копия Expression Design, воспользуйтесь файлом `bear _ paper.xaml` из папки `Ch 2 Code` загружаемого архива примеров проектов к этой книге.

Прежде чем экспортировать данные упомянутого выше графического изображения в файле формата XAML, в них нужно внести некоторые коррективы. В настоящий момент все графическое изображение состоит из мордочки и лапки плюшевого медвежонка. Но в действительности изображение, которое отображается в *видовом окне*, составляет всего лишь малую долю более крупного графического изображения.

Для просмотра всего графического изображения нажмите комбинацию клавиш <Ctrl+A>, в результате чего будут выделены все графические данные файла документа с расширением `*.design`. Как показано на рис. 2.34, большая часть тела плюшевого медвежонка оказывается за пределами текущего видового окна, а курсор находится на вытягиваемом маркере в правом верхнем углу рамки, ограничивающей видовое окно.

Итак, поместите курсор мыши на вытягиваемом маркере в правом верхнем углу рамки, ограничивающей видовое окно, и потяните маркер внутрь графического изображения, чтобы уменьшить его по высоте и ширине приблизительно наполовину. После этого можете выбрать мышью данные графического изображения и вернуть их обрат-

но в пределы ограничивающей рамки. По завершении всех этих операций графическое изображение плюшевого медвежонка должно выглядеть так, как показано на рис. 2.35, а также на цветной вклейке.

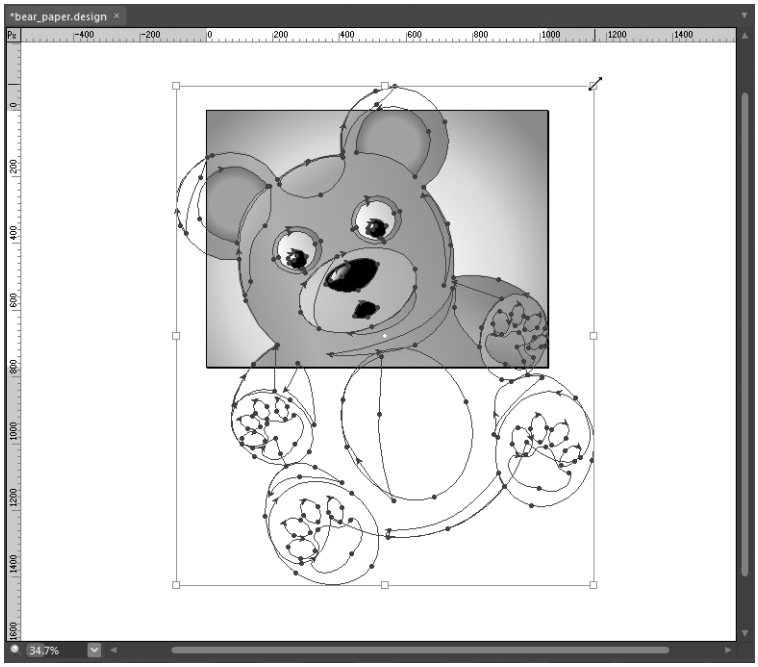


Рис. 2.34. В текущем видом окне наблюдается лишь малая часть всего графического изображения плюшевого медвежонка

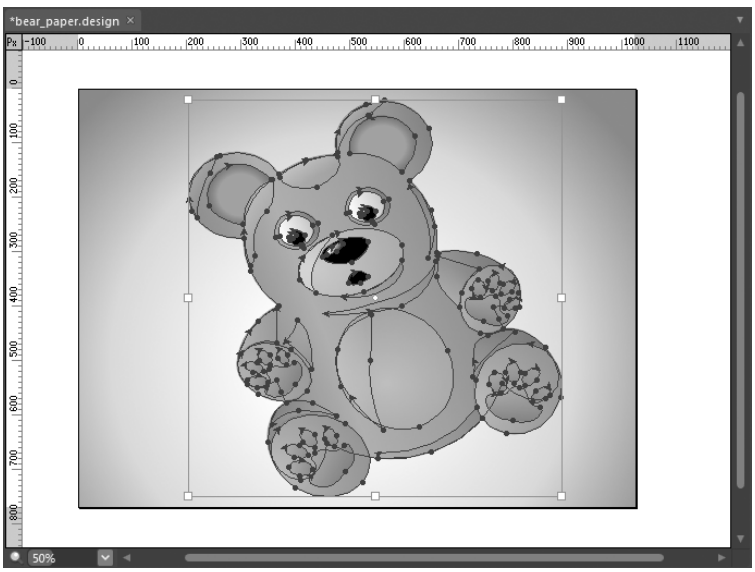


Рис. 2.35. Графическое изображение плюшевого медвежонка после изменения его размеров

А теперь, когда графические данные рассматриваемого здесь примера изображения подготовлены, можете экспортировать их по команде меню File⇨Export (Файл⇨Экспорт). Из раскрывающегося списка Format (Формат) открывшегося при этом диалогового окна можно выбрать любой из наиболее распространенных форматов файлов, но в данном примере нас интересует формат XAML Silverlight 4/WPF Canvas, поэтому выберите именно его (рис. 2.36).

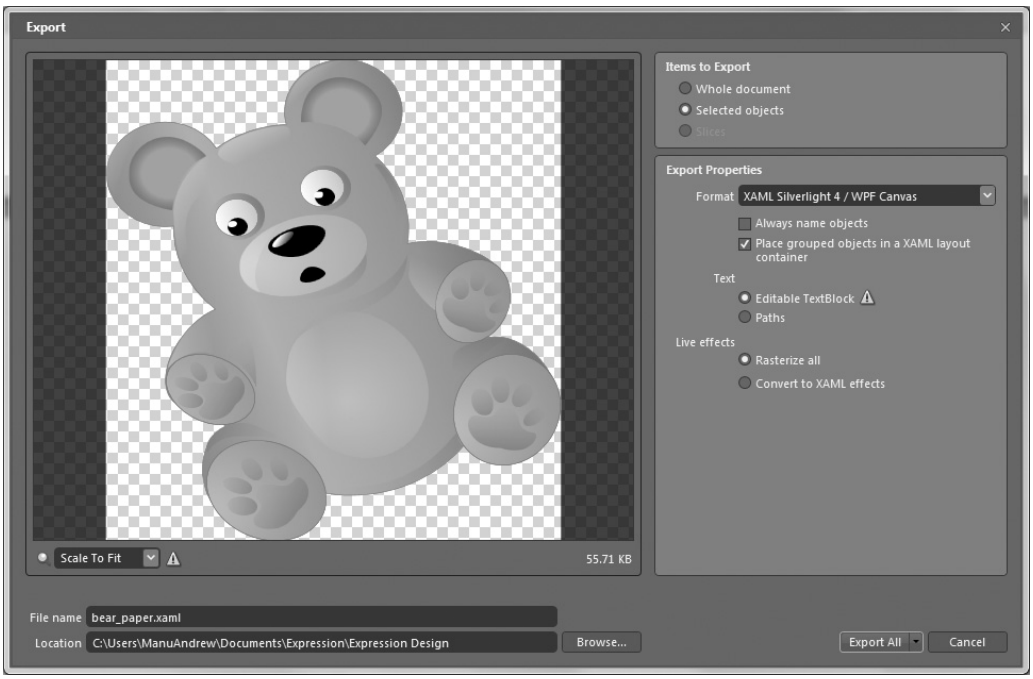


Рис. 2.36. Экспорт графического изображения плюшевого медвежонка в формате XAML

Выбрав подходящий формат XAML-файла, сбросьте флажок Always name objects (Всегда именовать объекты). Напомним, что если у графического элемента имеется атрибут `x:Name`, то с таким элементом можно взаимодействовать в коде. Но рассматриваемое здесь графическое изображение плюшевого медвежонка описывается с помощью довольно большого числа элементов XAML. Поэтому, если дать команду именовать все возможные объекты, оставив установленным упомянутый выше флажок, в базовый код C# (или VB) в конечном итоге будет добавлено довольно большое количество переменных. Вряд ли всеми этими переменными удастся воспользоваться в коде, а вот размер исполняемого файла окончательно скомпилированного приложения может заметно увеличиться.

Далее выберите подходящее место для сохранения данных, экспортируемых в файле `bear _ paper.xaml`, чтобы последний можно было легко найти в дальнейшем, например на рабочем столе Windows. Все остальные параметры настройки можно оставить установленными по умолчанию.

Когда все будет готово для экспорта графических данных, щелкните на кнопке Export All (Экспортировать все). В итоге данные графического изображения плюшевого медвежонка будут экспортированы в формате XAML. Теперь можете закрыть Expression Design.

Создание нового проекта приложения Silverlight

Итак, все готово для импорта упомянутых выше графических данных в приложение WPF или Silverlight. Но прежде нужно создать новый проект приложения Silverlight в Expression Blend и присвоить ему имя `InteractiveTeddyBear`, как показано на рис. 2.37.

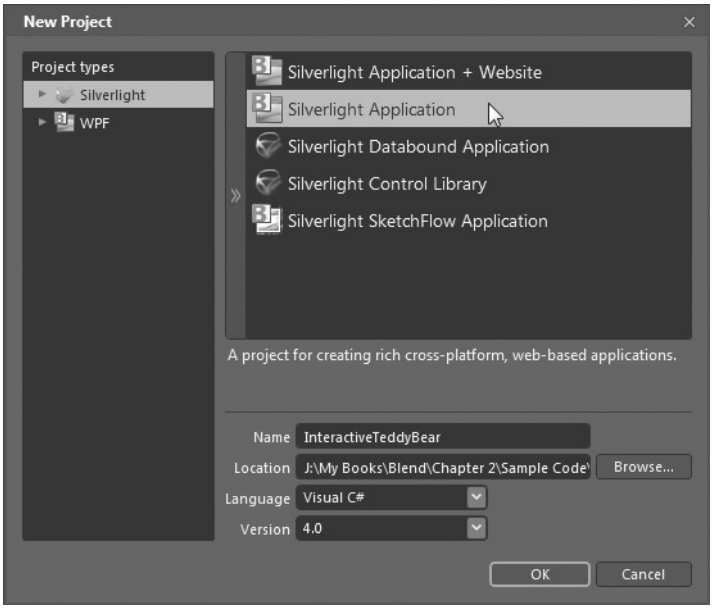


Рис. 2.37. Создание нового проекта приложения Silverlight в Expression Blend

Выберите сначала объект типа `UserControl` для приложения Silverlight на панели Objects and Timeline, а затем увеличьте размеры этого объекта в достаточной степени. Точные размеры особого значения не имеют, но в качестве ориентира можете установить значение **800** в полях свойств `Height` (Высота) и `Width` (Ширина), доступных в области Layout панели Properties.

Импорт данных из примера графического изображения в Expression Blend

А теперь наступает самое интересное! Выберите команду меню Project⇨Add Existing Item (Проект⇨Добавить существующий элемент) и перейдите в появившемся диалоговом окне к файлу `bear _ paper.xaml` (или к его копии из папки `Ch 2 Code` загружаемого архива примеров проектов к этой книге). Как только вы щелкнете на кнопке OK, этот XAML-файл будет добавлен в текущий проект (убедитесь в этом сами, исследовав содержимое панели Projects; см. главу 1).

Дважды щелкните на файле `bear _ paper.xaml`, появившемся на панели Projects, чтобы просмотреть импортированные данные графического изображения в среде Expression Blend IDE. Щелкните на кнопке Split, чтобы выбрать режим просмотра этих данных с разделением, а следовательно, иметь возможность видеть их исходную разметку в коде XAML (рис. 2.38).

Просматривая графическое изображение плюшевого медвежонка из файла `bear _ paper.xaml` в среде Expression Blend IDE, обратите внимание на каждый имеющийся и учитываемый в разметке элемент XAML. Ваша задача — найти *зрачок левого глаза* и *правое внутреннее ухо* плюшевого медвежонка и дать этим графическим элементам

подходящие имена. Искать эти графические объекты вручную не очень удобно и довольно утомительно, поэтому активизируйте инструмент Selection и выделите их. Нужный узел автоматически выделится на панели Objects and Timeline. Выделите сначала зрачок левого глаза и присвойте этому объекту имя `leftEye` (левый глаз) на панели Properties. Затем выделите часть правого уха по своему усмотрению и присвойте этому объекту имя `rightEar` (правое ухо). Для примера на рис. 2.39 показано, каким образом выделяется зрачок левого глаза.

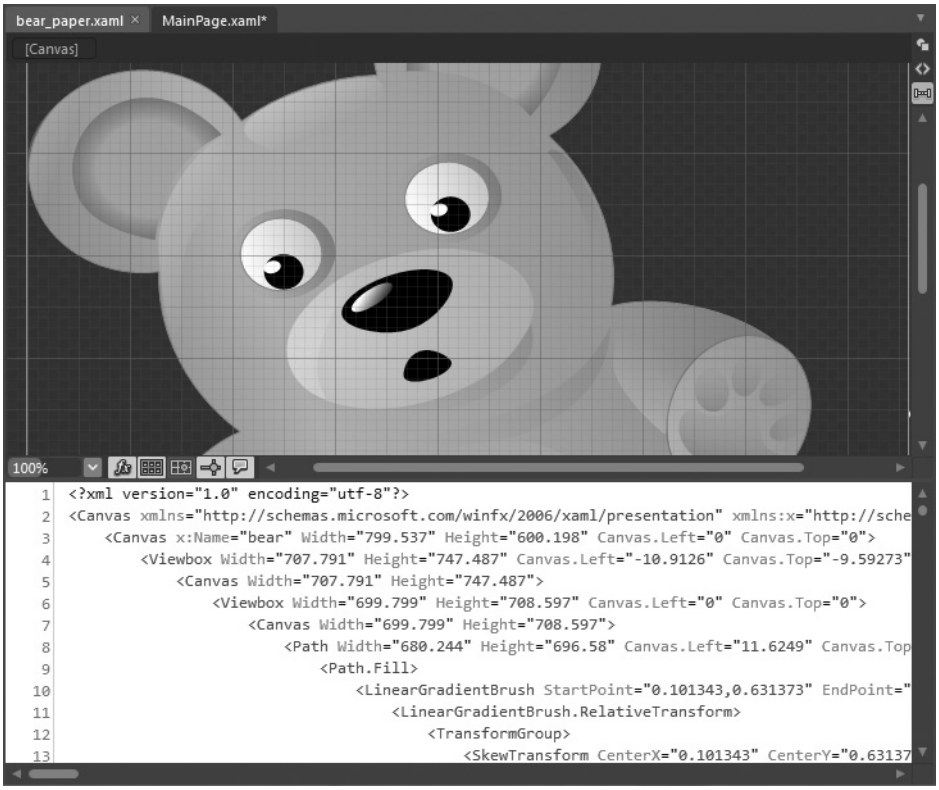


Рис. 2.38. Вид графического изображения плюшевого медвежонка и его разметки XAML в среде Expression Blend IDE

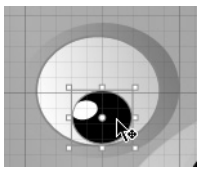


Рис. 2.39. Обнаружение нужных графических объектов с помощью инструмента Selection

Далее щелкните правой кнопкой мыши на узле `bear`, доступном на панели Objects and Timeline, и выберите команду Copy (Копировать) из всплывающего контекстного меню, как показано на рис. 2.40.

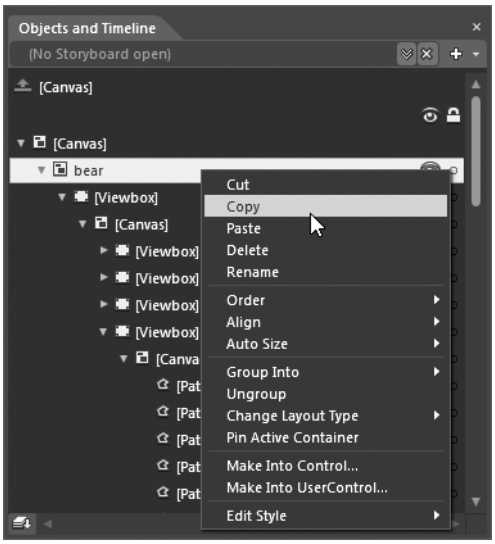


Рис. 2.40. Копирование узла `bear` в буфер обмена

Закройте файл `bear _ paper.xaml` и вернитесь к монтажному столу для работы с файлом `MainPage.xaml`. Щелкните правой кнопкой мыши на объекте `LayoutRoot`, доступном на панели Objects and Timeline, и вставьте содержимое буфера обмена. Как видите, объект медвежонка теперь находится на сетке как объект `bear` типа `Canvas`, вложенный в объект типа `Grid`!

Примечание. Возможно, изменять размеры или положение объекта `bear` типа `Canvas` в пределах объекта типа `Grid` вам будет труднее, чем вы предполагали, потому что в исходном примере графического изображения, импортированного из Expression Design, используется целый ряд накладывающихся один на другой слоев.

Взаимодействие с графическим изображением плюшевого медвежонка

Теперь можете обработать события, связанные с объектами `leftEye` и `rightEar`, аналогично обработке событий, связанных с простейшими формами, в предыдущем примере. С этой целью выделите объекты `leftEye` и `rightEar` по очереди на монтажном столе, перейдите к области Events на панели Properties и введите имена обработчиков соответствующих событий. В данном примере обрабатывается одно и то же событие `MouseLeftButtonDown`, наступающее после нажатия левой кнопки мыши, когда курсор находится на каждом из этих объектов, но методы его обработки должны иметь разные имена.

Ниже приведен простой код C#, изменяющий внешний вид каждого из упомянутых выше объектов, если щелкнуть на них левой кнопкой мыши. (Если у вас нет желания набирать весь этот код, ограничьтесь вводом оператора `MessageBox.Show()` в каждом из обработчиков событий для вывода на экран подходящего, на ваш взгляд, сообщения.)

```
private void leftEye_MouseLeftButtonDown(object sender,
  System.Windows.Input.MouseButtonEventArgs e)
{
  // Изменить цвет глаза, если щелкнуть на нем кнопкой мыши.
     leftEye.Fill = new SolidColorBrush(Colors.Red);
}
```

```
private void rightEar_MouseLeftButtonDown(object sender,
  System.Windows.Input.MouseButtonEventArgs e)
{
  // Сделать ухо размытым, если щелкнуть на нем кнопкой мыши.
  System.Windows.Media.Effects.BlurEffect blur =
    new System.Windows.Media.Effects.BlurEffect();
  blur.Radius = 80;
  rightEar.Effect = blur;
}
```

А теперь запустите приложение Silverlight на выполнение, нажав функциональную клавишу <F5>. Это приложение автоматически загрузится в окно избранного браузера. Щелкните на зрачке левого глаза плюшевого медвежонка. Его цвет должен измениться, как показано на рис. 2.41, а также на цветной вклейке.

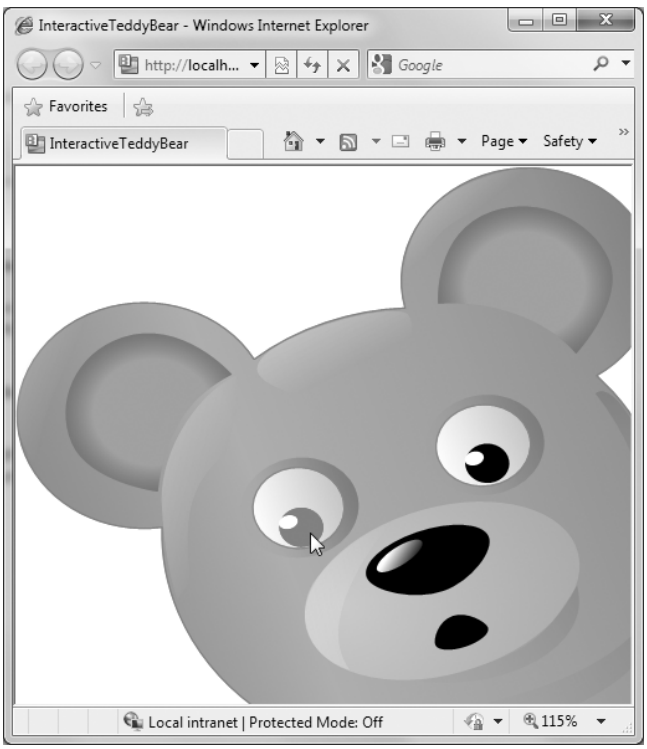


Рис. 2.41. Вид медвежонка после изменения цвета зрачка его левого глаза

Вот, собственно, и все! Теперь вам должен быть понятен процесс импорта сложных графических данных, сформированных средствами Expression Design, в проект Expression Blend, а самое главное — взаимодействие с графическими данными в коде.

По желанию можете вернуться к этому примеру, когда речь пойдет о применении редакторов анимации в Expression Blend, чтобы сделать его еще более интересным. В частности, можете организовать вращение зрачка глаза медвежонка по кругу и сплющивание его уха.

Исходный код. Все файлы проекта `InteractiveTeddyBear` находятся в папке `Ch 2 Code` загружаемого архива примеров проектов к данной книге.

Выполнение двухмерных графических преобразований

В приложениях Silverlight и WPF поддерживается возможность преобразования элементов пользовательского интерфейса самыми разными способами. В прикладных интерфейсах Silverlight API и WPF API предоставляется целый ряд функций для выполнения типичных графических преобразований, в том числе вращения, наклона, переворота и масштабирования. Следует иметь в виду, что целевым для графического преобразования может стать *любой* объект. Например, можете определить сложную компоновку элементов управления, внешний вид которых будет изменяться коренным образом в зависимости от вида входной информации от пользователя. Так, если пользователь щелкнет кнопкой мыши на элементе управления, последний может повернуться на определенный угол.

По мере приобретения опыта работы в среде Expression Blend IDE вы непременно обнаружите, что графические преобразования можно подвергать анимации и внедрять в специальные шаблоны для создания ряда весьма изящных визуальных эффектов. А до тех пор рассмотрим основные инструменты преобразования на примере нового проекта в Expression Blend.

Построение первоначального варианта пользовательского интерфейса

Итак, создайте новый проект приложения WPF, присвоив ему имя `Transformations`. Для данного примера может вполне подойти и приложение Silverlight, но создавать рекомендуется все же приложение WPF, поскольку далее будет рассмотрен ряд функциональных возможностей, отсутствующих в прикладном интерфейсе Silverlight API. А для того чтобы сделать данный пример более интересным, необходимо прежде всего определить *систему вложенной компоновки* для имеющегося объекта типа `Window`. О построении пользовательских интерфейсов с помощью диспетчеров компоновки и элементов управления более подробно речь пойдет в главе 4, а до тех пор придется ограничиться тем, что вы уже знаете о них.

Прежде всего разделите первоначальную сетку на два ряда, выбрав узел `LayoutRoot` на панели Objects and Timeline и щелкнув на пиктограмме внешнего редактора с изображением голубой сетки, чтобы добавить новый ряд. Конкретные размеры рядов сетки особого значения не имеют, но первый ряд должен быть значительно меньше второго. Убедившись в том, что объект типа `Grid` по-прежнему выбран на панели Objects and Timeline, найдите элемент управления типа `StackPanel` на панели Tools (рис. 2.42).

Найдя элемент управления типа `StackPanel`, дважды щелкните на его пиктограмме, находящейся на панели Tools, чтобы добавить этот элемент к объекту типа `Grid`, а затем растяните его, чтобы заполнить им все свободное пространство в первом ряду сетки. Выберите новый объект типа `StackPanel` на панели Objects and Timeline и найдите элемент управления типа `Button` (Кнопка) на панели Tools (рис. 2.43).

Дважды щелкните на элементе управления типа `Button` три раза подряд. В итоге три объекта типа `Button` должны быть вложены в объект типа `StackPanel`. Выберите все три элемента управления типа `Button` в качестве объектов на панели Objects and Timeline, нажав клавишу <Shift> и щелкнув сначала на первом элементе, а затем на последнем, после чего воспользуйтесь панелью Properties, чтобы установить значение **100** в поле свойства `Width`, доступного в области Layout, а также значение `10` в поле свойства `Margin`, определяющего нижнее поле компоновки пользовательского интерфейса и также доступного в области Layout (рис. 2.44).

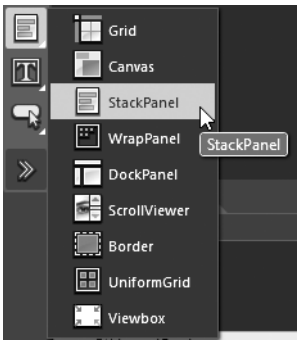


Рис. 2.42. Элемент управления типа `StackPanel`

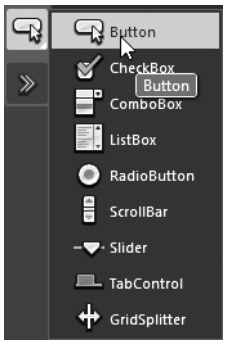


Рис. 2.43. Элемент управления типа `Button`

Рис. 2.44. Настройка элементов управления типа `Button`

А теперь выберите каждый из элементов управления `Button` по очереди инструментом Selection и замените значения их свойства `Content` (Содержимое), доступного в области Common Properties панели Properties, на `Skew` (Наклонить), `Rotate` (Повернуть) и `Flip` (Перевернуть) соответственно. Кроме того, воспользуйтесь полем Name на панели Properties, чтобы присвоить каждому объекту кнопки подходящее имя, например `btnSkew`, `btnRotate` и `btnFlip`. Щелкните сначала на кнопке Events, находящейся на панели Properties и имеющей знак молнии на пиктограмме, а затем организуйте обработку события `Click`, наступающего после щелчка на кнопке, для каждого элемента управления типа `Button`.

Примечание. Не забывайте о том, что можно дважды щелкнуть на отдельном событии, чтобы сформировать обработчик событий `Имя_Элемента_Имя_События`, используемый по умолчанию. Так, если обрабатывается событие `Click`, связанное с элементом управления `btnFlip`, соответствующий обработчик этого события будет называться `btnFlip_Click`.

И в завершение компоновки пользовательского интерфейса создайте любую геометрическую форму по вашему выбору любыми инструментами, представленными в этой главе и доступными на панели Tools, а затем введите эту форму во второй ряд сетки. (Напомним, что в области Layout панели Properties содержатся свойства, с помощью которых графический элемент назначается для отдельной ячейки сетки. Но в то же время можно перетащить мышью графический элемент на ячейку сетки.) Присвойте новому элементу пользовательского интерфейса имя `myShape`. На рис. 2.45 приведен вариант окончательной компоновки пользовательского интерфейса для рассматриваемого здесь примера.

Примечание. Ради простоты в качестве цели для преобразований в данном случае выбрана элементарная графика. Но не следует забывать, что любой элемент пользовательского интерфейса, включая диспетчеры компоновки, содержащие элементы управления, может быть преобразован представленными здесь и далее способами.

Выполнение преобразований на стадии разработки

Прежде чем вводить код в обработчики событий, покажем, каким образом графические преобразования выполняются на стадии разработки в редакторе преобразований, интегрированном в среду Expression Blend. С этой целью выделите специальную

форму инструментом Selection на монтажном столе и перейдите к области Transform (Преобразование) на панели Properties (рис. 2.46).

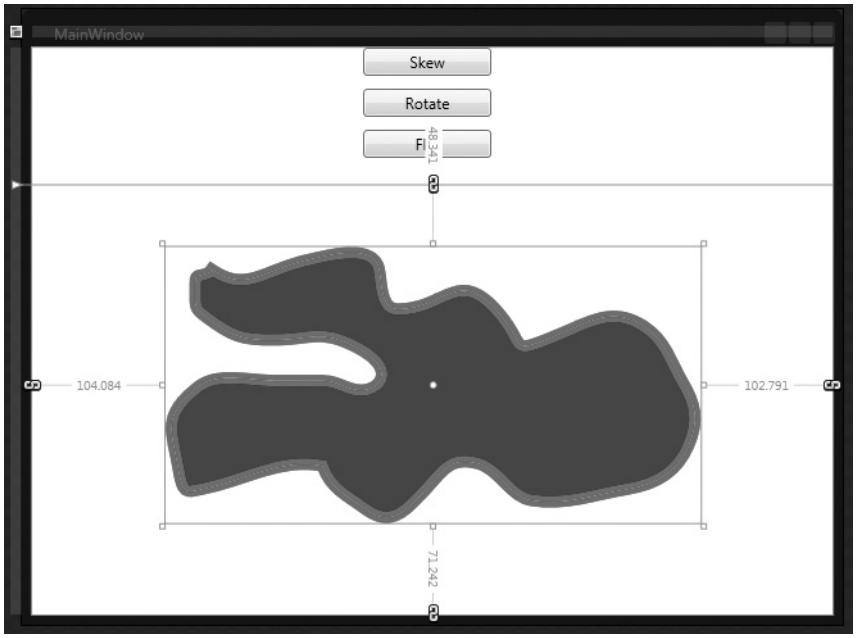


Рис. 2.45. Компоновка пользовательского интерфейса приложения WPF

Рис. 2.46. Редактор преобразований

Как и в области Brushes, в области Transform имеется целый ряд вкладок, предназначенных для настройки различных видов графических преобразований элемента, выбранного на панели Objects and Timeline. В табл. 2.1 кратко описано назначение каждого вида преобразования, перечисляемого по порядку следования вкладок слева направо в области Transform на панели Properties.

Таблица 2.1. Виды графических преобразований в Expression Blend

Вид преобразования	Назначение
Translate	Перемещает графический элемент, находящийся в точке с координатами X, Y
Rotate	Поворачивает графический элемент на угол до 360°
Scale	Увеличивает или уменьшает графический элемент, находящийся в точке с координатами X, Y
Skew	Наклоняет рамку, ограничивающую выбранный графический элемент, находящийся в точке с координатами X, Y

Окончание табл. 2.1

Вид преобразования	Назначение
Center Point	При вращении или переворачивании графического элемента он смещается относительно фиксированной точки, называемой *центральной*. По умолчанию центральная точка графического элемента находится в центре данного объекта. С помощью этого преобразования можно изменить центральную точку графического элемента, чтобы повернуть или перевернуть его вокруг другой точки
Flip	Переворачивает выбранный графический элемент по оси X или Y

Рекомендуется опробовать каждое из упомянутых выше графических преобразований, используя любую выбранную геометрическую форму в качестве цели и нажимая комбинацию клавиш <Ctrl+Z> для отмены предыдущей операции преобразования. Аналогично многим другим свойствам, доступным на панели Properties, для каждого вида преобразования имеется особый ряд свойств, с которыми вы должны познакомиться поближе, чтобы научиться их настраивать. Например, в редакторе преобразования наклоном (Skew) можете задавать величины наклона графического объекта по осям X и Y, используя ползунковые элементы управления, а в редакторе преобразования переворачиванием (Flip) — выбирать переворачивание по оси X или Y.

Преобразования во время визуализации или компоновки

Перейдя к области Transform панели Properties, вы обнаружите там область дополнительных свойств. Если развернуть эту область, то в ней появится вспомогательный редактор преобразований, более или менее похожий на исходный. Отличие между ними заключается в том, что преобразования в исходном (расположенном сверху) редакторе считаются выполняющимися во время *визуализации*, тогда как преобразования во вспомогательном (расположенном снизу) редакторе считаются выполняющимися во время *компоновки* (рис. 2.47).

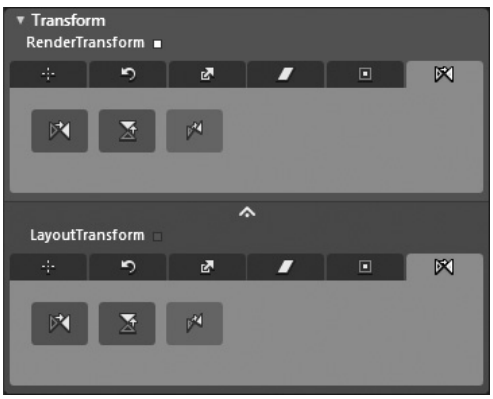


Рис. 2.47. Два способа выполнения графических преобразований, задаваемых в редакторе

Примечание. Графические элементы в приложениях Silverlight могут быть преобразованы только в режиме визуализации. А преобразования в режиме компоновки действительны только для приложений WPF.

Несмотря на то что в обоих упомянутых выше редакторах преобразований в конечном итоге изменяется порядок отображения выбранного графического элемента на экране, отличия между ними заключаются во времени выполнения самого преобразования. Выполнять преобразования лучше всего во время визуализации. В этом случае преобразования выполняются после визуализации графического элемента в окне (объект типа `Window`) или в элементе управления пользовательского интерфейса (объект типа `UserControl`), обеспечивая лучшую производительность.

С другой стороны, преобразования во время компоновки выполняются в оперативной памяти на стадии компоновки и перед выводом элементов пользовательского интерфейса на экран. И хотя это может в какой-то степени сказаться на производительности, преимущество преобразований во время компоновки заключается в том, что родительский объект (как правило, диспетчер компоновки) может динамически изменять преобразования порожденных объектов, оказывая влияние на порядок их следования.

Выполнение преобразований в коде

Реализация каждого обработчика событий типа `Click` оказывается более или менее одинаковой. Сначала настраивается объект преобразования, а затем ему присваивается объект `myShape`. Следовательно, при выполнении своего приложения можете щелкнуть на кнопке, чтобы увидеть результат выполнения соответствующего преобразования. Ниже приведен полный код каждого обработчика событий. Обратите внимание на то, что свойство `LayoutTransform` установлено таким образом, чтобы данные специальной формы оставались расположенными относительно своего родительского контейнера.

```
private void btnFlip_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
   myShape.LayoutTransform = new ScaleTransform(-1, 1);
}

private void btnRotate_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
  myShape.LayoutTransform = new RotateTransform(180);
}

private void btnSkew_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
  myShape.LayoutTransform = new SkewTransform(40, -20);
}
```

Запустив приложение на выполнение (нажатием функциональной клавиши <F5>), вы сможете динамически изменять расположение специальной формы щелчком на кнопке. В следующем разделе будет рассмотрен более сложный вид преобразования в трехмерном пространстве.

Исходный код. Исходный код примера проекта `Transformations` находится в папке `Ch 2 Code` загружаемого архива примеров проектов к данной книге.

Выполнение трехмерных графических преобразований

В обоих примерах проектов, созданных вами ранее в этой главе, применялась двухмерная графика. В Expression Blend имеются также специальные инструменты для создания, манипулирования и импортирования трехмерной графики. Но обращение с трехмерной графикой в Expression Blend имеет свои отличия и зависит от типа разрабатываемого приложения: WPF или Silverlight. Напомним, что технология WPF ограничивается только пределами Windows, и поэтому ее прикладной интерфейс API может опираться на конкретные графические службы этой операционной системы, в том числе на Microsoft DirectX. Благодаря этому платформа WPF способна поддерживать полноценную трехмерную среду. С другой стороны, технология Silverlight является межплатформенной, а ее прикладной интерфейс API опирается на функциональные возможности браузеров. Поэтому в прикладном интерфейсе Silverlight API не поддерживается полноценная трехмерная среда, а вместо этого применяется ее "облегченный" вариант, называемый *трехмерной графикой в перспективе*.

В силу разной поддержки трехмерной графики на уровне упомянутых выше прикладных интерфейсов API в Expression Blend предоставляются отдельные инструменты для работы с трехмерной графикой для каждой из платформ WPF и Silverlight. Поэтому в представленном ниже примере будут рассмотрены основы работы с трехмерной графикой в контексте проекта WPF. А о поддержке трехмерной графики в Silverlight речь пойдет далее в главе.

Примечание. Конечно, вы можете пропустить этот раздел, если вас интересует только разработка приложений на платформе Silverlight. Но все же откройте пример рассматриваемого здесь проекта WPF, чтобы сравнить возможности поддержки трехмерной графики на платформах WPF и Silverlight.

Введение в трехмерную графику на платформе WPF

В Expression Blend отсутствует возможность формирования полноценной трехмерной модели с помощью встроенных в эту среду инструментов. Но в ней все же имеются средства для импорта модели трехмерного объекта, сформированной специально предназначенными для этой цели сторонними инструментальными средствами. Прежде чем рассматривать одно из таких средств (ZAM 3D), обратимся к примеру, наглядно показывающему, каким образом двухмерное изображение накладывается на трехмерную плоскость, чтобы затем манипулировать им на монтажном столе посредством кода.

Примечание. Подробное рассмотрение особенностей трехмерного моделирования выходит за рамки этой книги. Если вам приходилось прежде работать с трехмерной графикой, то при чтении последующего материала у вас должно сложиться ясное представление о возможностях Expression Blend и WPF в отношении трехмерной графики. Если же вам требуется более основательное изучение данной темы, перейдите по адресу `www.msdn.com` и выполните поиск по критерию *WPF 3D*. В итоге вы получите целый ряд ссылок на информационные ресурсы для дальнейшего изучения данной темы.

Наложение двухмерного изображения на трехмерную плоскость

Прежде всего создайте новый проект приложения WPF и назовите его `Wpf3DExample`. Затем найдите на своем компьютере файл изображения, который хотели бы перенести в трехмерную систему координат. Найдя подходящее изображение, откройте панель

Projects, щелкните правой кнопкой мыши на имени своего проекта и выберите команду Add Existing Item из всплывающего контекстного меню. Найдите файл выбранного вами изображения в открывшемся диалоговом окне. Копия файла этого изображения будет помещена в текущий проект, как показано на рис. 2.48. (Если вы наведете курсор мыши на файл этого изображения, появится его миниатюрный вид[4].)

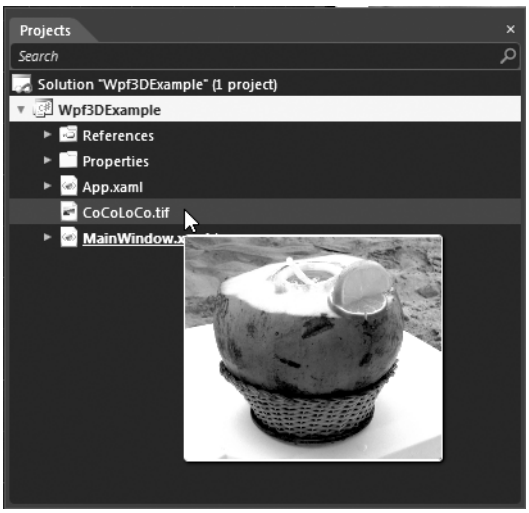


Рис. 2.48. Добавление изображения в текущий проект

Примечание. Если вы введете в свой проект файл слишком крупного изображения, вам будет предложено встроить данные изображения в скомпилированный исполняемый файл приложения. Но для целей рассматриваемого здесь примера это не имеет никакого значения.

А теперь перетащите мышью файл изображения из его узла в иерархическом представлении на панели Projects непосредственно на монтажный стол[5]. Измените размеры изображения таким образом, чтобы оно заняло верхнюю половину окна приложения WPF. Прежде чем двигаться дальше, просмотрите разметку, сформированную в коде XAML, обратив внимание на автоматически добавленный элемент управления типа `Image` (Изображение), в свойстве `Source` (Источник) которого установлено значение, указывающее на файл выбранного вами изображения, а иначе — на местоположение встроенных данных изображения, как показано ниже.

```
<Image Height="481" Margin="79.215,0,-284.215,-464"
      Source="pack://siteoforigin:,,,/CoCoLoCo.tif"
      Stretch="Fill" VerticalAlignment="Bottom"/>
```

Примечание. Необычное значение, присвоенное свойству `Source`, имеет особый синтаксис, используемый для обнаружения источника изображения. Если же данные изображения не были встроены в исполняемый файл приложения, то в разметке XAML будет указано более простое и понятное значение: `Source="CoCoLoCo.tif"`.

[4] Здесь показано изображение коктейля “Шальной кокос”. По завершении рассматриваемого здесь проекта, в котором вам придется вращать изображение в трехмерной плоскости, вы в полной мере испытаете такие же ощущения, как и после выпитого вышеназванного коктейля.

[5] В элементе управления `Image` на платформе Silverlight поддерживаются файлы изображений только формата JPEG или PNG.

Выберите сначала элемент управления `Image` на панели Objects and Timeline, а затем команду Tools⇨Make 3D Image (Сервис⇨Сформировать трехмерное изображение) из главного меню Expression Blend IDE. На монтажном столе вы не заметите никаких особых изменений, но если просмотрите разметку, сформированную в коде XAML, то обнаружите, что простой некогда элемент управления типа `Image` полностью преобразован в более сложный объект типа `Viewport3D` (Трехмерное видовое окно), часть которого составляет новый объект типа `ImageBrush`, построенный из файла исходного изображения. Ниже вкратце поясняются основные элементы объекта типа `Viewport3D`.

Элементы объекта типа `Viewport3D`

Объем кода XAML, представляющего объект `Viewport3D`, слишком велик, чтобы приводить его на этих страницах. Но если вы проанализируете содержимое панели Objects and Timeline, то заметите, что объект `Viewport3D` состоит из четырех подчиненных элементов: `Camera`, `ModelContainer`, `AmbientContainer` и `DirectionalContainer`. Разверните узел `Camera`, выберите подчиненный узел `[PerspectiveCamera]` и воспользуйтесь панелью Properties, чтобы присвоить этому объекту имя `my3DCamera` (моя трехмерная камера). На рис. 2.49 приведен результат переименования этого объекта со всеми выделенными подчиненными узлами.

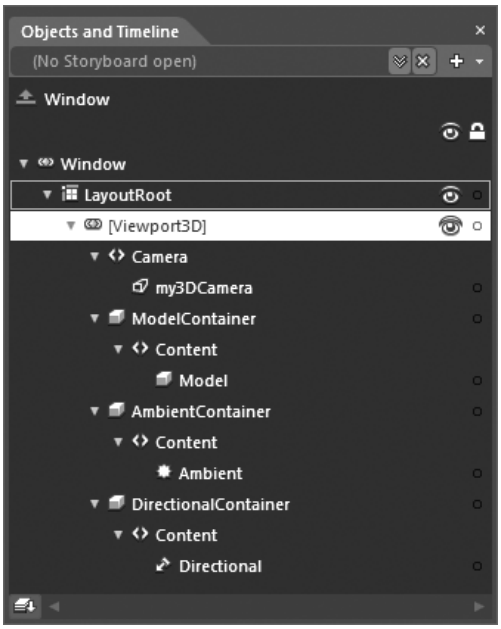


Рис. 2.49. Иерархическая структура элемента `Viewport3D`, с помощью которого осуществляется трехмерное проецирование изображений

Каждая составляющая объекта типа `Viewport3D` позволяет управлять различными свойствами визуализации данных изображения в трехмерном пространстве, в том числе местоположением, освещением и пр. Эти свойства будут представлены по отдельности далее в главе. В табл. 2.2 приводится краткое описание назначения отдельных составляющих объекта типа `Viewport3D`.

Таблица 2.2. Подчиненные элементы объекта типа `Viewport3D`

Подчиненный элемент	Назначение
`Camera`	Позволяет управлять расположением места, наблюдаемого в камеру, а также точкой расположения просматриваемого изображения
`ModelContainer`	Показывает исходную трехмерную модель, используемую для отображения трехмерного видового окна
`AmbientContainer`	Позволяет добавлять, удалять и манипулировать "материалами", определяющими порядок визуализации световых эффектов впереди и позади изображения в трехмерном видовом окне
`DirectionalContainer`	Позволяет добавлять, удалять и манипулировать "материалами", определяющими направление света, проецируемого из внешнего источника на трехмерное изображение

Преобразование трехмерного видового окна с помощью инструмента Camera Orbit

Выбрав объект типа `Viewport3D` на панели Objects and Timeline, активизируйте инструмент Camera Orbit (Вращение камеры по кругу) на панели Tools, как показано на рис. 2.50.

Рис. 2.50. Инструментом Camera Orbit можно преобразовать трехмерное видовое окно, манипулируя мышью

Щелкнув на инструменте Camera Orbit, поместите курсор мыши на выбранном объекте типа `Viewport3D` и попробуйте изменить положение осей X, Y, Z рассматриваемого здесь графического элемента. С этой целью щелкните левой кнопкой мыши, перемещая ее курсор, как показано на рис. 2.51.

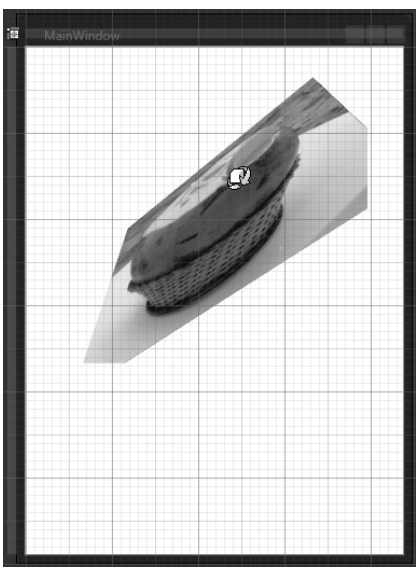


Рис. 2.51. Изменение ориентации видового окна в трехмерном пространстве инструментом Camera Orbit

Настройка свойств камеры на панели Properties

Инструмент Camera Orbit полезен в том случае, если требуется быстро преобразовать данные в трехмерном пространстве, используя приблизительные размеры. Когда же требуется большая точность, следует обратиться к свойствам камеры, доступным в области Camera на панели Properties. С этой целью выберите сначала узел `Camera`, подчиненный узлу `Viewport3D`, на панели Objects and Timeline (напомним, что этот объект был переименован в `my3DCamera`). Как только вы выберете этот графический элемент, он появится в области Camera на панели Properties, как показано на рис. 2.52.

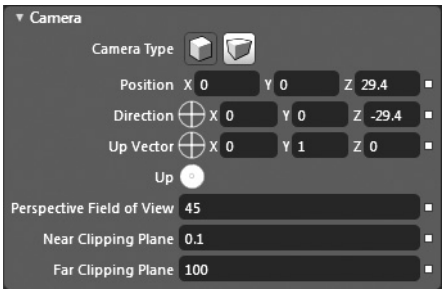


Рис. 2.52. Вид области Camera на панели Properties

Настраивая свойства в редакторе камеры из области Camera на панели Properties, можете изменить те же самые общие настройки, что и при манипулировании инструментом Camera Orbit, в том числе местоположение видового окна, расположение камеры и направление взгляда. В этой области можно также выбрать один из двух доступных типов камеры: *вид в перспективе* и *вид в ортогональной проекции*.

По умолчанию для нового объекта типа `Viewport3D` выбирается камера вида в перспективе. Она действует подобно обычной фотокамере, которой делают снимки: по мере отдаления объекта съемки от камеры он становится все меньше и меньше. Если же выбрать камеру вида в ортогональной проекции, графические данные в видовом окне *не* будут визуализироваться с уменьшением масштаба по мере отдаления объекта съемки от камеры. Такой вид из камеры может оказаться полезным в том случае, если требуется преобразовать изображение в трехмерном пространстве, сохранив общие пропорции (высоту и ширину) исходной области просмотра.

Примечание. Для того чтобы получить наглядное представление об отличиях в обоих типах камер, попробуйте переключаться с одной камеры на другую, когда пользуетесь инструментом Camera Orbit в процессе разработки. При этом вы сразу же ощутите отличия, которые невозможно передать на печатной странице!

В редакторе камеры имеется ряд других свойств, которые можете исследовать на досуге, управляя полем зрения камеры, местоположением плоскостей усечения и т.д. Все эти свойства здесь не рассматриваются. Имейте, однако, в виду, что остальные свойства будут изменяться в зависимости от типа выбранной камеры.

Изменение ориентации трехмерного видового окна инструментами на монтажном столе

Помимо инструмента Camera Orbit и редактора камеры, непосредственно на монтажном столе можно также выбрать инструмент, изменяющий положение графических объектов в трехмерном пространстве. Для того чтобы активизировать этот инструмент, найдите и выберите узел `Model` ниже узла `ModelContainer`, как показано на рис. 2.53.

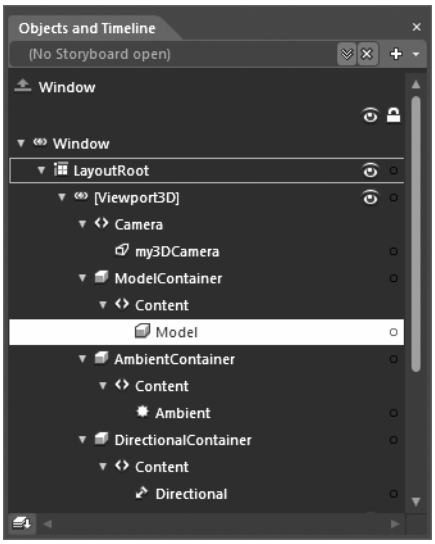


Рис. 2.53. В узле `ModelContainer` имеется возможность активизировать трехмерный редактор на монтажном столе

Теперь вы должны увидеть на монтажном столе инструмент, изменяющий положение графических объектов в трехмерном пространстве. Захватывая мышью различные маркеры, вы сможете свободно вращать видовое окно по осям X, Y, Z. В качестве упражнения по вращению захватите красный, зеленый или синий маркер в виде дуги, обозначающей вращение по соответствующей оси (рис. 2.54).

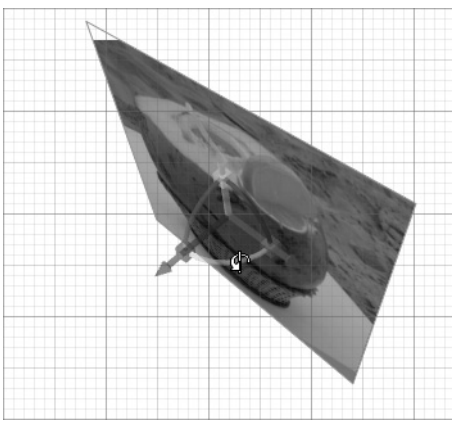


Рис. 2.54. Изменение ориентации трехмерного видового окна инструментом вращения, доступном на монтажном столе

Изменение трехмерных световых эффектов в редакторе освещения

С помощью двух оставшихся составляющих объекта типа `Viewport3D` (`AmbientContainer` и `DirectionalContainer`) можно определить порядок визуализации графического содержимого видового окна с применением различных световых эффектов. *Общий свет* (элемент `AmbientContainer`) служит для освещения видового окна со всех сторон. Этим светом можно равномерно осветить все части объектов. Если выбрать

подчиненный узел `Ambient` на панели Objects and Timeline, как показано на рис. 2.55, то доступными станут свойства общего света в области Light на панели Properties. По умолчанию общий свет имеет нейтрально-серую окраску, но ее можно изменить, выбрав другой цвет в стандартном селекторе, как показано на рис. 2.56, а также на цветной вклейке.

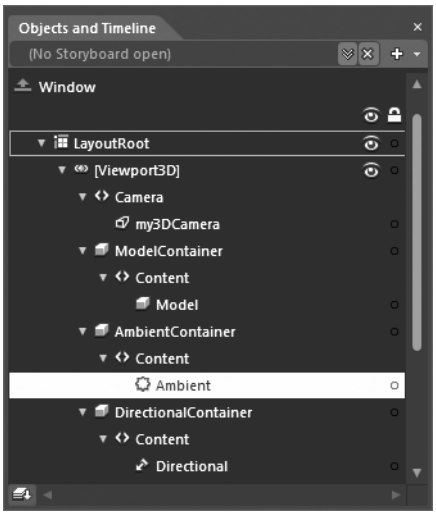


Рис. 2.55. Общее освещение действует равномерно

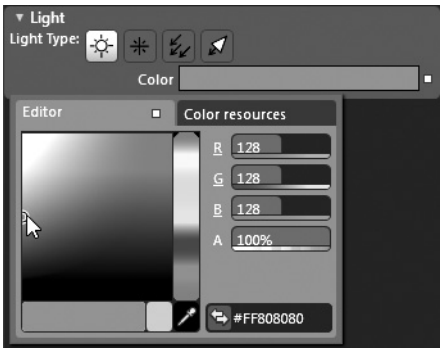


Рис. 2.56. Изменение свойств общего света

Если же выбрать узел `Directional`, подчиненный узлу `DirectionalContainer` (см. рис. 2.55), то появится возможность изменить местоположение источника света. Как показано на рис. 2.57, свойства направленности света можно изменять сообща, щелкнув на пиктограмме, обозначающей направление света, и перетащив соответственно курсор мыши.

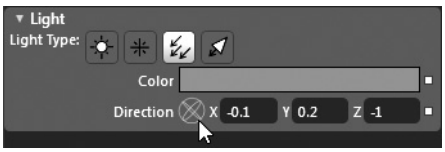


Рис. 2.57. Изменение местоположения источника света

На этом краткое введение в общие средства и способы настройки трехмерного видового окна в Expression Blend завершается. А для того чтобы завершить текущий пример проекта, добавим в него немного кода управления камерой.

Управление камерой в коде

Можете открыть файл исходного кода из рассматриваемого здесь проекта `Wpf3DExample` из загружаемого архива примеров проектов к этой книге, чтобы сэкономить время на ввод кода и заодно исключить ошибки его набора. Ведь о манипулировании элементами управления в среде Expression Blend IDE речь по-настоящему пойдет лишь в следующей главе. А в исходном коде рассматриваемого здесь проекта к объекту главного окна (`Window`) добавлены три ползунковых элемента управления (`Slider`) и несколько элементов управления для описательных меток (`Label`). Благодаря этому пользователь данного приложения может изменять положение камеры по осям X, Y, Z.

Примечание. Непременно просмотрите определения каждого элемента управления `Slider` в исходной разметке XAML, чтобы выяснить минимальный и максимальный пределы преобразований по каждой из осей координат.

В файле исходного кода C# определены три закрытые переменные экземпляра `xVal`, `yVal`, `zVal` типа `double`, а также организована обработка события `ValueChanged`, наступающего при изменении значения в каждом из элементов управления `Slider`. В обработчиках подобных событий координаты X, Y, Z обновляются правильными значениями, а затем вызывается вспомогательная функция `ChangeCamera()`, как показано ниже.

```
private void sliderXChange_ValueChanged(object sender,
  System.Windows.RoutedPropertyChangedEventArgs<double> e)
{
  // Изменить вид в направлении оси координат X.
  xVal = e.NewValue;
  ChangeCamera();
}
```

В методе `ChangeCamera()` задаются свойства `Position` (Положение) и `LookDirection` (Направление взгляда) объекта камеры вида в перспективе, которой рекомендуется присвоить имя `my3DCamera`.

```
private void ChangeCamera()
{
  this.my3DCamera.Position = new
    System.Windows.Media.Media3D.Point3D(-xVal, -yVal, -zVal);
  this.my3DCamera.LookDirection = new
   System.Windows.Media.Media3D.Vector3D(xVal, yVal, zVal);
```

Если вы теперь выполните рассматриваемый здесь пример проекта, то обнаружите, что изображение коктейля “Шальной кокос” вращается и скручивается при перемещении каждого ползунка. На рис. 2.58 созданное приложение показано в действии.

Рис. 2.58. Управление камерой в коде!

Краткий обзор инструментального средства ZAM 3D

Возможности проецировать двухмерные изображения в трехмерном пространстве действительно впечатляют. Но рассмотренные до сих пор способы, позволяющие сделать это, на самом деле не являются подлинно трехмерными в том отношении, что данные двухмерного изображения не обладают всеми признаками полноценной трехмерной визуализации. В частности, данные не отображают полностью поворот изображения к задней его стороне при вращении камеры на 180°.

Как упоминалось ранее, в Expression Blend не предусмотрена какая-либо внутренняя поддержка для построения полноценных трехмерных моделей. Правда, такую поддержку обеспечивают другие, сторонние инструментальные средства. К числу наиболее распространенных инструментальных средств данного типа относится ZAM 3D[6]. Этот программный продукт дает возможность создавать полноценную трехмерную графику, применять многочисленные эффекты к графическим данным и экспортировать их в формате XAML. Нетрудно догадаться, что полученный в итоге XAML-документ может быть импортирован в приложение WPF, разрабатываемое в среде Expression Blend IDE. После этого для манипулирования изображением в процессе разработки и управления им в коде можно воспользоваться описанными выше инструментами, предназначенными для работы с трехмерной графикой. На рис. 2.59 приведен простой трехмерный тор, созданный в ZAM 3D и импортированный в новый проект приложения WPF, разрабатываемого в среде Expression Blend IDE.

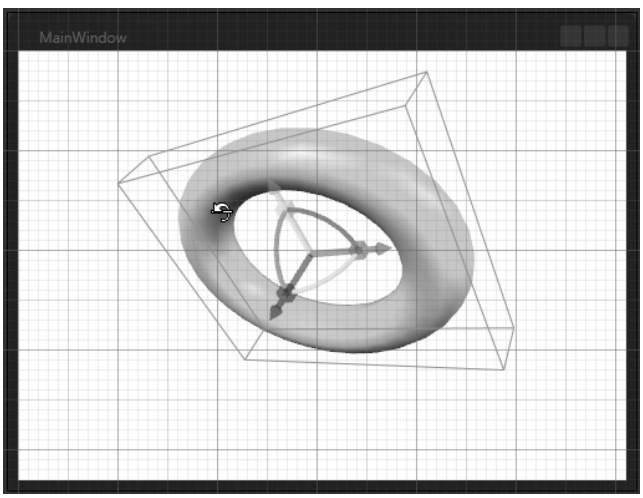


Рис. 2.59. В Expression Blend можно манипулировать полноценными моделями трехмерных объектов, но не создавать их

Мы не будем останавливаться на том, как пользоваться ZAM 3D, поскольку для описания этого инструментального средства потребовалась бы отдельная книга. Но если хотите ознакомиться с ZAM 3D поближе, загрузите оценочную версию этого инструментального средства и опробуйте его на практике.

Исходный код. Исходный код примера проекта `Wpf3DExample` находится в папке `Ch 2 Code` загружаемого архива примеров проектов к данной книге.

---

[6] Оценочную копию ZAM 3D можно загрузить по адресу `http://www.erain.com`.

Введение в трехмерную графику на платформе Silverlight

В прикладном интерфейсе Silverlight API также поддерживается манипулирование трехмерными данными, а результаты получаются сходными с теми, что и в рассмотренном выше примере проекта приложения WPF, где двухмерное изображение накладывалось на трехмерную плоскость. Но на платформе Silverlight в настоящее время отсутствует такая же полная поддержка трехмерной графики, как и на платформе WPF, а вместо нее предоставляется упрощенная среда для трехмерной графики в перспективе[7].

В частности, вместо того чтобы работать с элементами типа `Viewport3D` и соответствующими объектами камеры и освещения, в Silverlight можно изменять положение любого объекта, производного от базового класса `UIElement`, по осям координат X, Y, Z с помощью свойства `Projection` (Проекция). В этом свойстве может быть задан объект типа `PlaneProjection` (Проекция на плоскость), являющийся упрощенным вариантом объекта типа `Viewport3D`. В своей простейшей форме объект типа `PlaneProjection` может быть настроен на вращение с помощью свойств `RotationX`, `RotationY` и `RotationZ`. А управлять центральной точкой его вращения можно с помощью свойств `CenterOfRotationX`, `CenterOfRotationY` и `CenterOfRotationZ`, в которых задается как глобальное, так и локальное смещение данной точки.

Как и следовало ожидать, в среде Expression Blend IDE предоставляется отдельный редактор, доступный на панели Properties для полноценной настройки объекта типа `PlaneProjection` и его связывания с объектом типа `UIElement`. В качестве примера откройте проект `Silverlight3DExample` из папки `Ch 2 Code` загружаемого архива примеров проектов к данной книге. В этом проекте упоминавшееся ранее изображение коктейля "Шальной кокос" переносится на веб-страницу. В силу некоторых отличий между прикладными интерфейсами WPF API и Silverlight API данные изображения сохранены в файле формата PNG, поскольку в элементе управления `Image` на платформе Silverlight поддерживаются файлы изображений только формата JPEG или PNG. А в остальном компоновка пользовательского интерфейса, представленного в данном примере объектом типа `UserControl`, оказывается такой же, как и в рассмотренном выше примере проекта на платформе WPF.

Если выбрать на монтажном столе элемент управления `Image`, то на панели Properties появится область Transform с подобластью Projection, как показано на рис. 2.60. В этой подобласти можно настроить свойство `Projection` любого графического элемента, выбранного на монтажном столе.

Как и в остальных элементах управления пределами на панели Properties, для настройки значений координат проецирования X, Y, Z достаточно щелкнуть кнопкой мыши на соответствующем элементе управления и, не отпуская ее, переместить курсор мыши. А вместо объекта типа `Viewport3D` в разметке, формируемой в коде XAML, в данном случае обнаруживается описание объекта `PlaneProjection`, как показано ниже.

```
<Image x:Name="imgCoCoLoCo" Margin="162,18,162,185"
      Source="/CoCoLoCo.png" Stretch="Fill">
  <Image.Projection>
    <PlaneProjection RotationX="46"
                RotationY="12" RotationZ="-29"/>
  </Image.Projection>
</Image>
```

[7] Было бы неверно утверждать, что Silverlight вообще нельзя пользоваться для построения весьма экзотичных образцов трехмерной визуализации. Если хотите убедиться в подлинных возможностях этой платформы в отношении трехмерной графики, откройте пример проекта `Zune3D`, входящий в состав версии Expression Blend 4. Но имейте в виду, что большая часть функциональных возможностей в данном примере реализована в коде, а не в разметке XAML, и поэтому его рассмотрение выходит за рамки этой книги.

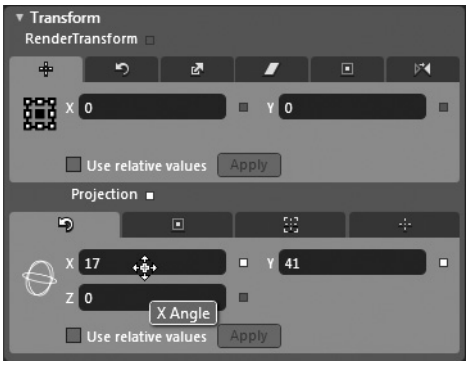


Рис. 2.60. В проекте на платформе Silverlight редактор проекции служит для переноса графических данных на трехмерную плоскость

Управление проекцией в коде

Исходный код, представляющий упомянутый выше элемент управления `UserControl` в рассматриваемом здесь примере проекта на платформе Silverlight, в какой-то степени похож на исходный код из предыдущего примера проекта на платформе WPF. В нем по-прежнему определяются три переменные экземпляра типа `double` для хранения текущих значений координат X, Y, Z, устанавливаемых с помощью элементов управления `Slider`, а в каждом обработчике событий `ValueChanged` вызывается вспомогательная функция `ChangeCamera()`. Но на этот раз в реализации данной функции настраивается объект типа `PlaneProjection` и устанавливается свойство `Projection` элемента управления `Image` (под названием `imgCoCoLoCo`), как показано ниже.

```
private void ChangeCamera()
{
  PlaneProjection pp = new PlaneProjection();
  pp.RotationX = xVal;
  pp.RotationY = yVal;
  pp.RotationZ = zVal;
  if(imgCoCoLoCo != null)
    this.imgCoCoLoCo.Projection = pp;
}
```

Если вы теперь запустите данный проект на выполнение в среде Expression Blend IDE, то сможете снова вращать и закручивать изображение, но на этот раз в окне веб-браузера.

Исходный код. Исходный код примера проекта `Silverlight3DExample` находится в папке `Ch 2 Code` загружаемого архива примеров проектов к данной книге.

Назначение ресурсов объектов

В заключение этой главы, посвященной графическим возможностям Expression Blend, рассмотрим назначение ресурсов объектов. Когда в среде Expression Blend формируются такие визуальные ресурсы, как специальные кисти, стили, шаблоны элементов управления и так далее, невольно возникает ситуация, в которой требуется еще раз воспользоваться отдельным ресурсом подобного рода. Допустим, вы создали идеальную кисть для раскраски градиента, на что вам потребовалось немало времени. Было бы очень неприятно воссоздавать те же самые действия в редакторе кистей для повторного формирования

разметки XAML, описывающей данную кисть. Еще более неприятная ситуация может возникнуть в том случае, если вам снова потребуется кисть, созданная несколько недель назад, а вы, естественно, не помните точно все свои действия в редакторе кистей, чтобы воссоздать эту кисть. Конечно, вы *могли* бы скопировать и вставить соответствующую разметку из одного проекта в другой, но это было бы в лучшем случае обременительно.

К счастью, на обеих платформах, WPF и Silverlight, поддерживается принцип *ресурса объекта*, называемого просто ресурсом. По существу, ресурс представляет собой именованный большой двоичный объект XAML, хранящийся в текущем проекте. Так, если требуется обратиться к такому объекту из элементов управления пользовательского интерфейса, его можно выбрать по присвоенному ему имени. В Expression Blend имеется целый ряд инструментов для определения, манипулирования и неоднократного использования ресурсов. На рассматриваемом далее примере манипулирования объектом специальной кисти будет показано, как пользоваться ресурсами кистей, а другие виды ресурсов (стилей, шаблонов и пр.) будут рассмотрены в последующих главах. Правда, независимо от типа ресурса процесс разработки приложений в Expression Blend остается неизменным.

Создание ресурсов в Expression Blend

Ради соблюдения принципа модульности в максимальной степени создайте новый проект приложения WPF, присвоив ему имя `BlendResources`, хотя в качестве отправной точки можете воспользоваться любым проектом, над которым работали ранее в этой главе. Затем нарисуйте произвольную геометрическую форму на монтажном столе инструментом Pencil и задайте в ее свойстве `Fill` специальную, сложную кисть для заполнения этой формы градиентом. По завершении обратите внимание на пиктограмму квадратика справа от свойства `Fill` (или другого свойства раскраски кистью) в редакторе кистей. Как упоминалось в главе 1, этой пиктограммой обозначается кнопка выбора дополнительных параметров настройки. Щелкнув на этой кнопке, вы увидите меню с выбранной командой Convert to New Resource (Преобразовать в новый ресурс), как показано на рис. 2.61.

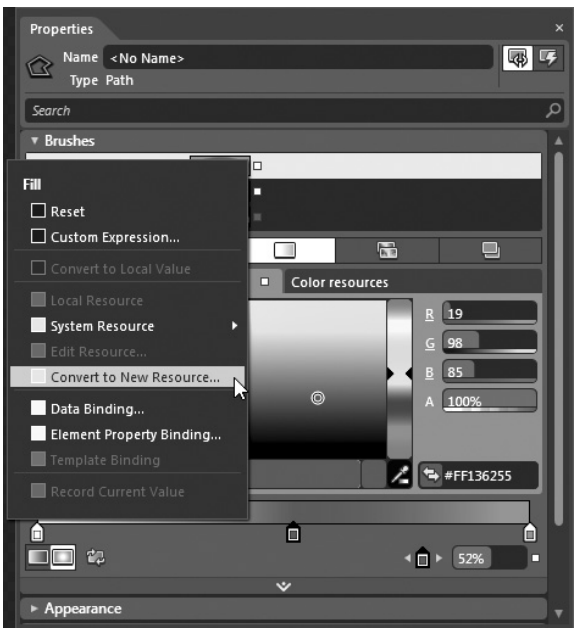


Рис. 2.61. Извлечение ресурса кисти

Как только вы выберете упомянутую выше команду, откроется диалоговое окно, в котором вам будет предложено ввести ряд важных данных. Прежде всего вам нужно присвоить подходящее имя ресурсу объекта для созданной кисти, например `myBrush`. Затем следует указать место для хранения ресурса. И для этого в вашем распоряжении имеются три варианта выбора, описываемые в табл. 2.3.

Таблица 2.3. Варианты хранения ресурсов Expression Blend

Местоположение ресурса	Описание
Приложение	Ресурсы перемещаются в файл `App.xaml` текущего проекта. Благодаря этому ресурсы могут быть использованы в любой части проекта на платформе WPF или Silverlight
Текущий документ	Ресурс может быть повторно использован только в объекте типа `Window` или `UserControl` текущего проекта WPF или Silverlight соответственно. Это удобно в том случае, если разрабатывается сложная система компоновки пользовательского интерфейса приложения с использованием вложенных диспетчеров компоновки и требуется, чтобы заданный ресурс объекта использовался во всех подчиненных компонентах
Словарь ресурсов	В этом случае можно создать новый XAML-документ, называемый *словарем ресурсов* и содержащий только ресурсы объектов. Такой вариант отлично подходит для создания ресурсов, применяемых в разных проектах, поскольку словарь ресурсов можно добавить в виде XAML-документа в новый проект, разрабатываемый в среде Expression Blend

В данном примере ресурс `myBrush` будет помещен в словарь ресурсов, находящийся в файле `MyResources.xaml`. Щелкните на кнопке New, чтобы создать этот файл. На рис. 2.62 приведены окончательные настройки для извлечения текущего ресурса.

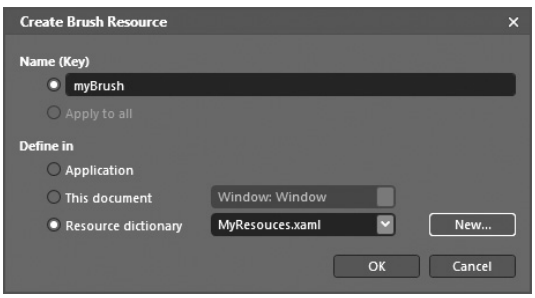


Рис. 2.62. Создание нового словаря ресурсов в Expression Blend

В итоге вы обнаружите, что вновь созданный XAML-файл добавлен в текущий проект, а файл разметки приложения `App.xaml` видоизменен с целью объединения с внешним файлом ресурсов. Проверьте и то и другое на панели Projects. А если вы проверите разметку формы, использованной для хранения специально созданной кисти, то увидите, что в расширении разметки, сформированном в коде XAML, делается ссылка на данный ресурс по имени, как показано ниже[8].

```
<Path ... Fill="{DynamicResource myBrush}"/>
```

[8] В проектах WPF, разрабатываемых в среде Extension Blend, расширение разметки {DynamicResource} используется по умолчанию. Этим гарантируется, что если ресурс изменится в коде, то все использующие его элементы пользовательского интерфейса автоматически обновятся. Но в проектах Silverlight используется расширение разметки {StaticResource}, которое поддерживается и в проектах WPF, хотя и не подразумевает автоматическое внесение изменений во время выполнения.

Управление имеющимися ресурсами

Как только ресурс будет извлечен, можете видоизменить его впоследствии несколькими способами. В данном примере ресурсом оказывается специально созданная кисть, поэтому можете выбрать последнюю вкладку Brush Resources в редакторе кистей, чтобы получить доступ ко всем ресурсам объектов кистей в своем проекте. Если же вы щелкнете на пиктограмме любого ресурса, то тем самым откроете соответствующий редактор кисти, чтобы изменить ресурс по ходу разработки, как показано на рис. 2.63.

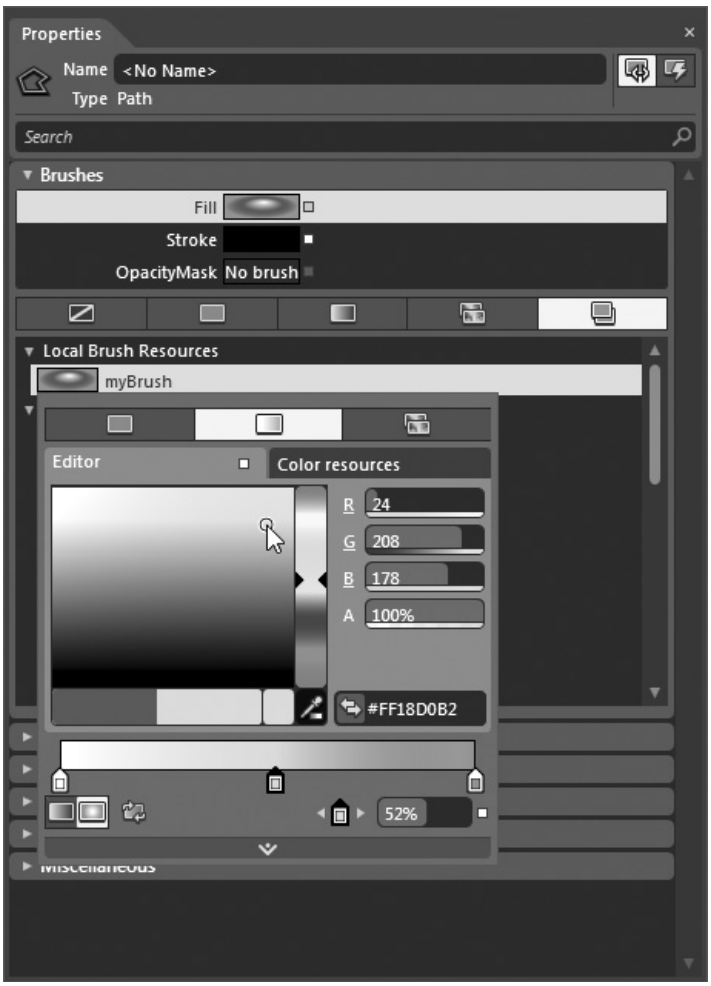


Рис. 2.63. Видоизменение имеющегося ресурса

Еще один способ просмотра и видоизменения ресурсов в приложении состоит в использовании панели Resources, расположенной с правой стороны рабочего окна Expression Blend рядом с панелью Properties. Эта панель удобна тем, что каждый ресурс можно просмотреть на ней не только по имени, но и по месту его хранения (в объекте типа `Window`, в словаре ресурсов и т.д.). И в этом случае нужный ресурс можно выбрать для последующей правки, как показано на рис. 2.64.

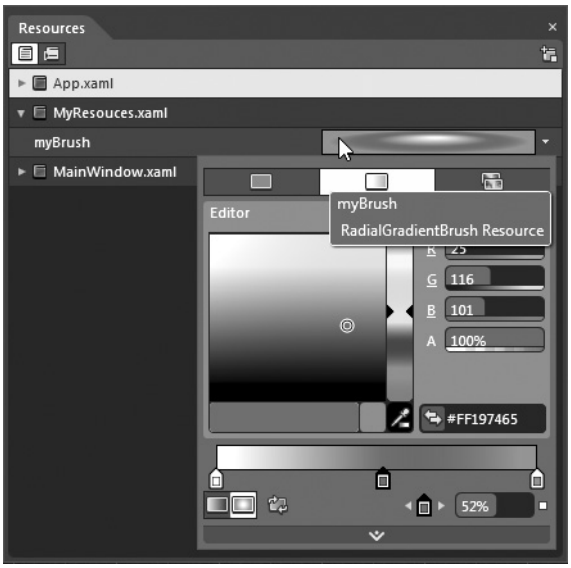


Рис. 2.64. Панель Resources

Применение ресурсов при создании новых элементов пользовательского интерфейса

И последнее и самое главное, что можно сделать с ресурсами объектов, — использовать их повторно при создании пользовательского интерфейса приложения. Для этого достаточно выбрать нужный ресурс по имени в соответствующем редакторе. В рассматриваемом здесь примере используется ресурс специально созданной кисти `myBrush`, поэтому можете выбрать его из редактора кистей при установке свойств раскраски кистью. Если вы добавите элемент пользовательского интерфейса на монтажном столе, например элемент управления `Button`, то можете перейти на вкладку Brush Resources в редакторе кистей, где и находится данный ресурс. На рис. 2.65 показано, каким образом свойство `Background` (Фон) элемента управления типа `Button` настраивается на применение кисти, специально созданной и сохраненной ранее в качестве ресурса объекта.

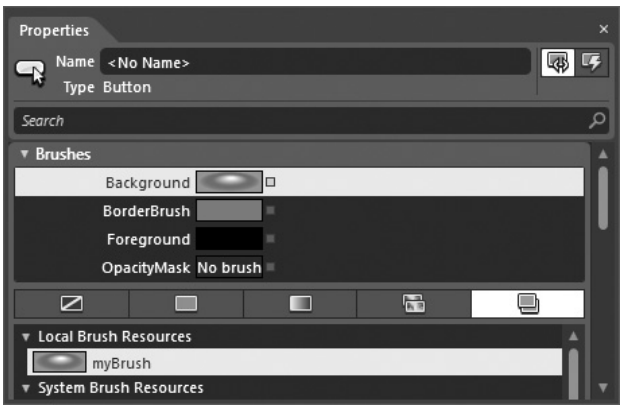


Рис. 2.65. Применение имеющегося ресурса

На этом рассмотрение возможностей работы с векторной и трехмерной графикой, а также с ресурсами объектов в среде Expression Blend IDE завершается. В последующих главах будут представлены другие способы манипулирования графическими данными, а до тех пор вы должны как следует усвоить самые основы. Кроме того, в последующих главах будут продемонстрированы дополнительные способы фиксации ресурсов объектов для последующего их применения, но и того, что вы уже знаете, достаточно для управления подобными ресурсами.

Исходный код. Исходный код примера проекта `BlendResources` находится в папке `Ch 2 Code` загружаемого архива примеров проектов к данной книге.

Резюме

В этой главе был представлен целый ряд способов и средств, поддерживаемых в Expression Blend, для создания и манипулирования векторной графикой. Сначала в ней были рассмотрены основные инструменты рисования (Pen, Pencil и т.д.), а также категория Shapes библиотеки ресурсов. Как было показано на конкретных примерах, цвета, используемые для обводки границ и заполнения внутреннего пространства геометрических форм, могут быть специально подобраны во встроенном редакторе кистей и дополнены различными визуальными эффектами. Затем было показано, каким образом векторная графика может быть сформирована в Expression Design, экспортирована в формате XAML и затем импортирована в проект, разрабатываемый в Expression Blend.

Далее были рассмотрены инструменты, предназначенные для выполнения двух- и трехмерных преобразований над элементами пользовательского интерфейса. В проектах приложений WPF и Silverlight используются одни и те же редакторы, когда требуется выполнить такие двухмерные преобразования, как переворачивание, вращение и наклон графического элемента. А вот трехмерные преобразования несколько отличаются на платформах WPF и Silverlight. Если на платформе WPF поддерживается полноценная модель трехмерного программирования, позволяющая манипулировать камерами, освещением и дополнительными слоями, которые в этой книге не рассматриваются, то на платформе Silverlight поддерживаются проекции на плоскость для получения аналогичного, хотя и более упрощенного визуального эффекта.

И в завершение главы было сделано краткое введение в ресурсы объектов. Следует иметь в виду, что в Expression Blend предоставляется целый ряд инструментов, которые позволяют неоднократно использовать, видоизменять и применять различные графические ресурсы. Дополнительные примеры извлечения графических ресурсов будут представлены в последующих главах.

ГЛАВА 3

Редактор анимации

В этой главе будут представлены инструменты анимации, доступные в Expression Blend. Если у вас имеется некоторый опыт создания вручную анимационных последовательностей в формате XAML при разработке приложений в Visual Studio, то вам должно быть известно, насколько трудоемким может оказаться этот процесс[1]. К счастью, в Expression Blend интегрирован довольно развитый редактор анимации, в котором можно без особого труда фиксировать изменения состояния объектов, используя простой принцип временной шкалы. Читая эту главу, вы узнаете, каким образом анимация образуется из *раскадровок* и *ключевых кадров*. Каждый ключевой кадр анимации отвечает за изменение значения указанного свойства целевого объекта.

Научившись сначала пользоваться основными инструментами анимации, вы затем откроете для себя назначение эффектов инерционности движения в анимации. По существу, *эффекты инерционности движения* упрощают внедрение физических явлений в анимацию, в том числе отскакивание, притягивание и упругость. Кроме того, в состав Expression Blend входит редактор ключевых сплайнов, в котором можно точнее изменять интерполируемые значения (т.е. эффекты ускорения) при подходе к заданному ключевому кадру или выходе из него.

И в заключение главы будет вкратце изложен вопрос, более подробно рассматриваемый в главе 4, а именно: назначение разных *объектов поведения*. Здесь вы ознакомитесь с объектом поведения типа `ControlStoryboardAction`, благодаря которому взаимодействие с раскадровкой полностью реализуется в разметке.

Назначение служб анимации

Как отмечалось в главе 2, ясное представление о том, как следует работать с графикой, имеет решающее значение для успешной разработки приложений на платформах WPF и Silverlight, независимо от того, что по этому поводу думают сами разработчики. Аналогично, следует отметить важную роль служб анимации при построении реальных приложений WPF и Silverlight промышленного уровня. Что бы вы ни думали по этому поводу, применение служб анимации никак не связано с процессом создания видеоигр и приложений, насыщенных мультимедийным содержимым, хотя службы анимации будут, очевидно, полезными и в этих случаях.

На платформе WPF или Silverlight под *анимацией* подразумевается простое изменение значения свойства объекта в течение времени. Так, если требуется управлять

[1] Если вам приходилось когда-нибудь создавать анимацию в других средах разработки, вы будете приятно удивлены, насколько просто это делается на платформах WPF и Silverlight. В подавляющем большинстве случаев теперь уже нет необходимости создавать потоки вручную, стирать и перерисовывать изображения, создавать закадровые буферы и выполнять неприятные расчеты прямоугольников.

изменением цвета фона объекта от ярко-зеленого до темно-зеленого оттенка в течение пяти секунд, можно воспользоваться *анимацией кисти*. А если требуется перемещать специальный графический элемент по неподвижной геометрической линии, то в этом случае можно воспользоваться *анимацией по траектории*. Читая эту и остальные главы данной книги, вы обнаружите, что практически любое свойство, доступное на панели Properties в среде Expression Blend, может стать целевым для служб анимации.

Область применения служб анимации

Подобно графике, анимация появляется в самых неожиданных местах. Как будет показано в главе 5, чаще всего анимация применяется для внедрения визуальных подсказок в специальные стили оформления или шаблоны элементов управления. Например, с помощью анимации можно легко определить внешний вид специального элемента управления, когда курсор наводится на него, когда курсор покидает его или же когда производится щелчок кнопкой мыши на его поверхности. Помимо этого, можно определить дополнительные виды анимации, определяющие внешний вид элемента управления, когда он получает логический фокус, теряет фокус и в остальных случаях.

Анимацию можно использовать для четкого и плавного перехода между значениями свойств объектов. В качестве примера можете создать объект типа `Window` в проекте приложения на платформе WPF и осуществить анимацию вращения диспетчера компоновки (со всеми элементами управления, которые он содержит) в трехмерной плоскости. А в проект приложения на платформе Silverlight можете добавить анимацию переворачивания страницы как при просмотре книжки с иллюстрациями. Если же вы занимаетесь созданием видеоигры, воспользуйтесь анимацией для перемещения стен лабиринта, вывода вспышкой на экран сообщения “Игра окончена” и т.д. Следует иметь в виду, что применение служб анимации стало обычным явлением в проектах приложений на платформах WPF и Silverlight, и в среде Expression Blend IDE подобные визуальные эффекты достигаются довольно просто.

Рабочее пространство анимации в Expression Blend

Как пояснялось в главе 1, панель Objects and Timeline служит для редактирования объекта, выбранного на монтажном столе. Оказывается, что та же самая панель служит местом доступа к редактору анимации в Expression Blend. Для того чтобы приступить к изучению средств создания анимации в среде Expression Blend IDE, создайте новый проект приложения Silverlight, присвоив ему имя `SimpleBlendAnimations`[2].

Первым шагом на пути к созданию анимации в Expression Blend должен стать выбор одного или нескольких объектов в качестве целевых для логики анимации. В настоящий момент во вновь созданном проекте приложения Silverlight имеется только объект `LayoutRoot` типа `Grid`, а также объект типа `UserControl`, определяющий собственно пользовательский интерфейс. Конечно, можно осуществить анимацию свойств и этих объектов, но рассматриваемый здесь пример окажется более интересным и полезным, если добавить новый объект на монтажном столе. С этой целью можете ввести объект любого типа, будь то `Button`, `Rectangle` или контур, нарисованный инструментом Pen либо Pencil, но ради примера создайте простой объект типа `Ellipse`, присвоив ему имя

[2] В прикладном интерфейсе WPF API поддерживается ряд дополнительных средств создания анимации, недоступных на платформе Silverlight. Способы, рассматриваемые в первом примере проекта, одинаково применимы для создания анимации независимо от используемого прикладного интерфейса API конкретной платформы.

myCircle (мой круг) и подобрав по своему вкусу сплошной цвет заливки для его свойства Fill (подробнее о том, как это делается в редакторе кистей, см. в главе 2).

Когда вы пользуетесь инструментами анимации в Expression Blend, то, скорее всего, стремитесь изменить компоновку рабочего пространства анимации, активизируемого с помощью команды меню Window⇨Workspaces (Окно⇨Рабочие пространства) или функциональной клавиши <F6>. Нажмите эту клавишу, чтобы увидеть, каким образом реорганизуются панели в рабочем окне среды Expression Blend IDE, а панель Objects and Timeline располагается вдоль нижнего края этого окна (рис. 3.1).

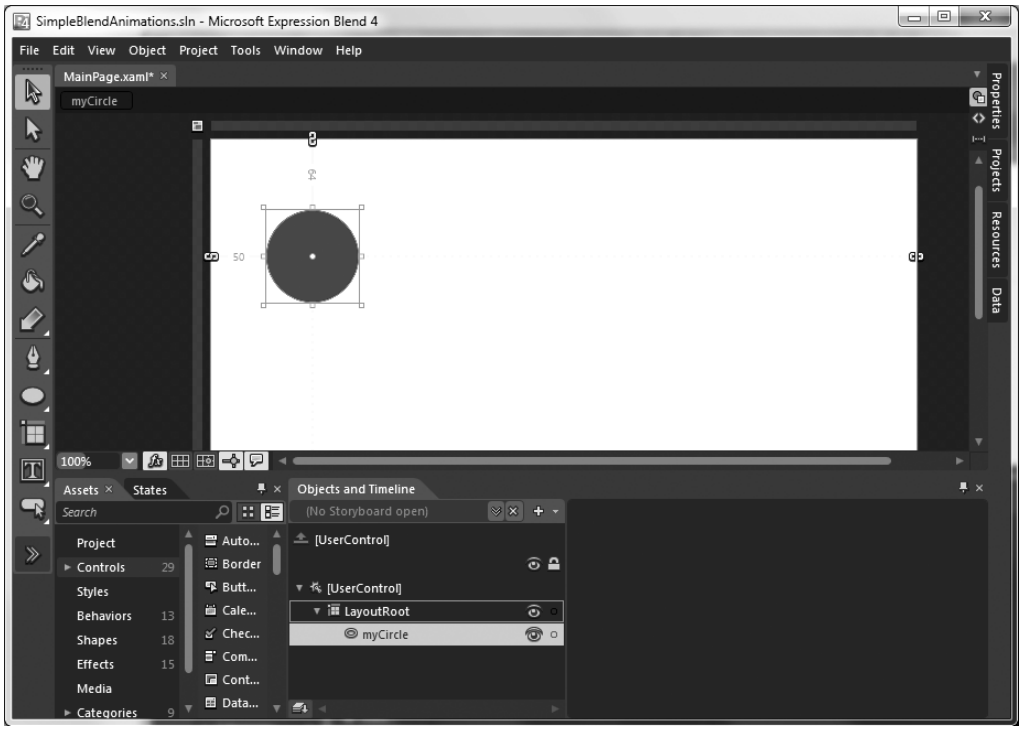


Рис. 3.1. Компоновка рабочего пространства анимации

Примечание. Нажимая функциональную клавишу <F6>, можете переходить от стандартного рабочего пространства конструирования (Design) к рабочему пространству анимации (Animation) и обратно.

Создание новой раскадровки

Анимация на платформе WPF или Silverlight фиксируется в формате XAML с использованием *раскадровки*. Следовательно, когда возникает потребность в получении новой анимационной последовательности, этот процесс вы должны начинать с создания новой раскадровки на панели Objects and Timeline. С этой целью щелкните на кнопке New со знаком + на пиктограмме, как показано на рис. 3.2. (В данный момент не имеет особого значения, какой именно объект выбран в иерархическом представлении объектов на панели Objects and Timeline.)

Как только вы щелкнете на этой кнопке, вам будет предложено присвоить новой раскадровке особое имя, например AnimateCircle (анимация круга), (рис. 3.3).

Присваивание имени раскадровке имеет большое значение, поскольку по этому имени вам придется манипулировать анимацией как в коде, так и в разметке XAML.

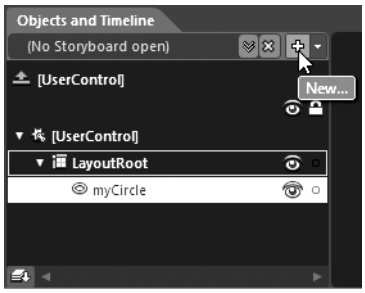


Рис. 3.2. Создание новой раскадровки

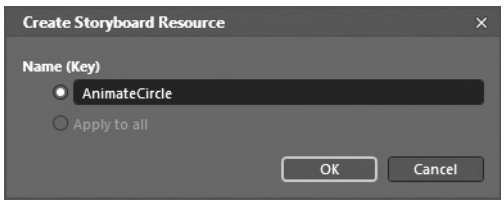


Рис. 3.3. Именование раскадровки

Следует также иметь в виду, что новая раскадровка анимации сохраняется в Expression Blend в качестве ресурса (см. главу 2) в текущем объекте типа `UserControl` на платформе Silverlight или объекте типа `Window` на платформе WPF. Так, если вы откроете окно для просмотра содержимого монтажного стола в коде XAML, то обнаружите разметку, выделенную на рис. 3.4.

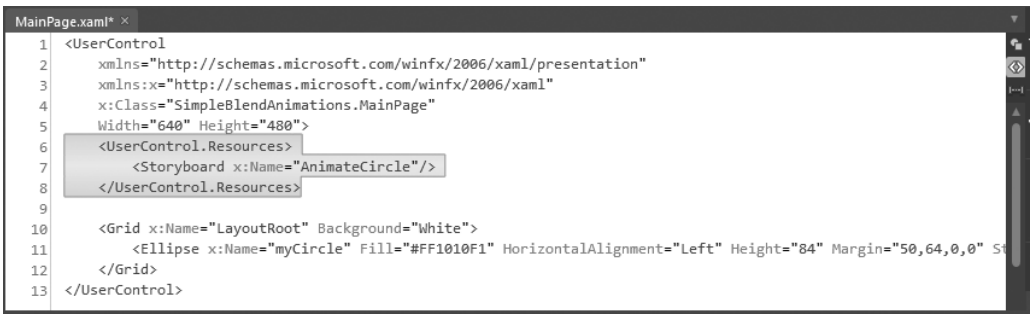


Рис. 3.4. Раскадровки сохраняются в виде ресурсов, формируемых на уровне документов

Примечание. В отличие от типичных ресурсов объектов, рассматривавшихся в главе 2, раскадровки тесно связаны с объектом, для анимации которого они предназначены. Следовательно, создавать раскадровку на уровне всего приложения нецелесообразно. Если вы предполагаете пользоваться созданной анимационной последовательностью неоднократно, то разместите соответствующую раскадровку в специальном стиле оформления, шаблоне или объекте типа `UserControl`, чтобы впоследствии извлечь из него содержащуюся в нем анимацию. Более подробно эти вопросы рассматриваются в главе 5.

Управление имеющимися раскадровками

Создав новый объект раскадровки, например `AnimateCircle`, вы сразу же обнаружите его в списке, раскрывающемся на панели Objects and Timeline. В связи с тем что в одном объекте типа `Window` или `UserControl` можно нередко обнаружить довольно большое количество раскадровок, отвечающих за анимацию разных объектов, отдельную раскадровку следует выбирать для правки, используя именно этот раскрывающийся список на панели Objects and Timeline.

Следует также иметь в виду, что раскадровки можно искать по имени аналогично поиску нужных средств в библиотеке ресурсов и на панели Properties, выбрав вариант Search из того же самого раскрывающегося списка, как показано на рис. 3.5.

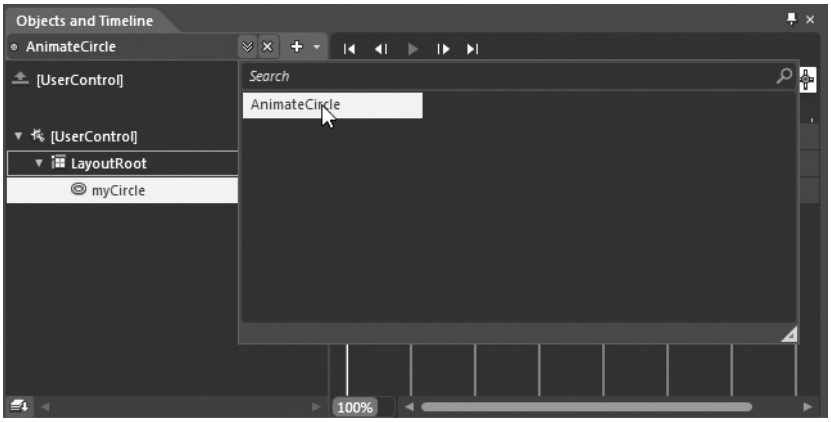


Рис. 3.5. Отдельные раскадровки можно выбрать из раскрывающегося списка или найти в режиме поиска

И еще одно, последнее вводное замечание: раскадровку можно легко переименовать, скопировать или удалить, щелкнув на ней правой кнопкой мыши и выбрав соответствующую команду из всплывающего контекстного меню (рис. 3.6). Так, если вы выберете из этого меню команду Reverse, последовательность ключевых кадров анимации изменится на обратную, что, согласитесь, очень удобно!

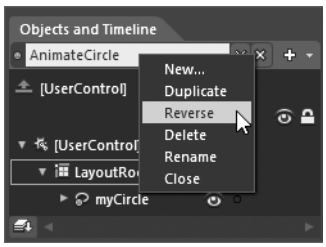


Рис. 3.6. Имеющимися раскадровками можно управлять на панели Objects and Timeline

Ввод ключевых кадров анимации

Создав новую раскадровку или выбрав уже имеющуюся для последующей правки, можете приступать к вводу любого количества ключевых кадров анимации в редакторе временной шкалы. Исследуя текущий редактор временной шкалы, обратите внимание на желтую вертикальную линию напротив нульсекундной отметки анимации. Эта линия обозначает текущий момент времени анимации в данной раскадровке. Прямо над этой желтой линией находится яйцевидная пиктограмма со знаком + справа от нее. Она обозначает кнопку Record Keyframe (Записывать ключевой кадр). Если щелкнуть на этой кнопке, в текущий момент времени анимации будет введен ключевой кадр.

Вводя ключевой кадр, вы должны держать в уме объект, выбранный в настоящий момент на панели Objects and Timeline. Дело в том, что в одной раскадровке можно управлять анимацией целого ряда объектов, находящихся на монтажном столе, а

следовательно, в один и тот же момент времени может присутствовать несколько ключевых кадров анимации. Что же касается рассматриваемого здесь примера, то непременно выберите созданный вами ранее объект типа `Ellipse` и щелкните на кнопке Record Keyframe, чтобы ввести новый ключевой кадр на нульсекундной отметке анимации, как показано на рис. 3.7.

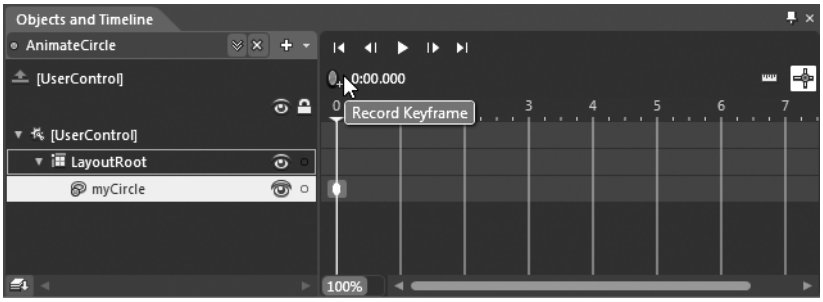


Рис. 3.7. Ввод ключевых кадров на временной шкале

Введя ключевой кадр анимации, обратите внимание то, что теперь монтажный стол обрамлен красной рамкой, а в левом верхнем углу монтажного стола появился небольшой элемент управления, с помощью которого можно включать и выключать режим записи анимационной последовательности (рис. 3.8).

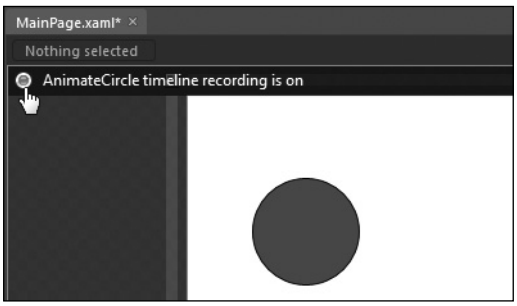


Рис. 3.8. На монтажном столе можно включать и выключать режим записи анимационной последовательности

Примечание. Отдельные ключевые кадры могут быть перемещены в редакторе временной шкалы стандартным методом перетаскивания мышью. Так, если вы ввели ключевой кадр на трехсекундной отметке анимации, но хотите установить его на шестисекундной отметке, можете легко сделать это, перетащив ключевой кадр к нужной отметке анимации. Кроме того, можете удалить или скопировать ключевой кадр, щелкнув на нем правой кнопкой мыши и выбрав соответствующую команду из всплывающего контекстного меню.

Фиксация изменений в свойствах объектов

После ввода первого ключевого кадра, который совсем *не* обязательно устанавливать в нулевой, но в любой удобный для анимации момент времени в зависимости от ее характера, вам нужно ввести хотя бы еще один, дополнительный ключевой кадр, чтобы зафиксировать вместе с первым ключевым кадром промежуток времени, в котором будет происходить данная анимация. Для перемещения по временной шкале щелкните на небольшой

треугольной “шапке” над желтой вертикальной линией и перетащите ее мышью или же щелкните непосредственно на нужном месте временной шкалы. Используя любой из этих двух способов, введите второй ключевой кадр на двухсекундной отметке анимации, т.е. щелкните еще раз на кнопке с яйцевидной пиктограммой, как показано на рис. 3.9.

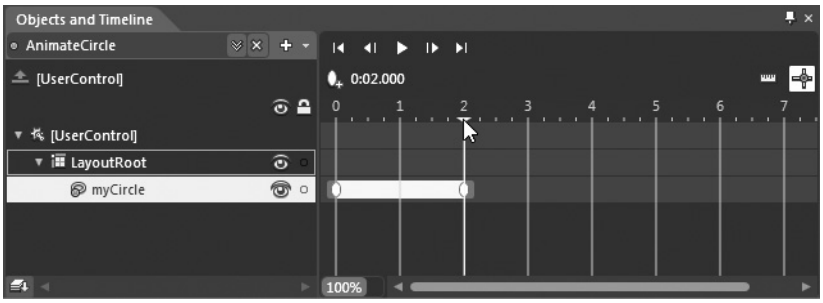


Рис. 3.9. Определение двухсекундного промежутка времени анимации

Далее следует самое интересное. Убедитесь сначала в том, что на монтажном столе выбран объект типа `Ellipse`, а затем воспользуйтесь панелью Properties и монтажным столом, чтобы внести ряд изменений в состояние этого объекта. В частности, можете выделить объект типа `Ellipse` и переместить его в новое положение на монтажном столе. При этом появится небольшая пунктирная линия, обозначающая расстояние, на которое был перемещен объект. Кроме того, можете выбрать новый цвет заливки в свойстве `Fill` объекта типа `Ellipse` посредством редактора кистей или внести изменения в свойства `Height` и `Width`, перейдя к области Layout на панели Properties. Аналогичным образом внесите еще три или четыре изменения в состояние объекта типа `Ellipse`. На рис. 3.10 показано, как выглядит объект `myCircle` по достижении двухсекундной отметки анимации.

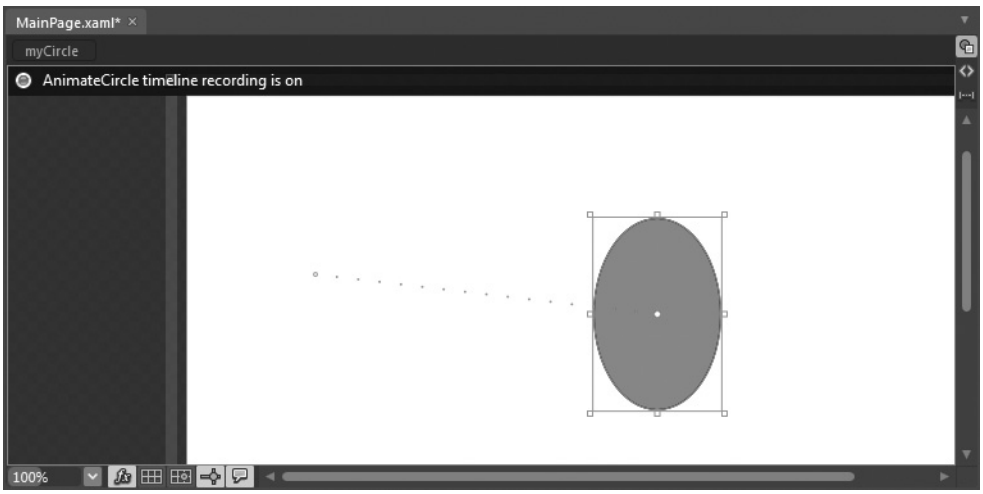


Рис. 3.10. Возможный вид объекта `myCircle` по достижении двухсекундной отметки анимации (у вас он может выглядеть иначе)

Проверка анимации

Теперь можете проверить созданную вами анимацию, щелкнув на кнопке Play (Воспроизвести), находящейся в области инструментов на временной шкале (рис. 3.11).

В итоге окружность плавно перейдет из своего исходного состояния в конечное на протяжении двух секунд анимации.

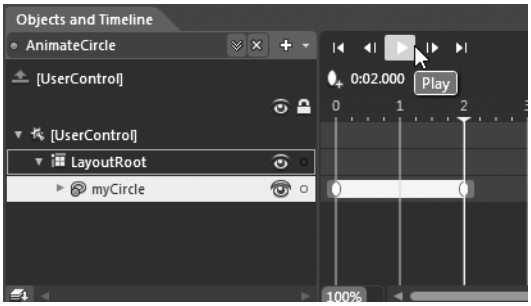


Рис. 3.11. Проверка созданной анимации в режиме воспроизведения

Упомянутые выше инструменты действуют аналогично органам управления на типичном цифровом универсальном проигрывателе и позволяют переходить от предыдущих кадров анимации к следующим, а также к первому и последнему кадру раскадровки. Уделите время опробованию этих инструментов и научитесь перемещать желтую вертикальную линию по временной шкале, захватив и перетащив мышью ее "шапку" или просто щелкнув непосредственно в нужном месте временной шкалы.

Просмотр разметки анимации

Прежде чем двигаться дальше, уделите время просмотру исходной разметки, автоматически сформированной в коде XAML средствами Expression Blend IDE. Как видите, элемент разметки `<Storyboard>` заполнен целым рядом новых подчиненных элементов, определяющих порядок анимации объекта типа `Ellipse`. Не вдаваясь в подробности этой разметки, следует все же обратить внимание на то, что объекту анимации (в данном случае `ColorAnimationUsingKeyFrames`) известно, свойства каких именно объектов должны изменяться и в какие именно моменты анимации это должно происходить. Для первой цели служат свойства `TargetProperty` (Свойство целевого объекта) и `TargetName` (Имя целевого объекта), а для второй — свойство `KeyTime` (Время наступления ключевого кадра) объекта анимации.

```
<ColorAnimationUsingKeyFrames
  Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
  Storyboard.TargetName="myCircle">
  <EasingColorKeyFrame KeyTime="0" Value="#FF1010F1"/>
  <EasingColorKeyFrame KeyTime="0:0:2" Value="#FFF110DE"/>
</ColorAnimationUsingKeyFrames>
```

Настройка свойств раскадровки

Если выбрать объект раскадровки из раскрывающегося списка на панели Objects and Timeline (см. рис. 3.5), то свойства самой раскадровки можно настроить на панели Properties[3]. Как показано на рис. 3.12, раскадровка может быть настроена на режимы автоматического изменения на обратное направления воспроизведения анимации. Так, если установить флажок свойства `AutoReverse` (Автоматически изменить направление

[3] Имеется также возможность настраивать отдельные ключевые кадры. О том, как это сделать, речь пойдет ниже, когда будет рассматриваться назначение эффектов инерционности движения в анимации.

на обратное), вся анимация займет уже четыре секунды: по две секунды в каждом направлении ее воспроизведения.

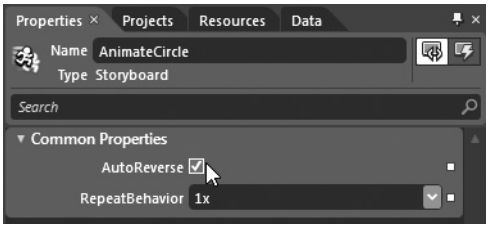


Рис. 3.12. Активизация режимов автоматического изменения на обратное направление воспроизведения анимации

Имеется также возможность настроить режим повторения анимации, выбрав из раскрывающегося списка свойства `RepeatBehavior` один следующих четырех вариантов: `1x` (Однократно), `2x` (Двукратно), `3x` (Трехкратно) и **`Forever`** (Бесконечно). Но вы вольны ввести любое число повторений анимации, дополнив его буквой **`x`**. Так, если вам требуется воспроизвести анимацию пять раз, введите **`5x`** в текстовом поле свойства `RepeatBehavior`.

В текстовом поле свойства `RepeatBehavior` можно также ввести числовое значение, обозначающее конкретное *время* воспроизведения анимации, а не число ее повторений. Для того чтобы указать единицу изменения времени, введите заданное время в приведенном ниже формате.

```
дни:часы:минуты:секунды:доли_секунды
```

Указывать `дни` и `доли_секунды` необязательно. Как правило, заданное время указывается в следующем обобщенном формате:

```
часы:минуты:секунды
```

Так, если требуется, чтобы анимация воспроизводилась в течение 15 секунд, а затем прекращалась полностью независимо от того, осуществляется ли в ней циклический проход всех ключевых кадров, следует ввести значение **`00:00:15`** в текстовом поле свойства `RepeatBehavior`, как показано на рис. 3.13.

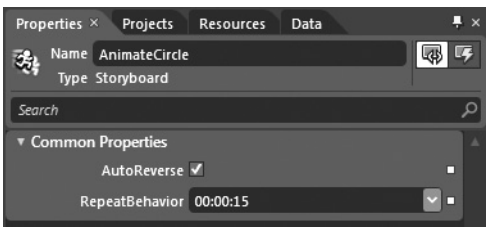


Рис. 3.13. Указание времени воспроизведения анимации, включая ее повторение, если таковое имеет место

Изменение масштаба временной шкалы

Прежде чем переходить к рассмотрению способов взаимодействия с раскадровками в коде, следует отметить еще одну особенность редактора анимации в Expression Blend. В левом нижнем углу этого редактора находится элемент, управляющий измене-

нием масштаба временной шкалы. По умолчанию в нем установлен масштаб 100%, а следовательно, на временной шкале продолжительность и отдельные моменты времени анимации указываются в секундах. Если же требуется более мелкая градация времени для установки ключевых кадров в моменты, обозначающие доли секунды, или, наоборот, более крупная градация времени, измените масштаб временной шкалы по своему усмотрению. В качестве примера на рис. 3.14 показано изменение масштаба временной шкалы до 750%, чтобы разделить анимацию на доли секунды.

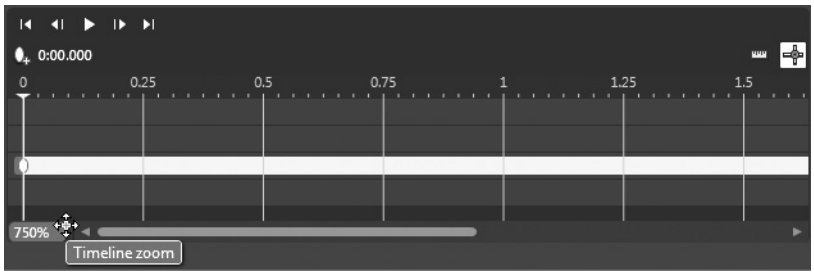


Рис. 3.14. Увеличение масштаба временной шкалы до более мелких градаций времени анимации

Примечание. Если вы изменили масштаб временной шкалы, вновь сделайте его равным 100%, чтобы вернуться к исходному виду этой шкалы с градацией по секундам.

Взаимодействие с раскадровками в коде

Когда вы создаете анимацию в Expression Blend и делаете это в проекте приложения Silverlight, то при запуске этого приложения на выполнение никакой логики, начинающей анимацию по раскадровке, автоматически не добавляется. С другой стороны, при запуске на выполнение приложения WPF анимация автоматически начинается по раскадровке, когда главное окно приложения загружается в оперативную память, хотя это положение можно впоследствии изменить. О том, как изменяется этот устанавливаемый по умолчанию режим работы приложения WPF, речь пойдет далее, а до тех пор покажем на рассматриваемом здесь примере проекта, как можно начинать анимацию посредством кода. Что же касается управления раскадровками в разметке, то мы вернемся к этому вопросу, когда будем рассматривать объект поведения типа `ControlStoryboardAction`.

Прежде всего необходимо решить, какое именно действие пользователя должно привести к началу анимации. Этим действием может стать щелчок на кнопке, выбор пункта меню, щелчок на самом круге, нажатие клавиши или любой другой, специально указываемый вид входной информации от пользователя. В целях рассматриваемого здесь примера допустим, что анимация должна начинаться в том случае, если пользователь щелкнет на элементе управления типа `Button`. Итак, найдите объект типа `Button` на панели Tools, но не забывайте, что искать нужные вам элементы управления можно также в библиотеке ресурсов и на панели Assets.

Примечание. Текст надписи на выбранном объекте типа `Button` можно изменить, соответственно настроив его свойство `Content` (Содержимое) в области Common Properties на панели Properties.

Добавив объект типа `Button` на монтажный стол, переименуйте новый элемент управления пользовательского интерфейса на `btnStartAnimation` (кнопка начала анима-

ции), воспользовавшись панелью Properties. Далее щелкните на кнопке Events, находящейся на этой же панели, и найдите событие Click, связанное с выбранным объектом типа Button[4]. Дважды щелкните на текстовом поле справа от имени события Click. Теперь в исходном коде текущего проекта вы обнаружите приведенный ниже пустой обработчик событий.

```
private void btnStartAnimation_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
  // Что сделать: добавить здесь реализацию обработчика событий в коде.
}
```

Напомним, что в среде Expression Blend IDE элементы раскадровки автоматически сохраняются в качестве ресурса объекта, размещаемого в словаре ресурсов соответствующего объекта типа Window на платформе WPF или же объекта типа UserControl на платформе Silverlight. С учетом этого обстоятельства ваша первая задача по взаимодействию с раскадровками в коде состоит в том, чтобы найти объект раскадровки в коллекции ресурсов по ключевому имени, присвоенному раскадровке при ее создании (это имя AnimateCircle, если вы следовали рассматриваемому здесь примеру). Найдя этот объект, вызовите метод Begin(). Ниже приведен полный код обработчика событий Click[5].

```
private void btnStartAnimation_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
  Storyboard animCircle;
  animCircle = (Storyboard)this.Resources["AnimateCircle"];
  animCircle.Begin();
}
```

Запустите свое приложение на выполнение, щелкните на элементе управления типа Button и посмотрите, как воспроизводится анимация!

Подробнее о классе Storyboard

Вы, вероятно, уже догадались, что в классе Storyboard определено намного больше функциональных возможностей, чем те, которыми обладает метод Begin(). В частности, в этом классе предоставляются методы Pause() и Stop() для частичной и полной остановки анимации соответственно, а также целый ряд свойств, с помощью которых определяются те же самые режимы повторения и автоматического изменения на обратное направление воспроизведения анимации, что и рассматривавшиеся ранее на панели Properties. Если вас интересует полное описание класса Storyboard, обращайтесь за справкой к документации на .NET Framework 4.0 SDK или Silverlight SDK.

Примечание. В исходный код рассматриваемого здесь примера проекта включено создание ряда других объектов кнопок для дополнительного управления анимацией по раскадровке. Надеюсь, этот код не окажется для вас слишком сложным.

На этом краткое введение в способы и средства создания анимации в среде Expression Blend IDE завершается. Нам еще предстоит рассмотреть целый ряд интересных вопро-

---

[4] В главе 2 демонстрировался процесс обработки на панели Properties событий, наступающих для заданного объекта. (Помните о кнопке с изображением молнии на пиктограмме?)

[5] В том файле исходного кода, где используется класс Storyboard, необходимо импортировать пространство имен System.Windows.Media.Animation, что, как правило, делается автоматически в среде Expression Blend IDE.

сов, поэтому если вы вполне освоились с представленными выше основными инструментами для создания анимации в Expression Blend, смело переходите к чтению следующей части этой главы.

Исходный код. Исходный код примера проекта `SimpleBlendAnimations` находится в папке `Ch 3 Code` загружаемого архива примеров проектов к данной книге.

Способы анимации, характерные для платформы WPF

В прикладном интерфейсе Windows Presentation Foundation API поддерживается ряд полезных способов анимации, которые, к сожалению, не находят поддержки в текущей версии платформы Silverlight. В первом из этих способов применяются *траектории движения*, с помощью которых можно легко перемещать элементы пользовательского интерфейса по объекту типа `Path`, нарисованному инструментом Pen или Pencil. И хотя аналогичного результата можно, конечно, добиться и в приложении Silverlight, для этого потребуется значительно больше усилий.

Второй характерный для платформы WPF способ анимации состоит в применении *триггеров* для взаимодействия с раскадровкой. На платформе WPF триггер обозначает применявшийся некогда способ реагирования на условие наступление события в разметке XAML. Например, с помощью триггеров можно написать такую разметку, в которой отслеживаются различные события от мыши (наведение курсора мыши на объект, нажатие и отпускание кнопки мыши и т.д.), события логического фокуса, события от клавиатуры и пр. Кроме того, можно написать дополнительную разметку, в которой отдельная раскадровка будет начинаться при наступлении подобных событий. Вам, возможно, известно, что среда триггеров на платформе WPF первоначально предназначалась для того, чтобы разработчики могли внедрять визуальные подсказки в специальные шаблоны с целью переопределить пользовательский интерфейс в отношении элементов управления и форм.

Несмотря на весьма ограниченную поддержку триггеров на платформе Silverlight, по тому же самому принципу действует диспетчер визуальных состояний (VSM). В действительности диспетчер VSM был настолько хорошо принят в программистских кругах, что на платформе WPF был внедрен свой вариант VSM в ее прикладной интерфейс API, начиная с версии .NET 4.0. С учетом этого обстоятельства у разработчиков приложений на платформе WPF теперь имеются два варианта выбора, когда дело доходит до внедрения визуальных подсказок в специальные шаблоны: триггеры или диспетчер VSM. Подробнее о том, как пользоваться триггерами и диспетчером VSM для построения специальных шаблонов, речь пойдет в главе 5, а до тех пор уделим основное внимание только триггерам для управления независимыми объектами раскадровки. Но сначала рассмотрим назначение траекторий движения, применяемых для анимации на платформе WPF.

Примечание. Как будет показано в конце главы, объект поведения типа `ControlStoryboardAction` дает также возможность манипулировать анимацией в разметке, используя средства WPF или Silverlight.

Работа с траекториями движения на платформе WPF

В Expression Blend предоставляется очень удобная возможность автоматически формировать на платформе WPF раскадровку, в которой выбранный объект должен перемещаться по предварительно заданной траектории движения. В частности, траектории

движения можно использовать для перемещения специальных графических элементов по вершинам линейного графика, показывая прибыли компании, что, конечно, приятнее, чем убытки, а возможно, и для перемещения элемента управления типа `Image`, содержащего изображение автомобиля, движущегося по пути, рассчитанному в прикладной программе глобальной системы местоопределения (GPS).

Траектория движения создается в среде Expression Blend довольно просто. В качестве примера создайте новый проект приложения WPF, присвоив ему имя `WPFMotionPathApp`. Как и при построении специального объекта раскадровки, ваша первая задача в данном случае — определить те объекты, которые должны перемещаться по данной траектории движения. А поскольку для этой цели может быть использован любой объект, создайте несколько разных элементов пользовательского интерфейса по своему усмотрению, например `Ellipse`, `Button` и `Rectangle`. Затем присвойте каждому из этих объектов подходящее имя в поле Name на панели Properties, например `circleOne`, `btnClickMe` (кнопка "Щелкни на мне") и `myRect`.

После определения ряда объектов в качестве целевых для анимации по траектории ваша следующая задача — создать траектории движения. Для этой цели можете выбрать соответствующие элементы управления из категории Shapes в библиотеке ресурсов или же нарисовать контуры траекторий произвольной геометрической формы инструментами Pen и Pencil. Итак, создайте три разные геометрические формы на монтажном столе (их местоположение, размеры и конфигурация особого значения не имеют). Но ради поддержания порядка на монтажном столе присвойте каждой созданной форме будущей траектории движения подходящее имя, например `myStar`, `myPolygon` и `myArc`.

Кроме того, для каждой геометрической формы будущей траектории движения (в данном случае звезды, многоугольника и дуги) рекомендуется настроить свойство `Fill` на режим раскраски No brush в редакторе кистей, а также установить значение в пределах 2-3 в свойстве `StrokeThickness`, доступном в области Appearance на панели Properties, чтобы траектории движения было лучше видно. В качестве примера на рис. 3.15 приведено главное окно (объект типа `Window`) на данном этапе разработки проекта на платформе WPF.

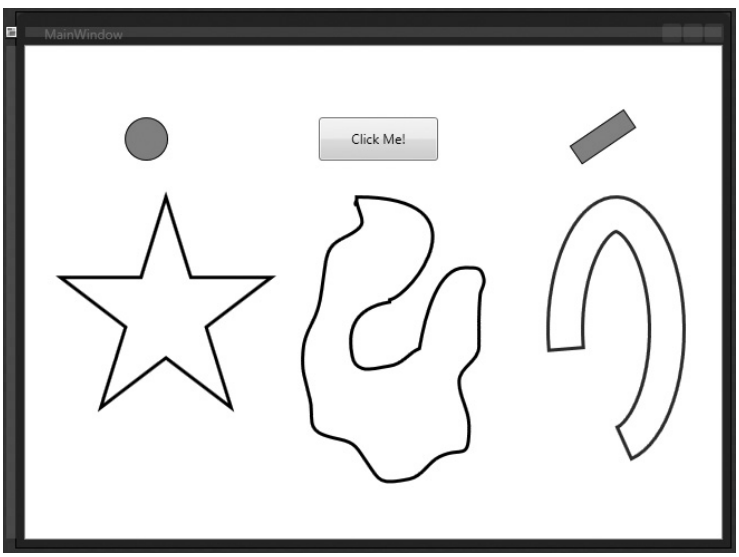


Рис. 3.15. Три элемента пользовательского интерфейса и три контура, определяющие анимацию по траектории

Выберите на панели Objects and Timeline контур первой формы, чтобы преобразовать его в траекторию движения (в данном случае это контур в форме звезды). Щелкните правой кнопкой мыши на узле этого объекта в иерархическом представлении и выберите команду Path⇨Convert to Motion Path (Контур⇨Преобразовать в траекторию движения) из всплывающего контекстного меню, как показано на рис. 3.16.

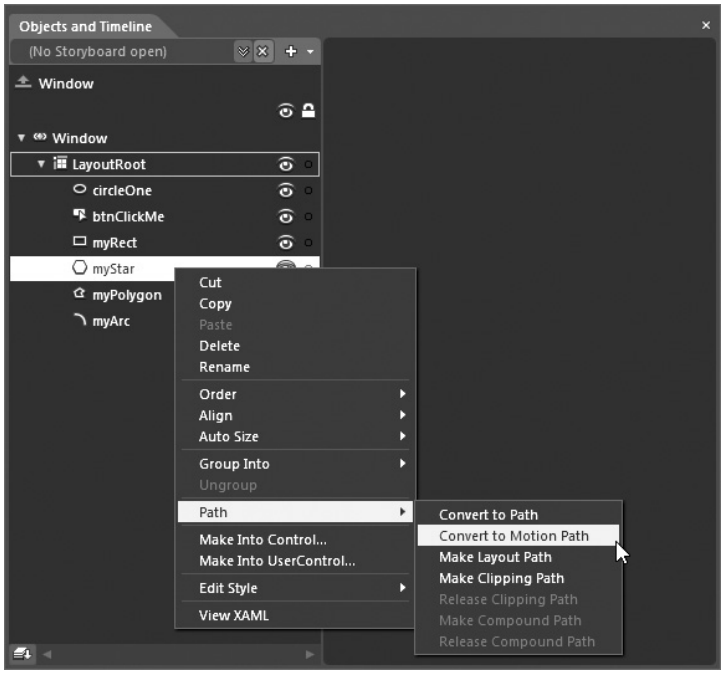


Рис. 3.16. Преобразование контура в траекторию движения

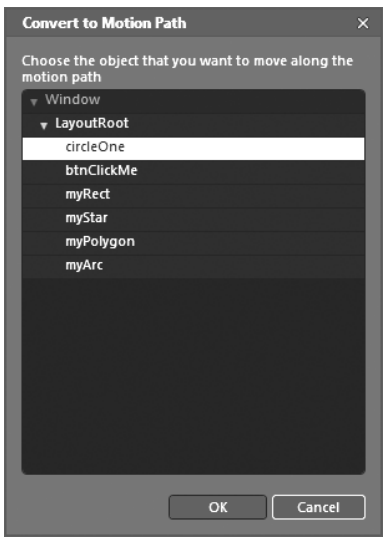


Рис. 3.17. Выбор объекта для перемещения по траектории движения

После выбора этой команды появится диалоговое окно, в котором можете далее выбрать один или несколько объектов для перемещения по траектории движения, преобразуемой из контура заданной формы. В данном случае выберите круг в качестве одного из целевых объектов, как показано на рис. 3.17.

Как только вы выберете целевой объект, будет создан новый объект раскадровки `Storyboard1` (он появится в верхней части иерархического представления на панели Objects and Timeline). Переименуйте этот объект описанным ранее способом в `MoveShapes` (перемещение форм). Обратите внимание на то, что после создания раскадровки вы сможете воспользоваться любым из рассмотренных ранее в этой главе способов для настройки свойств этой раскадровки (режима автоматического изменения на обратное направление воспроизведения анимации, продолжительности анимации и т.д.). Кроме того, можете изменить начальный и конечный моменты каждого промежутка времени между ключевыми кадрами.

Настройте анимационную последовательность таким образом, чтобы она начиналась на нульсекундной отметке и оканчивалась на пятисекундной. Таким образом, вращательное движение круга по траектории в форме звезды будет продолжаться около 5 секунд.

Повторите описанную выше общую процедуру для перемещения второго и третьего целевых объектов (в данном случае это объекты типа `Button` и `Rectangle`) по остальным траекториям движения. Измените по своему усмотрению начальный и конечный моменты анимации перемещения каждого из этих объектов по траектории движения. В качестве примера на рис. 3.18 показано, что общая продолжительность всей анимации составляет восемь секунд, хотя движение каждого целевого объекта (круга, кнопки и прямоугольника) начинается и оканчивается в разные моменты времени.

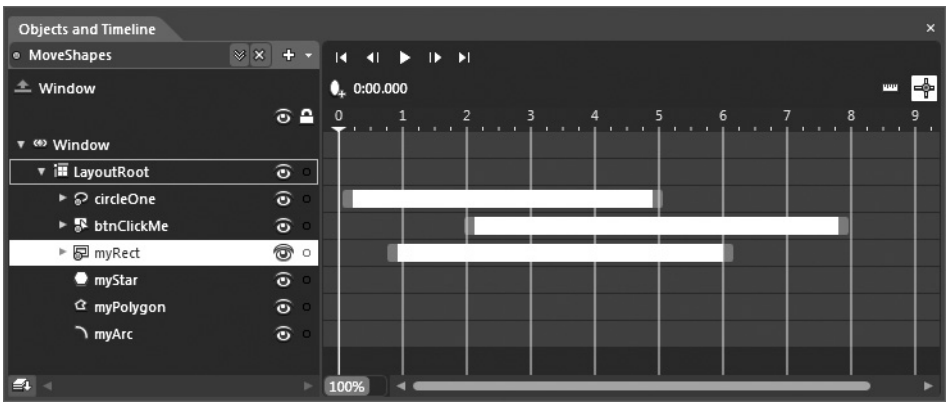


Рис. 3.18. Анимация движения каждого целевого объекта по заданной траектории

Для того чтобы сделать рассматриваемый здесь пример более интересным, организуйте на панели Properties обработку события `Click`, связанного с объектом типа `Button`. Так, если щелкнуть на этом объекте, появится информативное окно сообщения `You clicked me!` (Вы щелкнули на мне!), как показано в приведенном ниже коде обработчика событий.

```
private void btnClickMe_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
   MessageBox.Show("You clicked me!");
}
```

Запустите приложение на выполнение, нажав функциональную клавишу <F5> или комбинацию клавиш <Ctrl+F5>. В итоге все три объекта автоматически последуют по заданным траекториям движения. (Проверьте свою реакцию: успеете ли вы щелкнуть на кнопке, пока она движется.) В отличие от рассмотренного ранее примера проекта на платформе Silverlight, в данном случае анимация начинается, как только главное окно приложения (объект типа `Window`) загрузится в оперативную память. Такое поведение объясняется тем, что по умолчанию анимация по любой раскадровке, создаваемой в области действия приложения WPF, автоматически начинается после загрузки главного окна благодаря приведенной ниже разметке XAML из файла `MainWindow.xaml`.

```
<Window.Triggers>
  <EventTrigger RoutedEvent="FrameworkElement.Loaded">
    <BeginStoryboard x:Name="MoveShapes_BeginStoryboard"
                  Storyboard="{StaticResource MoveShapes}"/>
  </EventTrigger>
</Window.Triggers>
```

Найдите эту разметку, используя редактор XAML, доступный на монтажном столе. Как видите, в этой разметке описывается применяемый на платформе WPF триггер, с помощью которого, как известно, перехватывается условие наступления события, что избавляет от необходимости писать для этой цели специальный код на C# или VB. В данном случае триггер проверяет момент наступления события `Loaded`, связанного с загрузкой объекта типа `Window` в оперативную память[6]. Когда это событие наступает, обрабатывается элемент разметки `<BeginStoryboard>` и начинается воспроизведение анимации по раскадровке, представленной объектом `MoveShapes`.

Несмотря на то что данный триггер анимации на платформе WPF оказывается очень удобным в тех случаях, когда требуется быстро проверить, каким образом анимация будет вести себя во время выполнения разрабатываемого приложения, следует ясно отдавать себе отчет в том, что не всякая анимация должна начинаться после загрузки основного окна приложения. Вполне возможно, что вам потребуется начинать анимацию только в том случае, если пользователь выберет конкретный пункт меню, щелкнет на отдельной кнопке или выполнит иное предусмотренное действие. Разумеется, события можно обрабатывать и вручную, начиная анимацию по раскадровке непосредственно в коде, а среда триггеров на платформе WPF предоставляет лишь альтернативный вариант запуска анимации без написания специального кода. Рассмотрим далее инструменты, доступные в среде Expression Blend IDE для применения триггеров в проектах на платформе WPF.

Исходный код. Исходный код примера проекта `WPFMotionPathApp` находится в папке `Ch 3 Code` загружаемого архива примеров проектов к данной книге.

Управление анимацией с помощью триггеров на платформе WPF

Назначение триггеров на платформе WPF было ранее определено как способ реагирования в разметке на условия наступления событий, а в предыдущем примере проекта было продемонстрировано действие простого триггера на практике. Для более подробного ознакомления со средой триггеров на платформе WPF создайте новый проект приложения WPF, присвоив ему имя `WPFAnimationTriggerApp`.

В данном примере проекта вам предстоит создать новую раскадровку анимации, в ходе которой объект типа `Button` совершает полный оборот. Сначала добавьте новый объект типа `Button` на монтажном столе, присвойте ему имя `btnMyButton` и установите подходящее значение в его свойстве `Content`. Затем создайте новую раскадровку, присвоив ей имя `SpinButtonAnimation` (анимация вращения кнопки); напомним, что для этого достаточно щелкнуть на кнопке со знаком + на панели Objects and Timeline. И наконец, определите односекундный промежуток времени на временной шкале, введя два ключевых кадра анимации выбранного элемента управления начиная с нульсекундной отметки, как показано на рис. 3.19.

А теперь выберите объект типа `Button` на монтажном столе и поверните его на 360°, воспользовавшись мышью. Напомним, что графические преобразования объектов можно выполнять непосредственно на монтажном столе. В данном случае поместите курсор мыши с внешней стороны одного из углов кнопки, нажмите кнопку мыши и, не отпуская ее, переместите курсор в нужном направлении вращения кнопки, как показано на рис. 3.20.

---

[6] Класс `Window` расширяет свой родительский класс `FrameworkElement`.

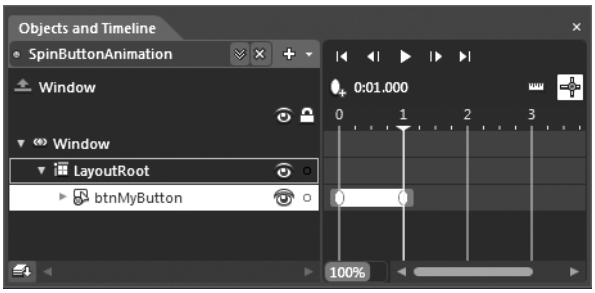


Рис. 3.19. Начальная раскадровка

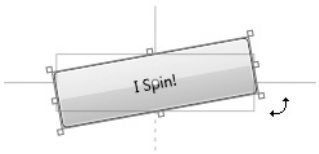


Рис. 3.20. Вращение кнопки, представленной объектом типа `Button`

Если вы теперь запустите приложение на выполнение, элемент управления типа `Button` должен совершить полный оборот при запуске приложения благодаря автоматически формируемому триггеру. Прежде чем исследовать причины такого первоначального поведения, просмотрите всю разметку, автоматически сформированную до сих пор в коде XAML. Эта разметка, немного отредактированная и аннотированная, приведена ниже.

```
<Window ... >
  <!-- Помните! Объекты раскадровки формируются как ресурсы объектов -->
  <Window.Resources>
  <Storyboard x:Key="SpinButtonAnimation">
    ...
  </Storyboard>
  </Window.Resources>

  <!-- Этот триггер запускает анимацию по раскадровке SpinButtonAnimation,
      когда объект типа Window загружается в оперативную память -->
  <Window.Triggers>
    <EventTrigger RoutedEvent="FrameworkElement.Loaded">
      <BeginStoryboard x:Name="SpinButtonAnimation_BeginStoryboard"
        Storyboard="{StaticResource SpinButtonAnimation}"/>
    </EventTrigger>
  </Window.Triggers>

  <Grid x:Name="LayoutRoot">
    <!-- Эта кнопка служит в качестве целевого объекта для анимации -->
    <Button x:Name="btnMyButton" ... >
      ...
    <Button/>
  </Grid>
</Window>
```

Ввод триггера на панели Triggers

Вернитесь сначала к виду рабочего пространства конструирования (Design), нажав функциональную клавишу <F6>, если вы еще не сделали этого, а затем найдите панель Triggers (Триггеры). В рабочем окне Expression Blend IDE она находится рядом с панелью Assets, но ее, как, впрочем, и любую другую панель в этой среде, можно показывать и скрывать по соответствующей команде, выбираемой из меню Window. Как показано на рис. 3.21, на панели Triggers доступен один и только один триггер для текущего объекта типа `Window`, а именно: триггер, запускающий анимацию по раскадровке `SpinButtonAnimation` после загрузки главного окна приложения в оперативную память.

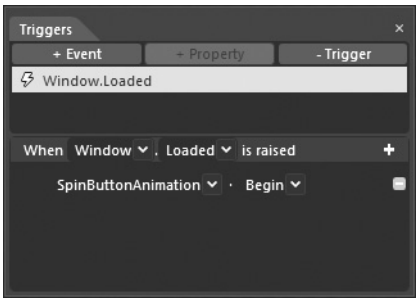


Рис. 3.21. На панели Triggers можно организовать запуск анимации по раскадровке при соблюдении условий наступления определенных событий

В верхней области панели Triggers находятся три выбираемые кнопки, с помощью которых можно взаимодействовать с перечисленными ниже типами триггеров на платформе WPF.

- **Триггер события.** Служит для взаимодействия с временной шкалой анимации при наступлении определенного события, например, `Click` при выполнении щелчка кнопкой мыши на выбранном элементе пользовательского интерфейса. Триггеры событий могут быть установлены для любого объекта, находящегося на монтажном столе.
- **Триггер свойства.** Оказывается полезным при создании специального стиля оформления или шаблона (подробнее об этом — в главе 5). По существу, такой механизм позволяет изменять значение одного свойства в зависимости от того, насколько изменяется значение другого свойства.

С помощью кнопки +Event можно вводить новый триггер события, с помощью кнопки +Property — новый триггер свойства, запускающий анимацию по раскадровке при изменении свойства выбранного объекта, а с помощью кнопки –Trigger — удалять выбранный в настоящий момент триггер из редактора или разметки. В этой главе не рассматривается назначение триггеров свойств, поэтому не будем пока что обращать внимания на кнопку +Property, а более подробно триггеры рассматриваются в главе 5.

Глядя на настройку текущего триггера, приведенную на рис. 3.21, можно заметить, что этот триггер отслеживает событие `Window.Loaded`, обозначенное знаком молнии слева от его имени. Ниже списка текущих триггеров находится ряд раскрывающихся списков, с помощью которых составляется простое и удобочитаемое предложение, описывающее то, что происходит при срабатывании триггера. Ниже приведен общий шаблон такого предложения.

```
When objectName.eventOnObject is raised storyboardName.storyboardAction
```

В настоящий момент это предложение гласит: “При наступлении события **`objectName.eventOnObject`** запускается анимация по раскадровке `storyboardName.storyboardAction`”. Итак, удалите текущий триггер полностью, выделив элемент списка `Window.Loaded` и щелкнув на кнопке –Trigger.

А теперь введите новый триггер, который начнет анимацию по раскадровке `SpinButtonAnimation`, если щелкнуть на самой кнопке. Для этого щелкните сначала на кнопке +Event, а затем выберите объект типа `Button` по его имени из раскрывающего списка When, где в настоящий момент выбран объект типа `Window`. После этого выберите событие `Click` из раскрывающегося списка is raised, как показано на рис. 3.22.

Примечание. Если не можете найти объект для настройки триггера события на панели Triggers, убедитесь в том, что этот объект выбран в соответствующем узле иерархического представления на панели Objects and Timeline panel! По умолчанию на панели Triggers отображаются варианты выбора только для самого верхнего в иерархии объекта типа `Window` и выбранного в настоящий момент элемента пользовательского интерфейса.

Выбрав объект и условие наступления события, щелкните на кнопке со знаком + — самой последней в определении триггера. В итоге появится дополнительная часть пользовательского интерфейса панели Triggers, где вы можете выбрать любую раскадровку из текущего документа. В данном случае выберите сначала раскадровку `SpinButtonAnimation`, а затем команду, позволяющую начать анимационную последовательность. (Обратите внимание на то, что, помимо команды начать анимацию, можно также выбрать другие команды управления анимацией, как показано на рис. 3.23. Эти команды аналогичны тем, что применяются непосредственно в коде для управления анимацией.)

Если после этого вы запустите свое приложение на выполнение, нажав функциональную клавишу <F5> или комбинацию клавиш <Ctrl+F5>, то обнаружите, что кнопка начнет вращаться только после того, как вы щелкнете на ней. А если вы проанализируете разметку, автоматически сформированную в коде XAML, то заметите, что теперь триггер описывается в ней так, как показано ниже.

```
<Window.Triggers>
  <EventTrigger RoutedEvent="ButtonBase.Click"
                SourceName="btnMyButton">
    <BeginStoryboard
       Storyboard="{StaticResource SpinButtonAnimation}"/>
  </EventTrigger>
</Window.Triggers>
```

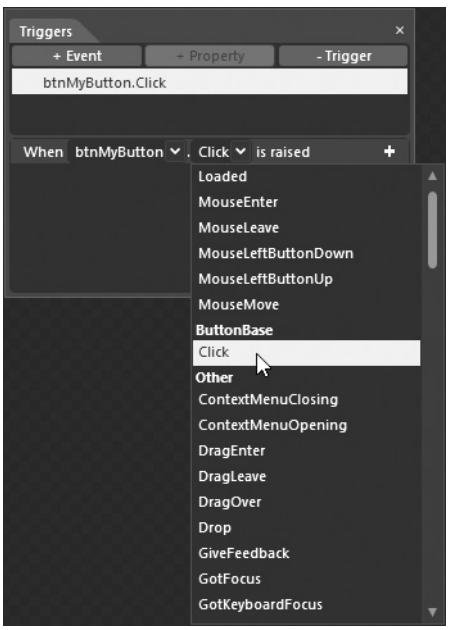


Рис. 3.22. Определение триггера события `Click` для объекта типа `Button`

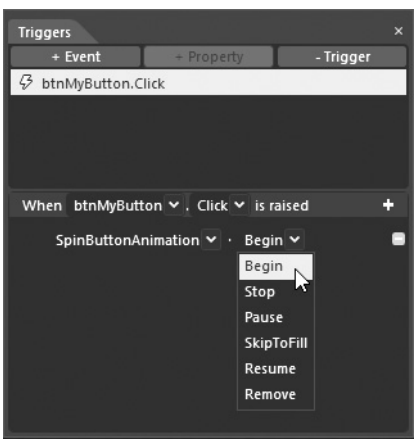


Рис. 3.23. Запуск анимации при наступлении определенного события

Построение системы меню в Expression Blend

В целях дальнейшей демонстрации возможностей триггеров событий нам потребуется расширить рассматриваемый здесь пример проекта, чтобы создать простую систему меню для управления анимационной последовательностью по раскадровке. Прежде всего удалите *полностью* текущий триггер события `Button.Click`. Затем выберите объект раскадровки `SpinButtonAnimation` на панели Objects and Timeline и установите значение `Forever` в свойстве `RepeatBehavior` этого объекта на панели Properties.

Далее воспользуйтесь панелью Assets, чтобы найти элемент управления типа `Menu` и дважды щелкните на нем, чтобы добавить его экземпляр на монтажном столе. Присвойте этому элементу управления имя `mainMenuSystem` (система главного меню) на панели Properties. Измените мышью размеры этого экземпляра объекта типа `Menu` таким образом, чтобы расположить его по всей ширине окна. Щелкните правой кнопкой мыши на данном элементе управления типа `Menu` непосредственно на монтажном столе и выберите команду Add MenuItem (Добавить пункт меню) из всплывающего контекстного меню, как показано на рис. 3.24.

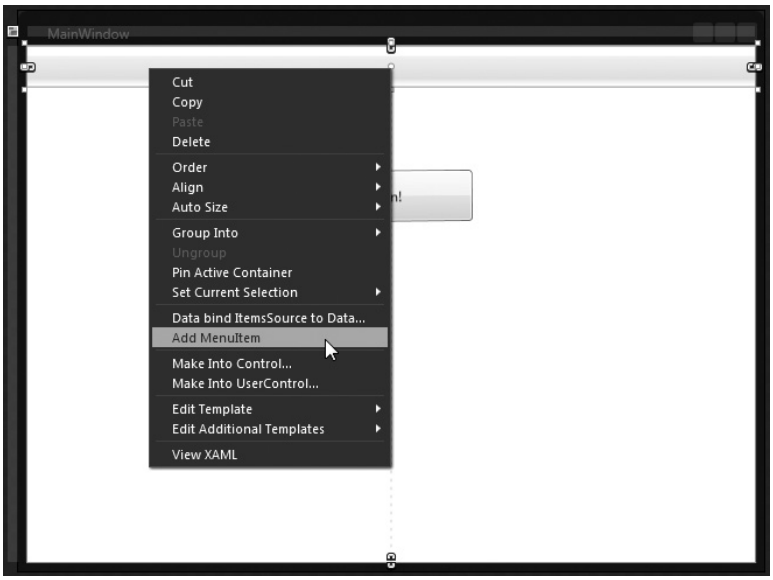


Рис. 3.24. Добавление пункта в меню

Выберите новый элемент управления типа `MenuItem`, найдите его свойство `Header` (Заголовок) на панели Properties и установите в нем значение `File` (свойства на этой панели обнаруживаются быстрее всего в режиме поиска, но в качестве подсказки заметим, что свойство `Header` находится в области Common Properties). Присвойте данному элементу управления имя `mnuFile` (меню файлов).

Примечание. Редактор меню в Expression Blend действует подобно аналогичному редактору в Visual Studio. В частности, щелкнув правой кнопкой мыши на каком угодно объекте типа `MenuItem` непосредственно на монтажном столе, вы сможете добавить к нему пункты подменю и разделительные полосы из всплывающего контекстного меню, как показано на рис. 3.24. А если вы выберете какой угодно объект типа `MenuItem` на панели Objects and Timeline, то сможете затем настроить любые его свойства и организовать обработку любого количества событий, связанных с этим объектом.

Еще раз выберите объект `mnuFile` на монтажном столе и щелкните на нем правой кнопкой мыши, чтобы добавить три пункта подменю. В целях рассматриваемого здесь примера присвойте свойствам `Header` этих элементов меню значения `Play!` (Воспроизвести!), **`Stop!`** (Остановить!) и **`Pause!`** (Приостановить!), а самим элементам — имена `mnuPlay`, `mnuStop` и `mnuPause` соответственно. По завершении построенная вами система меню должна выглядеть так, как показано на рис. 3.25.

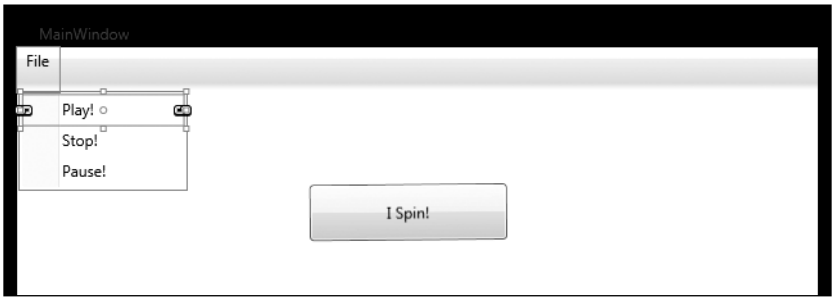


Рис. 3.25. Построение системы меню в Expression Blend

Обратите внимание на вложенный характер каждого компонента меню, отображаемого на панели Objects and Timeline (рис. 3.26). А теперь, когда у нас имеется элементарная система меню, мы можем приступить к созданию некоторых триггеров событий.

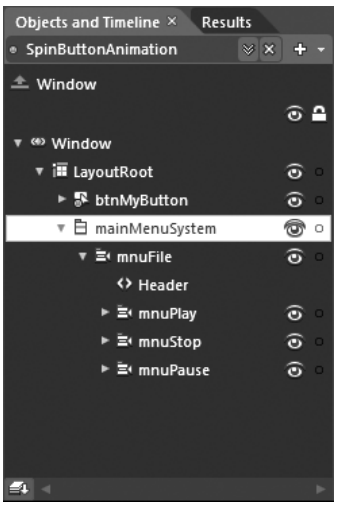


Рис. 3.26. Иерархическое представление системы вложенных меню на панели Objects and Timeline

Присваивание триггеров отдельным пунктам меню

Выберите объект `mnuPlay` на панели Objects and Timeline, перейдите к панели Triggers и щелкните на кнопке +Event. Воспользуйтесь редактором триггеров на панели Triggers, чтобы запускать анимацию по раскадровке `SpinButtonAnimation` при выборе пункта меню File⇨Play!. Аналогичным образом организуйте остановку и приостановку анимации при выборе пунктов меню File⇨Stop! и File⇨Pause! соответственно. На рис. 3.27 показан окончательно установленный ряд триггеров событий.

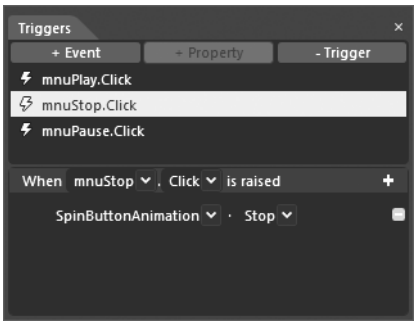


Рис. 3.27. Связывание триггеров событий с соответствующими пунктами меню

Вновь запустите приложение на выполнение и воспользуйтесь меню для управления анимацией по заданной раскадровке. Оцените по достоинству интерактивные возможности своего приложения, созданного без единой строки кода, написанной на C# или VB!

Примечание. Просмотрите ради интереса разметку, автоматически сформированную в коде XAML. Вы обнаружите, что в коллекцию `<Window.Triggers>` теперь входит несколько элементов `<EventTrigger>`, предназначенных для манипулирования различными вариантами управления анимацией по заданной раскадровке.

Для фиксации остальных условий наступления событий, связанных с объектом раскадровки, в меню можно, конечно, добавить дополнительные пункты, а еще лучше — дополнительные элементы пользовательского интерфейса и установить совершенно новые триггеры для других раскадровок. Эти возможности расширить рассматриваемый здесь пример проекта оставляются вам, читатель, в качестве упражнения по освоению на практике триггеров, целевых объектов анимации и ее раскадровки.

Исходный код. Исходный код примера проекта `WPFAnimationTriggerApp` находится в папке `Ch 3 Code` загружаемого архива примеров проектов к данной книге.

Представление об эффектах инерционности движения в анимации

Надеюсь, что, дойдя до этого раздела, вы вполне освоились с основными средствами и способами создания анимации в Expression Blend и теперь ясно представляете себе следующее.

- Общее назначение анимации на платформах WPF и Silverlight, которое состоит в изменении значений свойств объектов во времени.
- Назначение объекта раскадровки, которое состоит в связывании объектов, изменении их свойств и временных промежутков анимации.
- Использование панели Objects and Timeline для создания новой раскадровки и ряда ключевых кадров.
- Использование монтажного стола для изменения значений свойств объектов в режиме редактирования анимации на временной шкале.
- Управление анимацией по раскадровке в коде.
- Управление анимацией по раскадровке с помощью триггеров на платформе WPF.

Безусловно, ясное представление об упомянутых выше средствах и способах создания анимации означает заметный прогресс в освоении среды Expression Blend, но это совсем не означает, что вы можете теперь почивать на лаврах. Ведь имеется ряд дополнительных возможностей для создания анимации в Expression Blend, о которых следует непременно упомянуть. В частности, речь далее пойдет о применении редактора анимации для внедрения в нее *эффектов инерционности движения*. Проще говоря, подобные эффекты позволяют накладывать различные физические ограничения в промежутках времени между двумя последовательными ключевыми кадрами анимации. С помощью эффектов инерционности движения можно вносить в раскадровки такие физические явления, как отскакивание, притягивание, упругость объектов и пр.

Построение исходной компоновки

Как и прежде, начните с создания нового проекта, но на этот раз — приложения Silverlight, присвоив ему имя `AnimationEasingEffects`[7]. Как и в остальных примерах проектов, рассматривавшихся в этой главе, ваша первая задача — определить ряд объектов, а затем — ряд раскадровок для их анимации. Следуя рассматриваемому здесь примеру, нарисуйте три элемента пользовательского интерфейса на монтажном столе и присвойте им подходящие имена на панели Properties. Это могут быть любые графические элементы, но для данного примера выбраны формы круга, треугольника и многоугольника.

Следует иметь в виду, что на этот раз объекты должны располагаться у самого верхнего края монтажного стола, поскольку в результате применения к ним различных эффектов инерционности движения они будут отпущены для свободного падения вниз к нижнему краю экрана. Нарисуйте далее инструментом Pencil зону “приземления” объектов у нижнего края монтажного стола. На рис. 3.28 показан текущий вид компоновки пользовательского интерфейса в рассматриваемом здесь примере проекта приложения Silverlight.

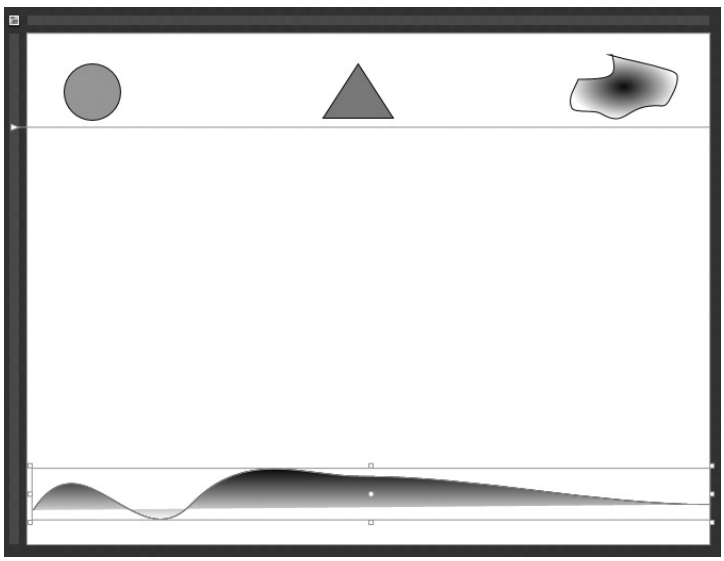


Рис. 3.28. Объекты, готовые к свободному падению

[7] В проектах WPF, разрабатываемых в среде Expression Blend IDE, также поддерживаются инструменты для организации инерционности движения в анимации, поэтому рассматриваемые далее способы в равной степени применимы к настольным приложениям, создаваемым на платформе Windows Presentation Foundation.

Создание исходных раскадровок

Следующая ваша задача состоит в создании отдельных раскадровок `DropAndBounceBall` (падающий и отскакивающий шар), `RubberbandTriangle` (треугольник, растягивающийся, как резиновая лента) и `HoverAndCrashPoly` (повисающий и разбивающийся многоугольник). В этой анимационной последовательности будет воспроизведено движение объекта типа `Ellipse` от исходной точки к зоне приземления. Итак, применяя способы, рассматривавшиеся в этой главе, настройте раскадровку `DropAndBounceBall` для фиксации этого движения в течение четырех секунд. Для этого просто переместите объект типа `Ellipse` в новое положение в режиме регистрации его движения.

Аналогичным образом создайте раскадровки `RubberbandTriangle` и `HoverAndCrashPoly`. Для этого переместите два оставшихся объекта от исходной точки к зоне приземления в течение четырех секунд (опять же в режиме регистрации их движения).

По завершении проверьте полученные в итоге анимационные последовательности, выбрав сначала раскадровку по ее имени на панели Objects and Timeline, а затем щелкнув на кнопке Play. Удовлетворившись первоначальными результатами анимации, перейдите к следующему разделу, где будет показано, каким образом применяются эффекты инерционности движения.

Применение эффектов инерционности движения в анимации

Выберите раскадровку `DropAndBounceBall` на панели Objects and Timeline и щелкните на кнопке с яйцевидной пиктограммой, обозначающей завершающий ключевой кадр анимации на временной шкале. Аналогично выбору объекта на монтажном столе, после щелчка непосредственно на нужном ключевом кадре в редакторе анимации на панели Properties появляется ряд настраиваемых свойств[8]. Как только вы щелкнете на ключевом кадре, на панели Properties появится область Easing (Инерционность движения), где по умолчанию выбирается кнопка EasingFunction (Функция инерционности движения). В настоящий момент ни один из эффектов инерционности движения в данном ключевом кадре не применяется, как показано на рис. 3.29.

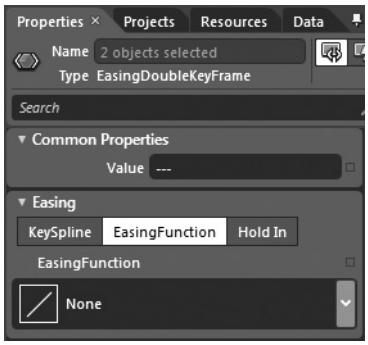


Рис. 3.29. Настройка ключевого кадра на эффекты инерционности движения

Каждый ключевой кадр может быть настроен на самые разные встроенные в Expression Blend эффекты инерционности движения, и все они выбираются из раскры-

[8] Имейте в виду, что отдельным ключевым кадрам можно присваивать имена, используя свойство `Name`. Это дает возможность управлять ключевыми кадрами и их свойствами непосредственно в коде.

вающегося списка EasingFunction. Как показано на рис. 3.30, все эффекты инерционности движения разделены на три категории.

- Варианты, перечисленные в столбце In, означают применение выбранного эффекта в начальной части ключевого кадра.
- Варианты, перечисленные в столбце Out, означают применение выбранного эффекта в завершающей части ключевого кадра.
- Варианты, перечисленные в столбце InOut, означают применение выбранного эффекта как в начальной, так и в завершающей части ключевого кадра.

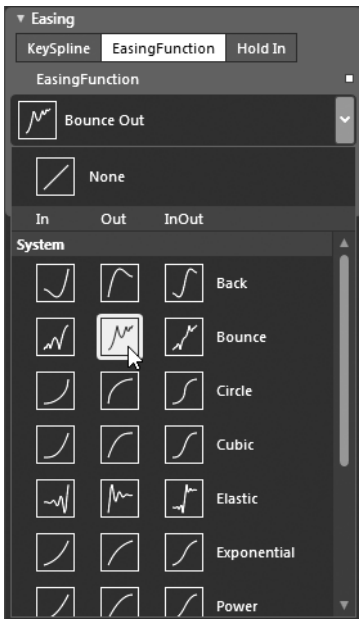


Рис. 3.30. Различные категории эффектов инерционности движения

Если вы раскроете список EasingFunction, то обнаружите в нем целый ряд встроенных в Expression Blend эффектов инерционности движения и их функций. Итак, выберите для данного ключевого кадра раскадровки `DropAndBounceBall` эффект Bounce (Отскакивание) в столбце Out (В конце), как показано на рис. 3.30.

Выбрав нужный эффект инерционности движения, можете далее настроить его на панели Properties. В качестве примера на рис. 3.31 показано, каким образом настраивается количество отскоков (свойство `Bounces`) и их интенсивность (свойство `Bounciness`).

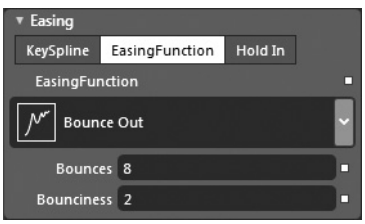


Рис. 3.31. Настройка свойств эффекта инерционности движения `Bounce Out`

Предварительно убедившись в том, что на панели Objects and Timeline выбрана раскадровка `DropAndBounceBall`, проверьте применение эффекта инерционности движения в анимации падающего и отскакивающего шара, щелкнув на кнопке Play. Продолжите экспериментирование с эффектом Bounce Out, а еще лучше — опробуйте в ознакомительных целях ряд других эффектов.

Выберите вторую раскадровку (в данном примере — `RubberbandTriangle`) и щелкните на завершающем ключевом кадре анимации. На этот раз попробуйте применить эффект инерционности движения `Elastic In` (Упругость в начале), у которого имеются интересные свойства `Oscillations` (Колебания) и `Springiness` (Пружинистость), как показано на рис. 3.32.

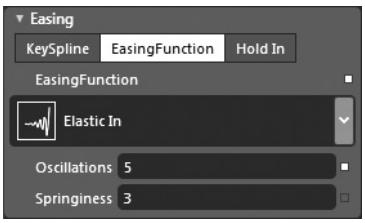


Рис. 3.32. Настройка свойств эффекта инерционности движения `Elastic In`

И на этот раз проверьте полученные результаты, щелкнув на кнопке Play панели Objects and Timeline.

Работа с редактором ключевых сплайнов

Преимущество встроенных в Expression Blend эффектов инерционности движения в анимации заключается в том, что они позволяют оперативно внедрять распространенные физические явления и свойства в анимационную последовательность. Если же вам требуется более точная настройка подобных эффектов, воспользуйтесь для этой цели редактором ключевых сплайнов (KeySpline). В качестве упражнения выберите объект завершенной раскадровки на панели Objects and Timeline и щелкните сначала на последнем ключевом кадре анимации, а затем на кнопке KeySpline. В итоге откроется новая область с редактором ключевых сплайнов, где можно откорректировать изменение скорости движения объекта по мере его приближения к данному ключевому кадру. Итак, откорректируйте значения координат `x1`, `x2`, `y1` и `y2`, переместив отдельные ползунковые элементы управления или щелкнув и перетащив желтую управляющую точку, но принимая во внимание следующее.

- Чем круче линия ключевого сплайна, тем быстрее изменяется значение в данной точке.
- Когда линия ключевого сплайна становится прямой, проходящей из левого нижнего угла графика к правому верхнему его углу, интерполяция получается линейной во времени.

На рис. 3.33 приведен пример коррекции формы ключевого сплайна, в результате которой последний целевой объект анимации в рассматриваемом здесь примере (т.е. многоугольник) сначала повисает на полпути к зоне приземления, а затем разбивается о нее.

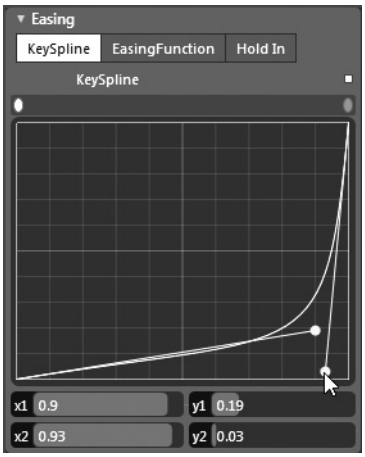


Рис. 3.33. Применение редактора ключевых сплайнов для точной настройки эффекта повисания и разбивания многоугольника

Воспроизведение анимации по раскадровке во время выполнения

Напомним, что при запуске приложения Silverlight, в отличие от приложения WPF, объекты раскадровки не настраиваются автоматически на начало анимации. В завершение рассматриваемого здесь примера проекта организуем обработку события `MouseLeftButtonDown`, наступающего при нажатии левой кнопки мыши для каждого объекта, который подлежит анимации. С этой целью выберите объекты трех геометрических форм по очереди на панели Objects and Timeline, щелкните сначала на кнопке Events, доступной на панели Properties, а затем дважды на текстовом поле справа от метки события `MouseLeftButtonDown`. После создания всех трех обработчиков событий внесите коррективы в их исходный код, как показано ниже на примере обработчика событий типа `MouseLeftButtonDown`, начинающего анимацию падающего и отскакивающего шара после щелчка левой кнопкой мыши на данном объекте.

```
private void ellipse_MouseLeftButtonDown(object sender,
    System.Windows.Input.MouseButtonEventArgs e)
{
  Storyboard sb = (Storyboard)this.Resources["DropAndBounceBall"];
  sb.Begin();
}
```

Если вы запустите теперь свое приложение Silverlight на выполнение, то, для того чтобы начать воспроизведение анимации, достаточно будет щелкнуть на любой из трех геометрических форм. Напомним, что совсем не обязательно ждать завершения одной анимации, чтобы начать другую. Можно воспроизвести одновременно все анимационные последовательности.

Исходный код. Исходный код примера проекта `AnimationEasingEffects` находится в папке `Ch 3 Code` загружаемого архива примеров проектов к данной книге.

Дальнейшее изучение эффектов инерционности движения в анимации

Надеюсь, вы согласитесь с тем, что описывать порядок настройки каждого эффекта инерционности движения в анимации нет никакого смысла. Ведь количество настроек каждого из подобных эффектов довольно велико, а проиллюстрировать конечный результат их применения в печатном издании все равно нельзя. Поэтому дальнейшее изучение различных эффектов инерционности движения в анимации вам придется продолжить самостоятельно, обратившись за справкой к разделу "Изменение интерполяции анимации в промежутках между ключевыми кадрами" (Change animation interpolation between keyframes) руководства пользователя Expression Blend (рис. 3.34).

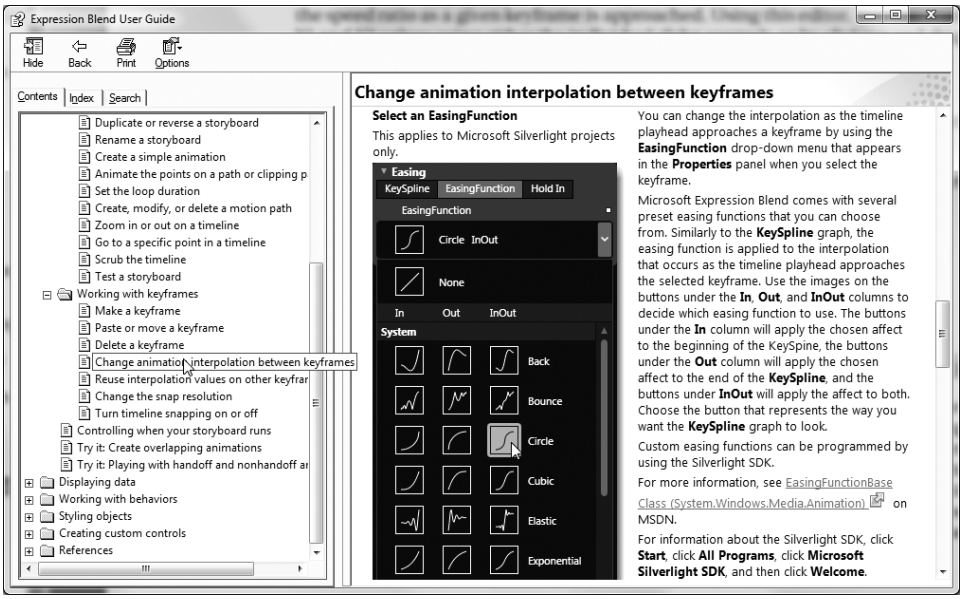


Рис. 3.34. За дополнительными сведениями о различных эффектах инерционности движения в анимации обращайтесь к руководству пользователя Expression Blend

Управление анимацией в разметке XAML с помощью объектов поведения

В завершение этой главы, посвященной средствам и способам создания анимации в Expression Blend, нельзя не упомянуть хотя бы вскользь о назначении *объектов поведения*, более подробно рассматриваемых в главе 4. По существу, объект поведения дает возможность применять сложную логику к элементу пользовательского интерфейса во время выполнения, вообще не прибегая к написанию процедурного кода.

Как будет показано в главе 4, существуют различные объекты поведения, с помощью которых можно перемещать мышью отдельные элементы пользовательского интерфейса, вызывать методы для этих объектов и выполнять прочие действия. Далее речь пойдет об объекте поведения типа `ControlStoryboardAction`, предоставляющем все необходимые средства для того, чтобы начинать, останавливать и приостанавливать анимацию в разметке XAML, причем делать это вне всякой связи со средой триггеров на платформе WPF.

Видоизмененный пример проекта SimpleBlendAnimations

Для демонстрации назначения и функциональных возможностей упомянутого выше объекта поведения еще раз откройте в среде Expression Blend IDE проект SimpleBlendAnimations, рассматривавшийся в качестве первого примера в этой главе, сохраните его полную копию по команде меню File⇨Save Copy of Project (Файл⇨Сохранить копию проекта) и присвойте новой копии имя SimpleBlendAnimations _ Behavior. Убедитесь в том, что новая копия данного проекта открыта теперь в среде Expression Blend IDE.

А теперь откройте в редакторе XAML файл разметки MainPage.xaml и найдите в разметке определение кнопки Start Animation! (Начать анимацию). В этом определении должно присутствовать значение обработчика событий Click, как показано ниже.

```
<Button x:Name="btnStartAnimation" Content="Start Animation!" ...
  Click="btnStartAnimation_Click"/>
```

Удалите значение обработчика событий Click полностью из определения упомянутой выше кнопки, чтобы разметка XAML выглядела в конечном итоге так, как показано ниже (конкретные свойства кнопки могут отличаться в зависимости от настройки данного объекта).

```
<Button x:Name="btnStartAnimation" Content="Start Animation!"
    HorizontalAlignment="Left" Margin="8,8,0,0"
    VerticalAlignment="Top" Width="124"/>
```

Откройте файл исходного кода C# и удалите полностью логику обработчика событий Click, где содержится код программного запуска анимации по раскадровке. Иными словами, удалите из исходного кода все следы метода btnStartAnimation _ Click(). Для проверки на отсутствие ошибок компиляции постройте и запустите рассматриваемый здесь проект на выполнение.

Добавление в проект объекта поведения типа ControlStoryboardAction

Перейдите к визуальному конструктору основной страницы, описываемой в разметке из файла MainPage.xaml, на монтажном столе, откройте панель Assets, выберите категорию Behaviors (Виды поведения) и найдите объект типа ControlStoryboardAction (рис. 3.35).

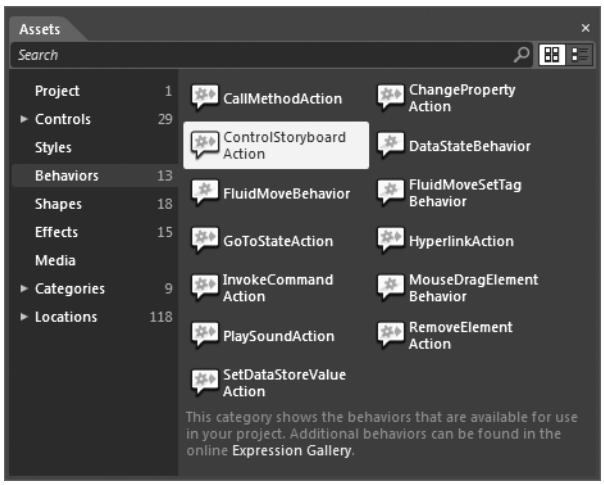


Рис. 3.35. Объект поведения типа ControlStoryboardAction

Перетащите этот объект поведения на объект типа `Button`, находящийся на монтажном столе (рис. 3.36). С другой стороны, можете перетащить объект поведения на соответствующий узел иерархического представления на панели Objects and Timeline.

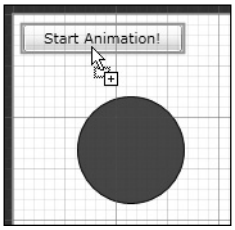


Рис. 3.36. Отдельные виды поведения можно перетаскивать в виде объектов на монтажный стол

Выберите вновь созданный узел на панели Objects and Timeline (в иерархическом представлении он должен появиться ниже узла `Button`), чтобы исследовать свойства, доступные в области Trigger на панели Properties, как показано на рис. 3.37. Обратите внимание на то, что в свойстве `SourceObject` (Исходный объект) устанавливается имя объекта типа `Button` вследствие предыдущей операции перетаскивания. В этом свойстве указывается тот объект на монтажном столе, с которым должен взаимодействовать объект поведения. Обратите также внимание на то, что в свойстве `EventName` (Имя события) автоматически устанавливается событие типа `Click`, хотя из раскрывающегося списка можно выбрать любое другое событие.

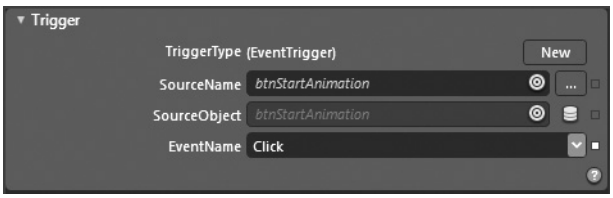


Рис. 3.37. Свойства `SourceName` и `EventName` служат для настройки порядка инициирования объектом конкретного поведения

Для того чтобы завершить настройку данного конкретного поведения, остается лишь выбрать сначала манипулируемый объект раскадровки, а затем указать действие, которое должно выполняться при срабатывании триггера соответствующего события. В качестве примера на рис. 3.38 показано, что если щелкнуть на кнопке, находящейся на главной странице приложения, начнется воспроизведение анимации круга по раскадровке, определяемой объектом `AnimateCircle`.

Запустите свое приложение на выполнение. Теперь у вас имеется возможность начинать анимацию без всякой помощи специального кода C# или обработчиков событий! Если вы проанализируете разметку, автоматически сформированную в коде XAML, то обнаружите в ней приведенный ниже фрагмент.

```
<Button x:Name="btnStartAnimation" Content="Start Animation!"
      HorizontalAlignment="Left" Margin="8,8,0,0"
      VerticalAlignment="Top" Width="124">
 <i:Interaction.Triggers>
   <i:EventTrigger EventName="Click">
   <ei:ControlStoryboardAction
```

```
      Storyboard="{StaticResource AnimateCircle}"/>
    </i:EventTrigger>
  </i:Interaction.Triggers>
</Button>
```

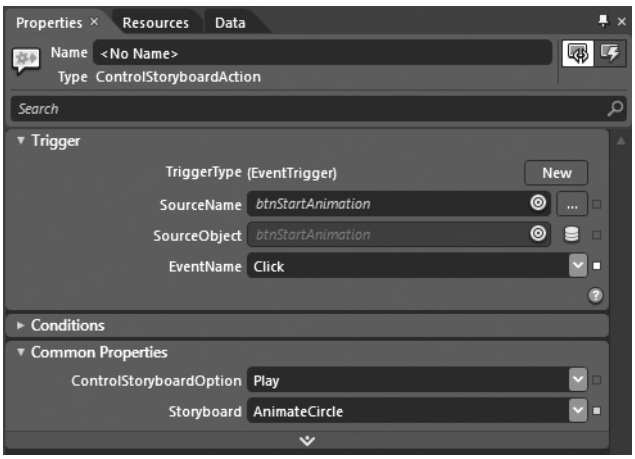


Рис. 3.38. Полностью настроенный объект поведения типа `ControlStoryboardAction`

По желанию можете добавить на монтажном столе ряд дополнительных объектов типа `Button` и определить для них новые объекты поведения типа `ControlStoryboardAction`, чтобы останавливать, приостанавливать и возобновлять циклическое воспроизведение анимации.

Исходный код. Исходный код примера проекта `impleBlendAnimations _ Behavior` находится в папке `Ch 3 Code` загружаемого архива примеров проектов к данной книге.

Резюме

В этой главе было показано, как пользоваться встроенным в Expression Blend редактором анимации, в котором можно создавать объекты раскадровки для приложений на платформах WPF и Silverlight. Напомним, что раскадровка состоит из ряда ключевых кадров анимации, в которых значения свойств объектов изменяются во времени. После рассмотрения различных элементов рабочего пространства анимации в этой главе было показано, каким образом организуется взаимодействие с раскадровкой в коде. В частности, получив нужный ресурс, можно далее воспользоваться средствами, предоставляемыми в классе `Storyboard`, для управления анимацией.

Далее в главе были рассмотрены некоторые средства и способы создания анимации на платформе WPF, начиная с назначения траекторий движения, по которым перемещаются целевые объекты, а затем перейдя к общему представлению о среде триггеров на платформе WPF. Как было показано на конкретных примерах, триггеры предоставляют возможность взаимодействовать с изменяющимися свойствами и наступающими событиями непосредственно в разметке для управления анимацией по раскадровке, а более подробно о триггерах речь пойдет в последующих главах.

И в заключение главы были вкратце представлены эффекты инерционности движения в анимации и объекты поведения. С помощью встроенных в Expression Blend эффектов инерционности движения можно очень просто (и довольно быстро) применять

физические явления и свойства в создаваемой анимации, в том числе отскакивание, притягивание, упругость и т.д. Для более точного управления подобными эффектами имеется также возможность настроить временные характеристики анимации по отдельным ключевым кадрам в интегрированном в Expression Blend редакторе ключевых сплайнов. А с помощью объектов поведения типа `ControlStoryboardAction` можно полностью манипулировать анимацией непосредственно в разметке XAML и вне всякой связи со средой триггеров на платформе WPF.

ГЛАВА 4

Элементы управления, виды компоновки и объекты поведения

Если вы не намерены создавать элементы пользовательского интерфейса только из геометрических форм, то в приложении на платформе WPF или Silverlight вам, вероятнее всего, понадобятся разнотипные элементы управления. Как и следовало ожидать, в состав прикладных интерфейсов API на платформах WPF и Silverlight входит целый ряд элементов управления, с помощью которых можно собирать данные, вводимые пользователем, а также реагировать на его действия. В этой главе поясняется, каким образом пользовательские интерфейсы компонуются в среде Expression Blend IDE из элементов управления, размещаемых в различных *диспетчерах компоновки*. Из нее вы узнаете, что на обеих платформах, WPF и Silverlight, для этой цели предоставляется отдельная коллекция диспетчеров компоновки, включая такие объекты, как, например, `Grid`, `StackPanel` и `Canvas`. И все они настраиваются на монтажном столе Expression Blend.

Научившись компоновать в среде Expression Blend IDE графические пользовательские интерфейсы на конкретных примерах проектов, можете далее ознакомиться с различными *объектами поведения*. Как пояснялось в главе 3, с помощью объектов поведения можно внедрять в приложение целый ряд дополнительных функциональных возможностей, не прибегая к написанию процедурного кода. В частности, имеются такие объекты поведения, которые позволяют перемещать мышью отдельные графические элементы, управлять анимацией по раскадровке, как было показано в предыдущей главе, а также изменять значения свойств или вызывать отдельные методы в зависимости от заданных условий и выполнять прочие действия.

Примечание. В этой главе представлены лишь самые основные элементы управления и объекты поведения, а другие их примеры будут рассматриваться в остальных главах.

Общее представление об элементах управления пользовательского интерфейса

Если вам приходилось заниматься созданием графических пользовательских интерфейсов в каком-нибудь из распространенных некогда графических конструкторов, например, в редакторе Java Swing, конструкторе веб-станиц или в интегрированной среде разработки Visual Studio (с помощью специально предназначенных для этих целей

инструментов), то вас не должно особенно удивлять то обстоятельство, что в среде Expression Blend IDE имеется возможность компоновать пользовательский интерфейс разрабатываемого проекта из таких элементов, как, например, кнопки, раскрывающиеся списки, комбинированные окна, меню, таблицы, индикаторы выполнения, представленные объектами типа `Button`, `ListBox`, `ComboBox`, `MenuItem`, `DataGrid` и `ProgressBar` соответственно.

На обеих платформах, WPF и Silverlight, предоставляется отдельная библиотека элементов управления, но еще большее их разнообразие доступно в прикладных интерфейсах API этих платформ. Так, целый ряд элементов управления поддерживается на платформе WPF в отдельном прикладном интерфейсе *Document API*. Как будет показано далее в главе, с помощью этого прикладного интерфейса в приложение WPF можно внедрять форматирование текста в стиле оформления документов в формате PDF по спецификации XML Paper Specification (XPS — спецификация бумажных XML-документов)[1].

Вообще говоря, в состав каждого прикладного интерфейса API входит отдельный набор элементов управления. Тем не менее каждый элемент управления может быть настроен сходным, но не одинаковым образом в среде Expression Blend IDE. Поэтому, независимо от того, на какой именно платформе (WPF или Silverlight) в качестве кнопки используется объект типа `Button`, на панели Properties в Expression Blend будут доступны одни и те же общие настраиваемые свойства, несмотря на то, что эти объекты происходят из разных классов, определенных в разных библиотеках.

Примечание. На панелях Tools и Assets в Expression Blend всегда отображаются компоненты, соответствующие выбранному типу проекта: WPF, Silverlight или Windows Phone 7, как поясняется в главе 7.

Обнаружение элементов управления в среде Expression Blend IDE

Когда элемент графического пользовательского интерфейса требуется добавить на монтажном столе, его можно найти на панели Tools. Но не следует забывать, что на панели Tools доступны для выбора лишь наиболее часто используемые элементы управления пользовательского интерфейса, сгруппированные по отдельным категориям. Уделив немного времени доступным для этих целей возможностям, вы обнаружите на панели Tools отдельные группы диспетчеров компоновки, элементов управления текстом (объекты типа `Label`, `TextBox` и т.д.), а также основных элементов управления вводом данных от пользователя (объекты типа `Button`, `Slider`, `ListBox` и т.д.), как показано на рис. 4.1.

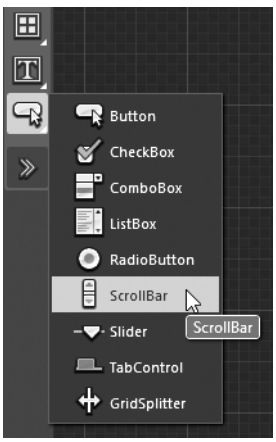


Рис. 4.1. На панели Tools предоставляются средства для быстрого доступа к наиболее часто применяемым элементам пользовательского интерфейса

Примечание. Напомним, что с графическими элементами можно взаимодействовать в диалоговом режиме работы панелей Expression Blend лишь в том случае, если монтажный стол находится в режиме конструирования или разделения, но не в режиме полноэкранного редактора XAML.

Для просмотра всего набора элементов управления пользовательского интерфейса следует открыть панель Assets или библиотеку ресурсов, доступную на панели Tools. Здесь можно

[1] XPS, по существу, является альтернативным вариантом формата файлов Adobe PDF, разработанным и предложенным корпорацией Microsoft.

обнаружить все имеющиеся элементы управления, перейдя к разделу All в категории Controls. В качестве примера на рис. 4.2 показаны элементы управления, доступные для проекта приложения WPF.

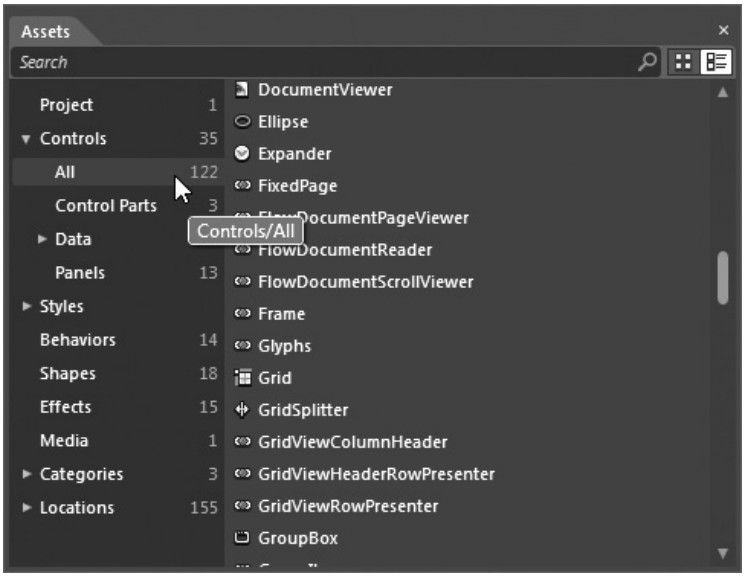


Рис. 4.2. В категории Controls библиотеки ресурсов для выбора доступны все элементы управления, предоставляемые в конкретном прикладном интерфейсе API

Настройка элементов управления на панели Properties

Независимо от способа обнаружения и выбора элементов пользовательского интерфейса можно нарисовать экземпляр нужного компонента на монтажном столе подобно рисованию геометрической формы (см. главу 2). После выделения этого компонента инструментом, активизируемым нажатием оперативной клавиши <V>, можно настроить выбранный графический элемент на панели Properties, организовать обработку связанных с ним событий (с помощью кнопки Events на той же панели Properties) или выполнить иные действия, необходимые для компоновки пользовательского интерфейса. А поскольку в предыдущих главах на конкретных примерах было показано, как работать с подобными компонентами панели Properties, мы не будем больше возвращаться к данному вопросу в этой главе, чтобы не повторяться. Вместо этих основ лучше рассмотрим более скрытые (и не всегда очевидные) особенности работы с компонентами панели Properties.

Дальнейшее изучение функциональных возможностей элементов управления

Учитывая характер данной книги, в этой главе не рассматриваются во всех подробностях функциональные возможности каждого элемента управления на платформе WPF или Silverlight и лишь в общих чертах поясняется, каким образом в среде Expression Blend IDE настраивается внешний вид элементов управления. Если вы занимаетесь программированием профессионально, то, безусловно, отдаете должное документации на .NET Framework 4.0 SDK и Silverlight SDK. Оба эти руководства пользователя

доступны для загрузки в Интернете и могут быть установлены на локальной машине с помощью диспетчера справочной библиотеки Visual Studio 2010 Help Library Manager[2].

Если вас интересуют подробные сведения об элементах управления на платформе WPF, но вы не установили справочную систему на локальной машине, обратитесь за справкой к разделу "Справка по элементам управления, свойствам и событиям" (Controls, properties, and events reference) руководства пользователя Expression Blend, где содержатся ссылки на доступную в Интернете документацию на платформы WPF и Silverlight. Так, если вы откроете руководство пользователя Expression Blend, то найдете интересующую вас тему в папке `References` (Справки), как показано на рис. 4.3.

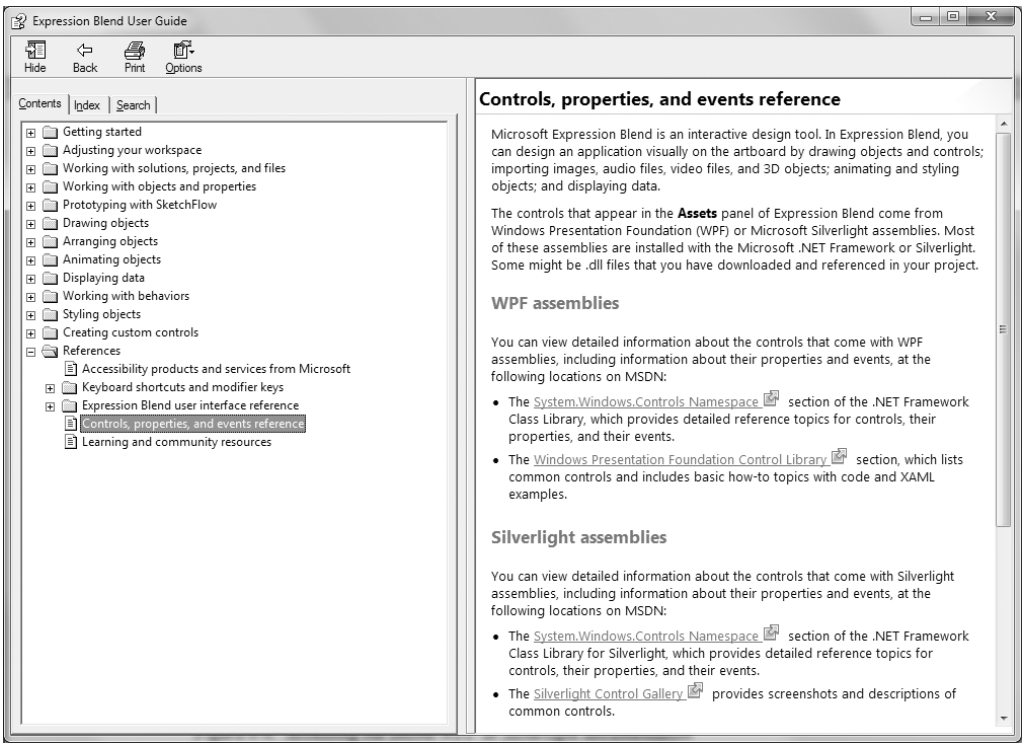


Рис. 4.3. В руководстве пользователя Expression Blend предоставляются ссылки на доступную в Интернете документацию на платформы WPF и Silverlight и соответствующие элементы управления

После этого щелкните на предоставляемых ссылках, чтобы перейти непосредственно к системе документации на интересующие вас элементы управления. В качестве примера на рис. 4.4 показана веб-страница, которая открывается после щелчка на ссылке `Silverlight System.Windows.Controls Namespace`.

[2] Запустите это инструментальное средство по команде Start⇨All Programs⇨Microsoft Visual Studio 2010⇨Visual Studio Tools⇨Manage Help Settings (Пуск⇨Все программы⇨Microsoft Visual Studio 2010⇨Инструменты Visual Studio⇨Управление параметрами справки) в Windows. После этого загрузите и установите документацию на целый ряд прикладных интерфейсов API, в том числе и для платформ Silverlight и Windows Phone 7.

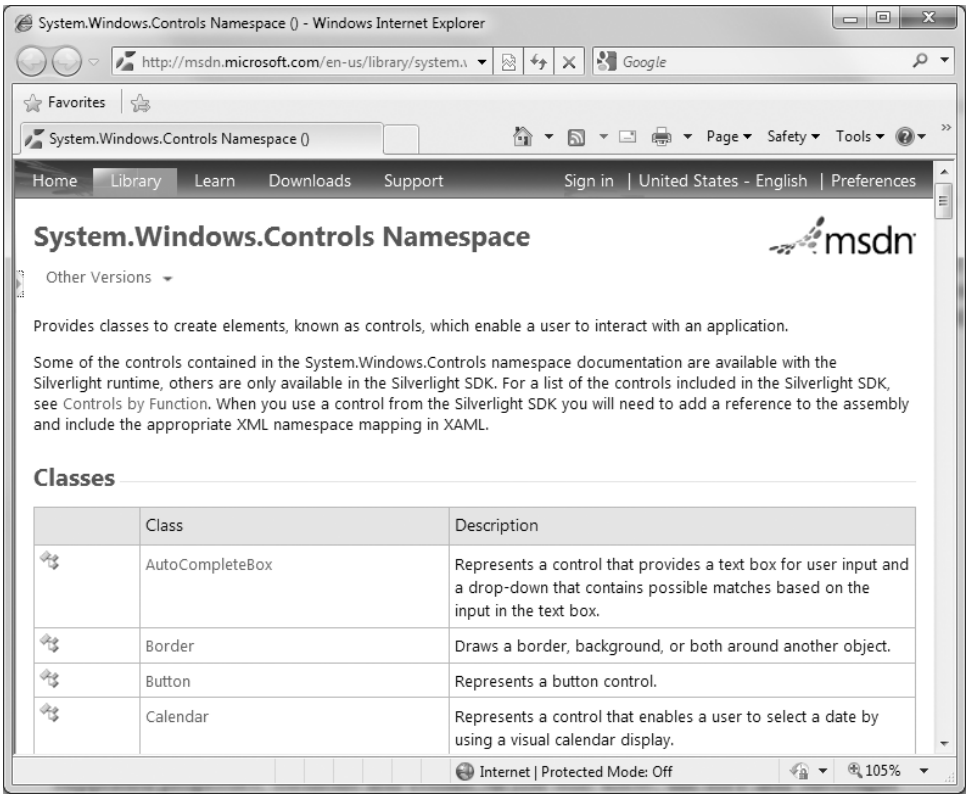


Рис. 4.4. Доступ в Интернете к документации на элементы управления, применяемые на платформе Silverlight

В этой и последующих главах подробно поясняются различные элементы управления, но за самой полной справкой по ним следует обращаться к соответствующей документации.

Представление о модели содержимого элементов управления

В первом примере проекта, рассматриваемом в этой главе, основное внимание уделяется такому заслуживающему особого внимания компоненту прикладных интерфейсов API на платформах WPF и Silverlight, как *модель содержимого элементов управления*. В этой модели под термином *содержимое* подразумевается то, что обычно служит для заполнения внутренней части элемента управления графического пользовательского интерфейса. В частности, содержимым для типичного элемента управления типа `Button` являются текстовые данные надписи на кнопке, например, OK, Отмена, Предъявить и т.п.

Для некоторых элементов управления графического пользовательского интерфейса может быть достаточно самого простого фрагмента текста, тем не менее на платформах WPF и Silverlight допускается определение намного более сложного содержимого[3].

[3] Элемент управления, в котором поддерживается модель содержимого элементов управления, должен быть производным от базового класса `ContentControl`.

В частности, можно определить элемент управления типа `Button` с оживляемой стрелкой и отображаемым фрагментом текста в качестве содержимого. Для определения сложного внутреннего содержимого в среде Expression Blend IDE предоставляются соответствующие инструменты, хотя их применение в подобных целях может оказаться не вполне очевидным. Для того чтобы разобраться в этом вопросе, начните, как всегда, с создания нового проекта приложения WPF, присвоив ему имя `BlendControlContent`[4].

Найдите на панели Tools или в библиотеке ресурсов элемент управления типа `Button` и расположите один его экземпляр у верхнего края монтажного стола визуального конструктора главного окна (объекта типа `Window`). Как только вы выберете этот элемент управления на монтажном столе, в области Common Properties на панели Properties появится текстовое поле свойства `Content`, где можно ввести простой текст, как показано на рис. 4.5.

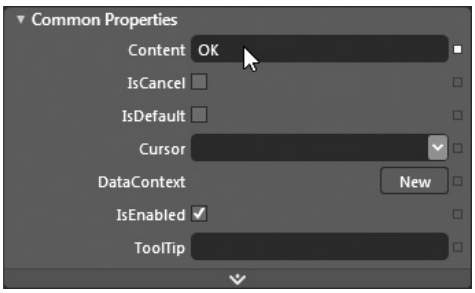


Рис. 4.5. Простое содержимое элемента управления можно задать на панели Properties

Очевидно, что такое простое поле ввода информации нельзя использовать для определения более сложного содержимого, в том числе встраиваемой графики, анимации и т.п. Когда же требуется добавить в элемент управления сложное, составное содержимое, следует прежде всего ввести *диспетчер компоновки* в выбранный элемент управления графического пользовательского интерфейса. Этот диспетчер компоновки будет, в свою очередь, содержать ряд отдельных элементов, представляющих содержимое в целом.

Создание составного содержимого

Подробнее о работе с диспетчерами компоновки речь пойдет ниже, а пока найдите элемент управления типа `StackPanel` в библиотеке ресурсов (напомним, что интересующий вас элемент можно быстро найти, введя его имя в поле поиска Search). Дважды щелкните на нем, чтобы добавить его в главное окно приложения, представленное объектом типа `Window`. Затем перетащите его на элемент управления типа `Button` и нажмите клавишу <Alt> (рис. 4.6). В итоге объект типа `StackPanel` окажется в области действия элемента управления `Button` и будет использоваться для хранения нового составного содержимого.

Примечание. При создании составного содержимого порой оказывается полезно увеличивать масштаб редактируемого элемента управления, используя доступные на монтажном столе элементы управления или колесико мыши (см. главу 1).

[4] Способы, представленные в данном примере, вполне применимы и для проекта приложения Silverlight.

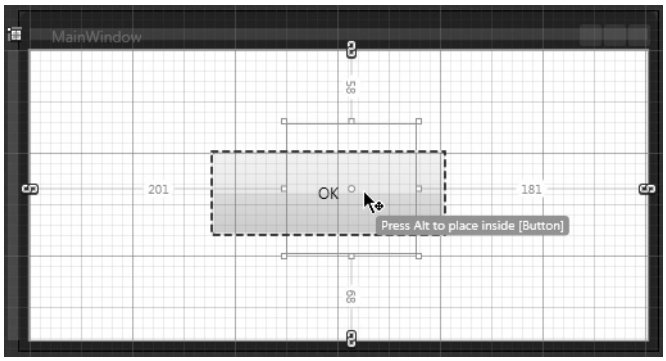


Рис. 4.6. Составное содержимое хранится в диспетчере компоновки

В настоящий момент совершенно очевидно, что размеры встроенного диспетчера компоновки нужно увеличить, чтобы охватить им внутреннюю часть элемента управления. Это можно сделать инструментом обычного выделения Selection, доступным на панели Tools с помощью кнопки, обозначенной пиктограммой черной стрелки, или же оперативной клавиши <V>. В рассматриваемом здесь примере требуется расположить по горизонтали, а не по вертикали данные, хранящиеся в объекте типа `StackPanel`. С этой целью установите в его свойстве `Orientation` (Ориентация) значение `Horizontal` (Горизонтальная) на панели Properties.

Если вы теперь перейдете к панели Objects and Timeline, то сможете выбрать диспетчер компоновки элемента управления типа `Button`, т.е. объект типа `StackPanel`, а затем добавить любое количество дополнительных компонентов. Итак, добавьте выбранную вами геометрическую форму, например эллипс, звезду или стрелку, а также элемент управления типа `Label` в данный диспетчер компоновки. На рис. 4.7 показан один из возможных вариантов заполнения содержимым диспетчера компоновки.

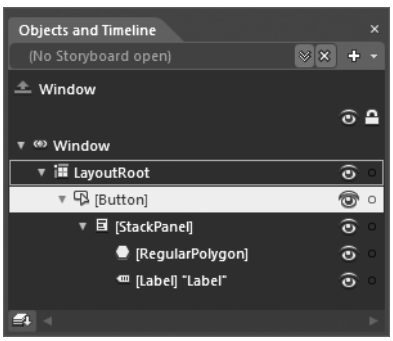


Рис. 4.7. Иерархическое представление элемента управления типа `Button` с диспетчером компоновки типа `StackPanel` и его потомками в качестве содержимого

Теперь можете выбрать каждый компонент для последующей правки. Как показано на рис. 4.8, некоторые основные свойства объекта типа `Label`, обозначающего метку, были изменены на панели Properties, в том числе размер шрифта в области Text, расположение текста в области Layout, а также текст **OK!** самой метки в свойстве `Content`. Кроме того, для обводки по контуру и заполнения внутренней части формы звезды нужными цветами были специально настроены соответствующие кисти.

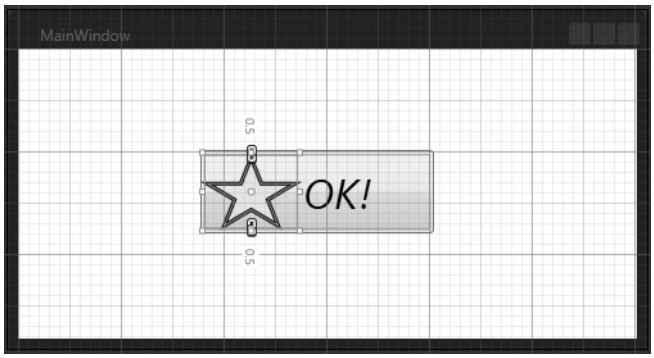


Рис. 4.8. Необычный вид кнопки

Но самое интересное, что для рассматриваемого здесь элемента управления была создана анимация изменения цвета звезды в течение одной секунды, при условии, что объект типа `Button` находится в оперативной памяти. Для этой цели было активизировано свойство `AutoReverse`, а в свойстве `RepeatBehavior` установлено значение **`Forever`**. Для того чтобы не повторять снова все особенности работы с редактором анимации, подробно рассматриваемым в главе 3, приведем лишь самые основные этапы создания раскадровки новой анимации.

- Создайте раскадровку на панели Objects and Timeline.
- Введите в раскадровку ключевые кадры анимации, воспользовавшись кнопкой Record Keyframe с яйцевидной пиктограммой.
- Внесите изменения в свойства выбранного элемента пользовательского интерфейса, чтобы зарегистрировать изменения в его состоянии.

Обработка событий, наступающих для элементов управления с составным содержимым

При создании элемента управления с составным содержимым можно по желанию организовать обработку событий, наступающих для любых подчиненных компонентов. В частности, для компонента в форме звезды, представленного объектом типа `RegularPolygon`, можно обрабатывать событие `MouseDown`, наступающее при нажатии кнопки мыши, а для самой кнопки — событие `Click`. Подобным образом удается зафиксировать факт щелчка не только на кнопке вообще, но и на отдельных частях ее составного содержимого. Ниже приведен исходный код некоторых обработчиков упомянутых выше событий, при наступлении которых в главном окне приложения выводится сообщение в зависимости от того, где именно был произведен щелчок[5].

```
private void regularPolygon_MouseDown(object sender,
    System.Windows.Input.MouseButtonEventArgs e)
{
  this.Title = "You clicked on the star!"
  // Оставить всплывание события от мыши на уровень кнопки.
  e.Handled = true;
}
```

[5] Как пояснялось в главе 2, с помощью кнопки Events, доступной на панели Properties и обозначенной пиктограммой с изображением молнии, можно сформировать обработчики событий, наступающих для графического элемента, выбранного в настоящий момент на монтажном столе.

```
private void Button_Click(object sender, System.Windows.RoutedEventArgs e)
{
  this.Title = "You clicked on the Button!";
}
```

В приведенном выше коде обратите внимание на следующее: при выполнении щелчка кнопкой мыши исключается распространение события по дереву разметки XAML благодаря установке логического значения `true` в свойстве `Handled` (Обработано) наступающего события, передаваемого обработчику событий в качестве аргумента. Если этого не сделать, событие от мыши будет сначала обработано на уровне формы звезды, а затем на уровне самой кнопки. Но в любом случае запустите свое приложение на выполнение, нажав функциональную клавишу <F5>, чтобы проверить, будет ли кнопка реагировать на щелчки кнопкой мыши предполагаемым образом.

Повторное использование составного содержимого

Проанализировав разметку, автоматически сформированную в коде XAML, вы обнаружите, что элемент управления типа `Button` определен в ней предполагаемым образом. При этом диспетчер компоновки типа `StackPanel` определяет непосредственное содержимое данного элемента управления и сам содержит два элемента:

```
<Button Height="65" Margin="147,73,153,0" VerticalAlignment="Top"
   Click="Button_Click">
  <StackPanel Height="57" Width="174" Orientation="Horizontal">
    <ed:RegularPolygon x:Name="regularPolygon" Fill="#FFE3EF0D" Height="56"
      InnerRadius="0.47211" PointCount="5" Stretch="Fill" Stroke="Black"
      HorizontalAlignment="Left" Width="71" StrokeThickness="3"
      MouseDown="regularPolygon_MouseDown"/>
    <Label Content="OK!" HorizontalAlignment="Center"
          VerticalAlignment="Center"
          FontSize="32" FontStyle="Italic"/>
  </StackPanel>
</Button>
```

Но, как упоминалось в главе 3, при создании анимации в среде Expression Blend IDE ее раскадровка вводится в виде именованного ресурса объекта типа `Window` или объекта типа `UserControl`. Проанализировав разметку, описывающую анимацию, вы непременно обнаружите, что объекты, содержащиеся в специальной кнопке, упоминаются в этой разметке прямо по имени, как, например, объект `regularPolygon`, обозначающий форму звезды в составном содержимом данной кнопки.

```
<Window.Resources>
  <Storyboard x:Key="AnimateStar" AutoReverse="True" RepeatBehavior="Forever">
    <ColorAnimationUsingKeyFrames
      Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
      Storyboard.TargetName="regularPolygon">
      <EasingColorKeyFrame KeyTime="0" Value="#FFE3EF0D"/>
      <EasingColorKeyFrame KeyTime="0:0:1" Value="#FF2620F1"/>
    </ColorAnimationUsingKeyFrames>
  </Storyboard>
</Window.Resources>
```

Аналогичным образом в разметке для объекта типа `Window` описывается триггер, запускающий анимационную последовательность. По существу, рассматриваемая здесь специальная кнопка действует довольно обособленно. И в связи с этим возникает следующий вопрос: что, если в пользовательском интерфейсе потребуются *три* специальные кнопки с формой звезды?

В настоящий момент элемент управления типа `Button` тесно связан с конкретным объектом типа `Window`, и поэтому им *нельзя* воспользоваться повторно. Конечно, элемент управления типа `Button` можно скопировать и вставить для повторного использования его составного содержимого, но анимация все равно будет постоянно связана с формой звезды из оригинала кнопки.

В главе 5 будет подробно рассмотрен процесс построения специальных объектов типа `UserControl`, шаблонов и стилей оформления. При этом будет показано, что чаще всего имеет смысл (и даже полезно) определять элемент управления с составным содержимым в качестве особого стиля оформления или объекта типа `UserControl`, поскольку именно так его можно использовать повторно.

А до тех пор просто запомните, что в большинстве элементов управления на платформах WPF и Silverlight поддерживается составное содержимое. Такое содержимое можно сформировать в среде Expression Blend IDE, введя в его пределы диспетчер компоновки и заполнив связанными элементами. С данным вопросом тесно связано понятие рассматриваемой далее модели содержимого многокомпонентных элементов управления.

Исходный код. Исходный код примера проекта `BlendControlContent` находится в папке `Ch 4 Code` загружаемого архива примеров проектов к данной книге.

Представление о модели содержимого многокомпонентных элементов управления

Целый ряд элементов управления на платформах WPF и Silverlight обладает способностью содержать список элементов типа `ListBox` или `ComboBox`. Подобно подлинным элементам управления содержимым, элементы управления типа `ListBox` могут содержать простую порцию строковых данных или более сложные элементы, составляющие целый список. В самом общем смысле такой список специальных элементов можно считать "содержимым", а более конкретно — в подобных элементах управления применяется *модель содержимого многокомпонентных элементов управления*[6].

В качестве примера создайте не совсем обычный элемент управления типа `ListBox` в совершенно новом проекте приложения WPF — `FancyListBox`. С этой целью добавьте элемент управления типа `ListBox` на монтажном столе нового проекта и присвойте ему имя `customListBox` (специальное списковое окно) на панели Properties. Затем добавьте на монтажном столе элемент управления типа `Label`, присвоив ему имя `currentSelection` (выбранное в текущий момент) и расположив его чуть ниже элемента управления типа `ListBox`. Вновь введенный элемент управления типа `Label` будет служить для отображения конкретного значения выбранного в настоящий момент элемента, и для этой цели он будет настроен чуть позже.

Добавление объектов типа `ListBoxItems`

Для того чтобы добавить элементы в список, представленный элементом управления типа `ListBox`, можно, в частности, щелкнуть правой кнопкой мыши на этом элементе управления непосредственно на монтажном столе и выбрать команду Add ListBoxItem (Добавить элемент спискового окна) из всплывающего контекстного меню. (Если же выбрать элемент управления типа `ComboBoxItem`, то в контекстном меню соответственно появится команда Add ComboBoxItem.) Подобным способом к выбранному элементу управления добавляется новый объект типа `ListBoxItem`, содержащий простой текст,

---

[6] Такие элементы управления расширяют свой родительский класс `ItemsControl`.

который можно в дальнейшем отформатировать. Итак, воспользуйтесь данным способом, чтобы добавить в список, представленный элементом управления типа `ListBox`, единственный элемент в виде объекта типа `ListBoxItem` (рис. 4.9).

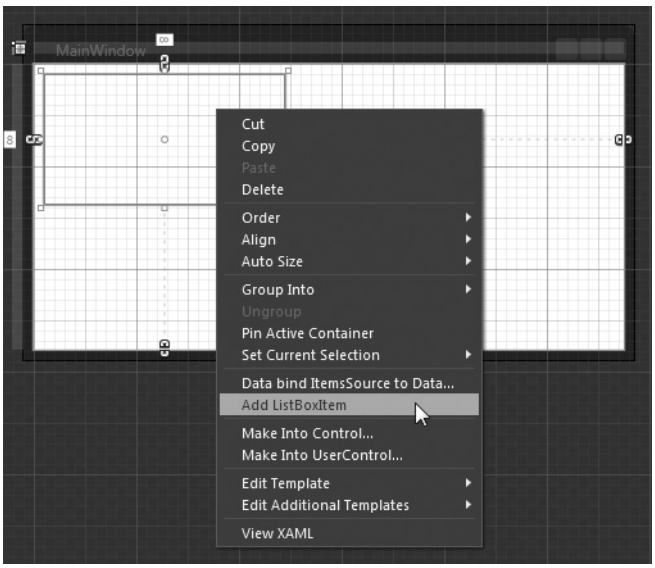


Рис. 4.9. Добавление новых элементов в список основного элемента управления

Выделите и растяните мышью новый объект типа `ListBoxItem`, чтобы заполнить им элемент управления типа `ListBox` приблизительно наполовину высоты последнего, как показано на рис. 4.10.

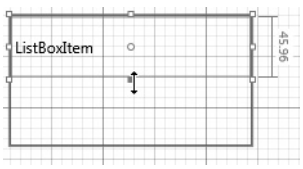


Рис. 4.10. Изменение размеров объекта типа `ListBoxItem`

Убедитесь сначала в том, что объект типа `ListBoxItem` выбран в настоящий момент на панели Objects and Timeline, а затем добавьте к этому объекту элемент управления типа `StackPanel` таким же образом, как делали это в предыдущем примере проекта с необычной кнопкой. Выберите элемент управления типа `StackPanel` на панели Objects and Timeline, найдите свойство `Orientation` на панели Properties, введя, например, его имя в поле поиска Search, и установите в этом свойстве значение **`Horizontal`**. После этого вам придется изменить размеры нового элемента управления типа `StackPanel`, чтобы благополучно вписать его в объект типа `ListBoxItem`, который его содержит. Поэтому уделите немного времени упорядочению этих объектов на монтажном столе по своему усмотрению.

А далее следует самое интересное. Выберите сначала элемент управления типа `StackPanel` на панели Objects and Timeline, а затем добавьте к нему объекты типа `Ellipse` и `Label` стандартными для Expression Blend способами, используя, в частности, панель Tools, библиотеку ресурсов и т.д. После этого иерархическое представление

рассматриваемого здесь элемента управления на панели Objects and Timeline должно выглядеть так, как показано на рис. 4.11.

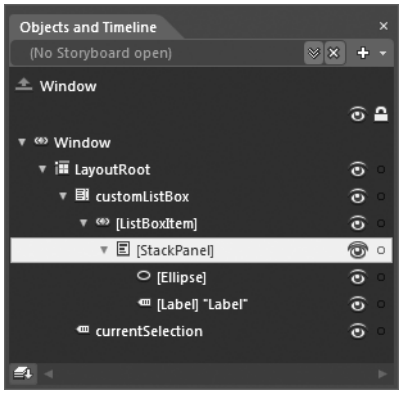


Рис. 4.11. Иерархическое представление объекта типа `ListBoxItem` в качестве элемента управления с составным содержимым

Измените размеры новых объектов, чтобы вписать их в элемент управления с составным содержимым. Выберите на панели Properties красный цвет заполнения для свойства `Fill` объекта типа `Ellipse` и установите текстовое значение **`Red!`** (Красный) в свойстве `Content` объекта типа `Label`. Текущая компоновка элемента управления с составным содержимым приведена на рис. 4.12.

Выберите объект типа `ListBoxItem` на панели Objects and Timeline, щелкните на нем правой кнопкой мыши и выберите команду Copy из всплывающего контекстного меню. После этого выберите элемент управления типа `ListBox` на панели Objects and Timeline, щелкните на нем правой кнопкой мыши и выберите команду Paste из всплывающего контекстного меню. Повторите эту же процедуру еще три раза, чтобы в конечном итоге заполнить элемент управления типа `ListBox` четырьмя объектами типа `ListBoxItem`. Затем воспользуйтесь панелью Properties, чтобы подобрать особые цвета заполнения для свойств `Fill` новых объектов типа `Ellipse` и установить соответствующие им текстовые значения в свойствах `Content` новых объектов типа `Label`. В качестве примера на рис. 4.13 (а также на цветной вклейке к книге) приведен окончательный вид компоновки элемента управления типа `ListBox`.

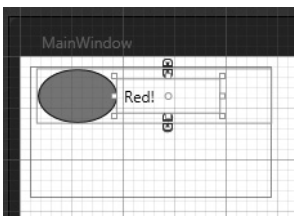


Рис. 4.12. Компоновка объекта типа `ListBoxItem` в качестве элемента управления с составным содержимым

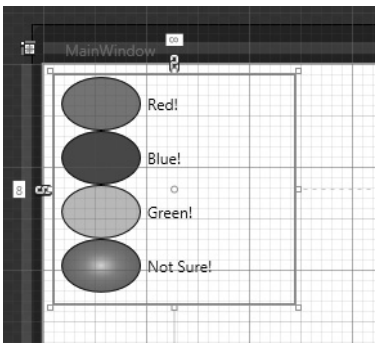


Рис. 4.13. Окончательный и не совсем обычный вид элемента управления типа `ListBox`

Просмотр разметки в коде XAML

Если вы перейдете к редактору XAML, то обнаружите, что элемент управления типа `ListBox` действительно содержит четыре объекта типа `ListBoxItem`. Но вместо простого текста они теперь содержат составные элементы пользовательского интерфейса, как показано ниже.

```
<ListBoxItem Height="41.96" Width="180">
  <StackPanel Height="41.96" Width="180" Orientation="Horizontal">
    <Ellipse Fill="#FF48D416" Stroke="Black" Width="62"/>
    <Label Content="Green!" Width="78.993" HorizontalAlignment="Center"
          VerticalAlignment="Center"/>
  </StackPanel>
</ListBoxItem>
```

Обнаружение того, что выбрано в текущий момент

Независимо от содержимого объектов типа `ListBoxItem`, будь то простой текст или блочная панель типа `StackPanel` с несколькими элементами, выявить элемент, выбранный из списка, можно с помощью свойства `SelectedIndex` (Индекс выбранного), имеющегося у элемента управления типа `ListBox`. Из этого свойства возвращается отсчитываемое от нуля значение, где **0** обозначает первый элемент в списке, **1** — второй элемент и т.д.

Итак, выберите элемент управления типа `ListBox` на панели Objects and Timeline и щелкните на кнопке Events панели Properties, чтобы организовать обработку события `SelectionChanged`, наступающего при выборе нового элемента из списка, представленного элементом управления типа `ListBox`, как показано на рис. 4.14. (Напомним, что для автоматического формирования обработчика событий в файле исходного кода текущего проекта достаточно дважды щелкнуть на текстовом поле рядом с меткой обрабатываемого события на панели Properties.)

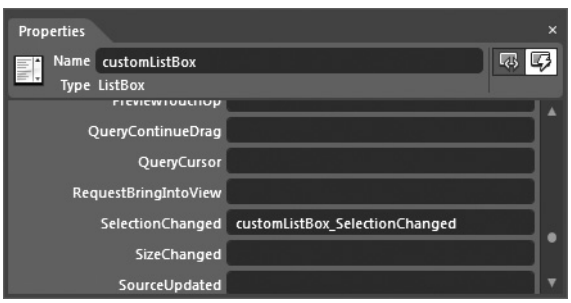


Рис. 4.14. Обработка события `SelectionChanged`, связанного с элементом управления типа `ListBox`

Ниже показано, каким образом в коде обработчика событий реализуется отображение порядкового номера элемента, выбранного в текущий момент из списка, в поле элемента управления `currentSelection` типа `Label`.

```
private void customListBox_SelectionChanged(object sender,
  System.Windows.Controls.SelectionChangedEventArgs e)
{
  this.currentSelection.Content =
    string.Format("You Picked Item Number: {0}",
          this.customListBox.SelectedIndex);
}
```

Запустите свое приложение на выполнение и обратите внимание на возможность выбрать, как и предполагалось, любой специальный элемент из списка. Пример такого выбора приведен на рис. 4.15 (а также на цветной вклейке к книге).

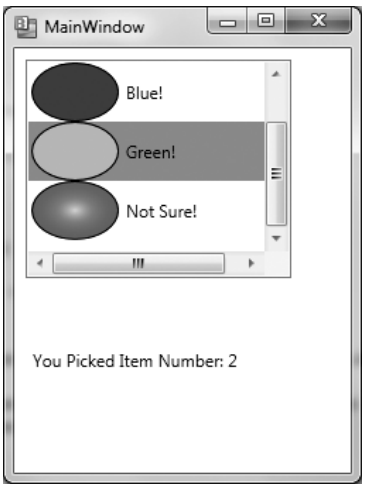


Рис. 4.15. Не совсем обычное списковое окно типа `ListBox` в действии

Применение свойства `Tag`

Итак, вы научились обнаруживать порядковый номер элемента, выбранного в текущий момент из списка. Но было бы намного интереснее обнаруживать и цвет (красный, зеленый и т.д.), связанный с выбранным элементом. И этого, как оказывается, вполне возможно добиться с помощью *шаблона данных*, более подробно рассматриваемого в главе 6. А до тех пор в качестве простой альтернативы придется воспользоваться свойством `Tag`.

Выберите на панели Objects and Timeline объект типа `ListBoxItem`, содержащий красный эллипс. После этого найдите на панели Properties свойство `Tag` (Признак) — оно находится среди дополнительных свойств, выбираемых из области Common Properties. Установите в этом свойстве значение `Red`. Повторите эту же процедуру для остальных объектов типа `ListBoxItem`, настроив их свойства `Tag` на соответствующий цвет эллипса. Ниже приведен пример описания данного свойства в разметке объекта типа `ListBoxItem` с зеленым эллипсом.

```
<ListBoxItem Height="41.96" Width="180" Tag="Green">
  ...
</ListBoxItem>
```

Теперь обновите имеющийся обработчик событий типа `SelectionChanged`, чтобы извлекать значение свойства `Tag` выбранного в настоящий момента объекта типа `ListBoxItem` и присваивать это значение свойству `Content` элемента управления типа `Label`, как показано ниже.

```
private void customListBox_SelectionChanged(object sender,
  System.Windows.Controls.SelectionChangedEventArgs e)
{
  string selectedColor =
    ((ListBoxItem)this.customListBox.SelectedItem).Tag.ToString();
```

```
    this.currentSelection.Content = string.Format("You Picked the Color: {0}",
        selectedColor);
}
```

На примере рассмотренного здесь проекта было показано, насколько упрощается процесс специального оформления внутреннего содержимого элементов управления в приложениях WPF и Silverlight. С помощью всего лишь нескольких строк разметки можно заполнить элемент управления содержимым практически любой сложности. Но это содержимое должно быть упорядочено в элементе управления по определенной системе. А теперь перейдем к более подробному рассмотрению диспетчеров компоновки, имеющихся в нашем распоряжении.

Исходный код. Исходный код примера проекта `FancyListBox` находится в папке `Ch 4 Code` загружаемого архива примеров проектов к данной книге.

Работа с диспетчерами компоновки

Как пояснялось в предыдущих главах, при создании нового проекта приложения WPF или Silverlight в среде Expression Blend IDE в качестве используемого по умолчанию диспетчера компоновки автоматически предоставляется объект типа `Grid`. Напомним, что этот объект называется `LayoutRoot`, хотя его имя нетрудно изменить с помощью свойства `Name`. Так, если создать новый проект приложения WPF, в среде Expression Blend IDE будет автоматически сформирована приведенная ниже разметка в коде XAML.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="WPFLayoutManagers.MainWindow"
  x:Name="Window"
  Title="MainWindow"
  Width="640" Height="480">
  <Grid x:Name="LayoutRoot"/>
</Window>
```

Предоставляемый по умолчанию объект типа `Grid` вполне подходит для большинства приложений на платформе WPF или Silverlight, но имеются и другие диспетчеры компоновки. Оказывается, что в прикладном интерфейсе WPF API предоставляется более широкий выбор диспетчеров компоновки, чем в прикладном интерфейсе Silverlight API. Но на практике это редко вызывает какие-либо осложнения в работе над проектами. Напомним, что приложение Silverlight, как правило, разрабатывается лишь как малая часть более крупной веб-страницы и практически не изменяет свои исходные размеры в окне браузера, где оно размещается. Именно поэтому на платформе Silverlight отсутствуют диспетчеры компоновки, динамически переставляющие элементы пользовательского интерфейса в зависимости от их размеров.

С другой стороны, приложения WPF чаще всего разрабатываются таким образом, чтобы объекты типа `Window` можно было разворачивать, сворачивать и изменять их размеры подобно обычным окнам. Именно по этой причине в прикладном интерфейсе WPF API предоставляется целый ряд дополнительных средств для динамической компоновки пользовательского интерфейса. В табл. 4.1 приводится краткое описание основных диспетчеров компоновки и указываются те прикладные интерфейсы API, в которых они поддерживаются.

Таблица 4.1. Основные диспетчеры компоновки на платформах WPF и Silverlight

Диспетчер компоновки	Поддержка в WPF API	Поддержка в Silverlight API	Назначение
`Canvas`	X	X	Располагает порожденные объекты в абсолютных координатах X, Y. Идеально подходит для хранения сложных графических данных
`DockPanel`	X	X	Пристыковывает порожденные объекты к указанному (верхнему, нижнему, левому, правому) краю контейнера
`Grid`	X	X	Определяет ряд ячеек сетки (по строкам и столбцам) для расположения порожденных объектов
`ScrollViewer`	X		Прокручивает содержащиеся в нем элементы. Такой диспетчер компоновки может содержать только один элемент пользовательского интерфейса, который почти всегда оказывается еще одним диспетчером компоновки
`StackPanel`	X	X	Располагает порожденные объекты на одной вертикальной или горизонтальной линии
`UniformGrid`	X		Располагает порожденные объекты равномерной сеткой, как, например, в игре в крестики-нолики
`ViewBox`	X		Масштабирует все порожденные объекты подобно инструменту изменения масштаба. Как и диспетчер компоновки типа `ScrollViewer`, почти всегда содержит подчиненный диспетчер компоновки
`WrapPanel`	X		Располагает порожденные объекты рядами слева направо. По достижении правого края панели остальные объекты автоматически переносятся на следующую строку и так далее слева направо и сверху вниз

Дополнительные типы диспетчеров компоновки

Помимо основных диспетчеров компоновки, перечисленных в табл. 4.1, на платформах WPF и Silverlight предоставляются классы `Border` и `TabControl`. Как подразумевают их имена, в классе `Border` определяются видимые границы (и дополнительно фон) для порожденных объектов, связанных одним и тем же расположением, тогда как в классе `TabControl` предоставляются средства для расположения элементов пользовательского интерфейса на независимых панелях, снабженных вкладками. Ознакомиться с классами `TabControl` и `Border` на практике вы сможете далее в этой главе.

Кроме того, в прикладном интерфейсе WPF API предоставляется элемент управления типа `BulletDecorator`, позволяющий связывать вместе два элемента пользовательского интерфейса (как правило, графику и фрагмент текста) для быстрого построения специальных маркированных списков. Преимущество такого группирования связанного вместе содержимого в едином элементе управления типа `BulletDecorator`, а не в автономном и пустом диспетчере компоновки типа `StackPanel` заключается в том, что у данного элемента управления имеются свойства для независимого (и простого) взаимодействия с графическим изображением маркера абзаца через одноименное свойство `Bullet`. Ниже приведен пример связывания данных изображения формата JPG с текстовыми данными в элементе разметки текстового блока.

```
<BulletDecorator Background="#FF001BFF" Height="53">
  <BulletDecorator.Bullet>
    <Image Source="apple.jpg" Height="60" Width="60"/>
```

```
  </BulletDecorator.Bullet>
  <TextBlock Width="257" TextWrapping="Wrap"
    HorizontalAlignment="Center"
    Foreground ="#FF849DB8"
    FontSize="35"
    VerticalAlignment="Center">
      <Run Text="Yum! Apples!"/>
  </TextBlock>
</BulletDecorator>
```

Так или иначе, работая с любым типом диспетчера компоновки, в том числе и с таким простым, как `BulletDecorator`, следует иметь в виду, что подходящую компоновку по определенной системе можно создать самыми разными способами. В частности, можно активизировать режим виртуального полотна на сетке, представленной объектом типа `Grid`, чтобы добиться абсолютного расположения содержимого в зависимости от размеров и полей, установленных для элементов пользовательского интерфейса из этого содержимого. С другой стороны, ту же самую компоновку можно создать с помощью коллекции вложенных объектов типа `StackPanel`, расположенных с разной ориентацией.

Непременно уделите время экспериментированию с каждым типом диспетчеров компоновки и проверке получаемых результатов, изменяя размеры, а возможно, и положение этих контейнеров содержимого во время выполнения. Далее в главе вам предстоит поработать с целым рядом диспетчеров компоновки в контексте более крупного приложения, разрабатываемого в среде Expression Blend IDE, а до тех пор рассмотрим самые основные способы их применения.

Смена типа диспетчера компоновки

После создания нового проекта приложения WPF или Silverlight в среде Expression Blend IDE можете изменить тип любого диспетчера компоновки. Для этого щелкните правой кнопкой мыши на узле выбранного диспетчера компоновки в иерархическом представлении на панели Objects and Timeline и выберите команду Change Layout Type (Сменить тип компоновки) из всплывающего контекстного меню.

Примечание. Если вы сменяете диспетчеры компоновки, содержащие элементы управления, имейте в виду, что эти элементы управления подчиняются правилам компоновки, установленным для их контейнера! Так, если вы расположите графические элементы на сетке, представленной объектом типа `Grid`, а затем смените его на объект типа `StackPanel`, все эти элементы расположатся друг за другом.

На рис. 4.16 приведены варианты компоновки пользовательского интерфейса в проекте приложения WPF. А в проекте приложения Silverlight этих вариантов окажется меньше.

Конструирование вложенных компоновок

В приложениях на платформе WPF или Silverlight очень часто применяется система вложенной компоновки. В частности, можно разделить исходную сетку типа `Grid` на два столбца, расположить в левом столбце вертикальную блочную панель типа `StackPanel`, а в правом столбце — виртуальное полотно типа `Canvas`, чтобы построить систему компоновки для простой программы раскраски.

Когда же требуется добавить порожденные диспетчеры компоновки ниже родительского диспетчера компоновки, выберите сначала родительский узел на панели Objects and Timeline, затем порожденный узел для добавления в области Layout на панели Tools

и дважды щелкните на нужном элементе. Вам совсем не обязательно делать это теперь, а сам процесс наглядно показан на рис. 4.17.

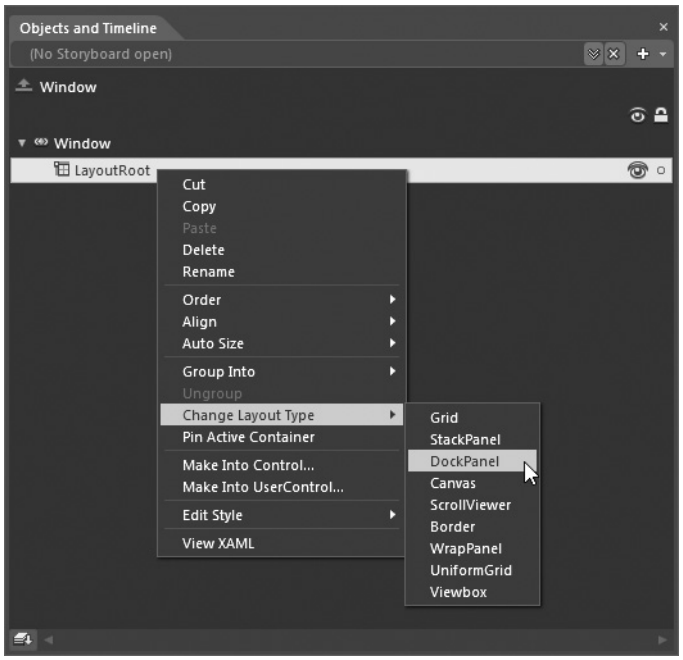


Рис. 4.16. Смена диспетчера компоновки на панели Objects and Timeline

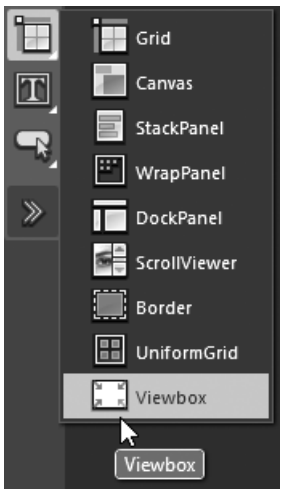


Рис. 4.17. На панели Tools можно выбирать и добавлять порожденные объекты компоновки

Группирование и разгруппирование выбранных элементов пользовательского интерфейса

Выделение на монтажном столе и группирование выбранных элементов пользовательского интерфейса в новый диспетчер компоновки, вложенный в родительский диспетчер, осуществляется в Expression Blend довольно просто. Допустим, имеется объект типа `Grid`, содержащий три объекта типа `Button`, которые требуется расположить рядом в объекте типа `Grid`. Если выбрать все три объекта на монтажном столе (для чего достаточно нажать клавишу <Shift> и щелкнуть сначала на первом объекте, а затем на последнем), то далее остается лишь щелкнуть правой кнопкой мыши на выделенных объектах и выбрать команду Group Into (Сгруппировать) из всплывающего контекстного меню, как показано на рис. 4.18.

Примечание. Несколько элементов пользовательского интерфейса можно также выбрать на панели Objects and Timeline и затем активизировать команду меню Group Into.

Если предположить, что три объекта типа `Button` сгруппированы в диспетчер компоновки типа `StackPanel`, то их иерархическое представление на панели Objects and Timeline будет выглядеть так, как показано на рис. 4.19. Однако новый диспетчер типа `StackPanel` для данной компоновки может быть выбран независимым образом на панели Properties.

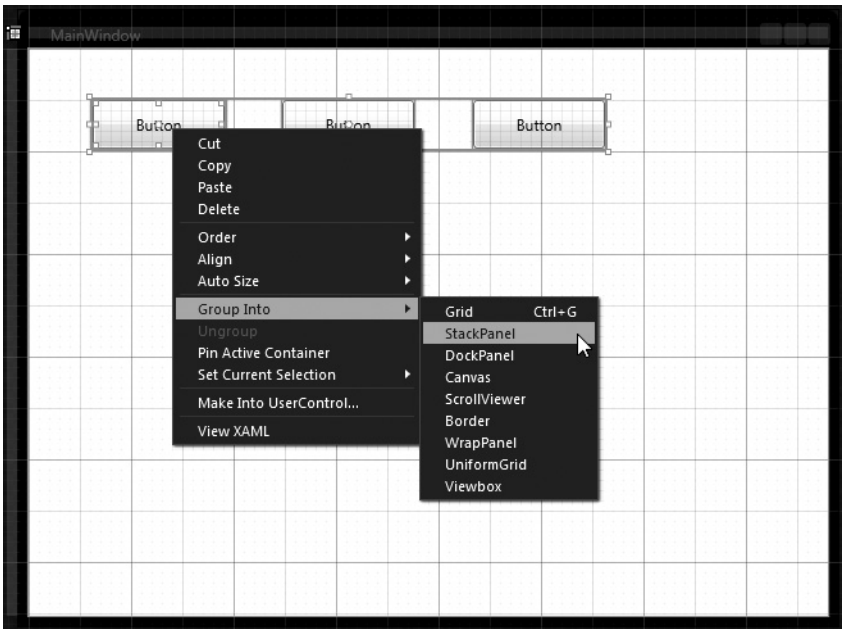


Рис. 4.18. Группирование выбранных объектов в диспетчер компоновки

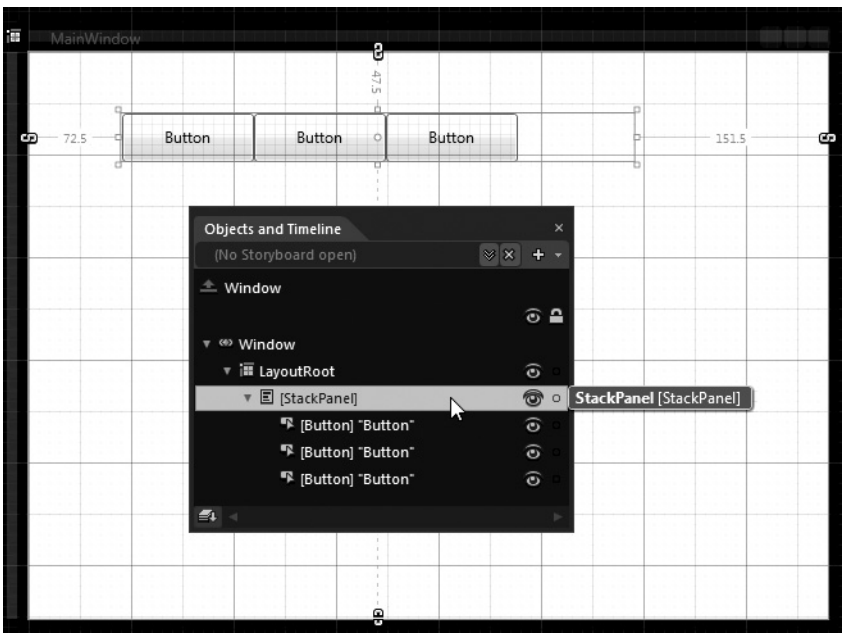


Рис. 4.19. Простая система вложенной компоновки

Для расформирования системы вложенной компоновки достаточно выбрать сначала диспетчер компоновки на монтажном столе и на панели Objects and Timeline, а затем щелкнуть на нем правой кнопкой мыши и выбрать команду Ungroup (Разгруппировать)

из всплывающего контекстного меню. В итоге порожденный диспетчер компоновки будет удален из иерархического представления в разметке XAML, а содержавшиеся в нем элементы пользовательского интерфейса будут размещены непосредственно в родительском диспетчере компоновки.

Перестановка элементов пользовательского интерфейса в диспетчерах компоновки

Если для пользовательского интерфейса главного окна (объекта типа `Window`) приложения WPF или начальной веб-страницы (объекта типа `UserControl`) приложения Silverlight построена система вложенной компоновки, то в ней иногда возникает потребность в перестановке отдельных элементов. Продолжая рассмотренный выше пример, допустим, что, после того, как объекты типа `Button` были сгруппированы в новый диспетчер компоновки типа `StackPanel`, крайний справа объект типа `Button` решено расположить в родительском диспетчере компоновки типа `Grid`. Подобную перестановку можно, в частности, сделать, выбрав нужный объект типа `Button` на монтажном столе и перетащив его за пределы текущего контейнера в требуемый внешний контейнер, а затем нажав клавишу <Alt>, аналогично тому, как это делалось при создании составного содержимого ранее в этой главе (рис. 4.20).

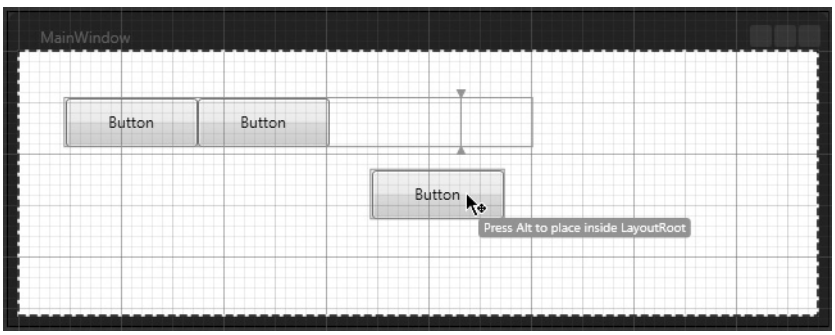


Рис. 4.20. Перемещение элемента пользовательского интерфейса в родительский диспетчер компоновки

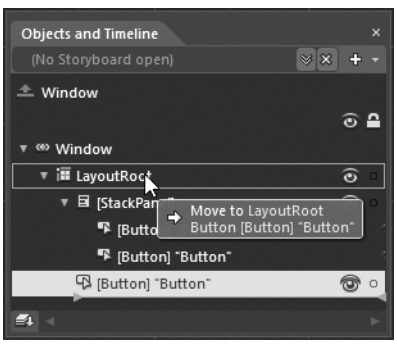


Рис. 4.21. Перемещать элементы пользовательского интерфейса из одного диспетчера компоновки в другой можно и на панели Objects and Timeline

Элементы пользовательского интерфейса можно также перетаскивать из одного диспетчера компоновки в другой непосредственно на панели Objects and Timeline. О такой возможности полезно всегда помнить, поскольку после добавления нового элемента из панели Tools на монтажный стол он будет содержаться в выбранном в настоящий момент диспетчере компоновки. Поэтому если случайно выбрать не тот диспетчер компоновки перед добавлением очередного элемента пользовательского интерфейса, то эту ошибку нетрудно исправить, перетащив узел данного объекта в нужный контейнер, как показано на рис. 4.21.

Построение пользовательского интерфейса в Expression Blend

Теперь, когда у вас сложилось представление о назначении диспетчеров компоновки, модели содержимого элементов управления и модели содержимого многокомпонентных элементов управления, перейдем к рассмотрению примера нового проекта, чтобы продемонстрировать применение на практике различных средств Expression Blend IDE для построения пользовательского интерфейса приложения. В данном примере речь пойдет о проекте приложения на платформе WPF, но многие из представленных далее способов вполне применимы и в проектах приложений на платформе Silverlight. Итак, создайте новый проект приложения WPF, присвоив ему имя `WpfControlsApp`.

Построение системы компоновки с вкладками

В данном проекте вам предстоит построить систему компоновки с вкладками, где на каждой вкладке демонстрируется применение конкретного ряда элементов управления, диспетчеров компоновки и прикладных интерфейсов API. Поэтому вы должны прежде всего найти компонент `TabControl` в библиотеке ресурсов, воспользовавшись функцией поиска Search, как показано на рис. 4.22.

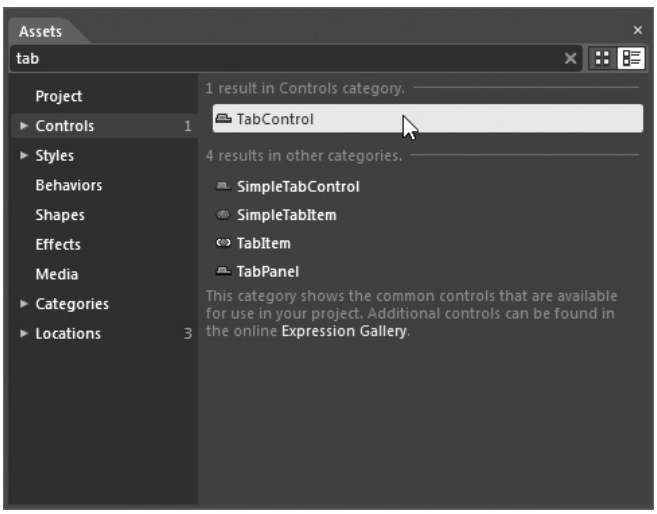


Рис. 4.22. Обнаружение компонента `TabControl`

Примечание. В оставшейся части главы предполагается, что для оперативного поиска нужного элемента пользовательского интерфейса вы пользуетесь функцией поиска, и поэтому моментальные снимки экранов с элементами, найденными в библиотеке ресурсов, далее не показываются.

Найдя компонент `TabControl`, выберите его на панели Tools и перетащите экземпляр данного элемента управления на исходную сетку (объект типа `Grid`), чтобы он занял большую ее часть. По умолчанию в элементе управления типа `TabControl` определяются два исходных элемента (объекты типа `TabItem`), каждый из которых представляет в нем отдельную вкладку. Кроме того, эти элементы располагаются в верхней части пространства, принадлежащего элементу управления типа `TabControl`. Если хотите

изменить расположение вкладок, то сделайте это с помощью свойства `TabStripPlacement` (Расположение панели вкладок), доступного в области Common Properties на панели Properties выбранного элемента управления типа `TabControl`, но не отдельных его элементов (т.е. объектов типа `TabItem`), как показано на рис. 4.23.

Примечание. В оставшейся части главы предполагается, что для оперативного обнаружения рассматриваемого свойства вы также пользуетесь функцией поиска. Там, где это действительно необходимо, соответствующие моментальные снимки экрана все же приводятся для большей наглядности описываемого процесса разработки. А в остальном поиск нужных свойств и событий возлагается на вас, читатель.

А теперь перейдите к панели Objects and Timeline и обратите внимание на то, что каждый объект типа `TabItem` является порожденным, т.е. потомком родительского объекта типа `TabControl`. Но еще интереснее отметить то обстоятельство, что у каждого объекта типа `TabItem` имеется свой собственный диспетчер компоновки (в данном случае объект типа `Grid`).[7] Более того, у каждого объекта типа `TabItem` имеется элемент `Header` (Заголовок), в котором можно задать текст или иное сложное содержимое для отдельной вкладки. В качестве примера на рис. 4.24 приведено иерархическое представление не видоизмененной до сих пор системы компоновки.

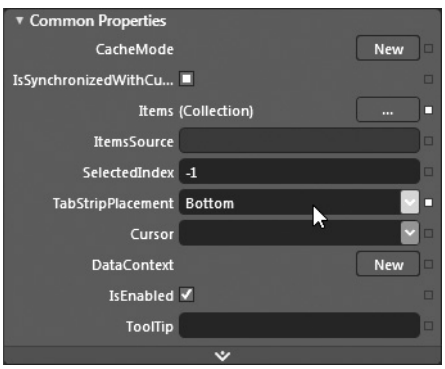


Рис. 4.23. Определение местоположения вкладок (объектов типа `TabItem`) в элементе управления типа `TabControl`

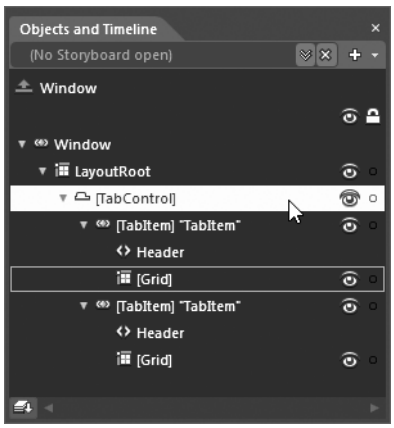


Рис. 4.24. У каждого объекта типа `TabItem` имеется свой собственный диспетчер компоновки

В целях рассматриваемого здесь примера добавьте еще одну вкладку в элемент управления типа `TabControl`. Для этого щелкните правой кнопкой мыши на узле `TabControl` непосредственно на панели Objects and Timeline и выберите команду Add TabItem (Добавить элемент вкладки) из всплывающего контекстного меню. После этого внесите в свойство `Header` каждого объекта типа `TabItem` изменения, введя в его поле приведенный ниже текст заголовков.

- **`Fun with Ink`** (Забавное рисование пером).
- **`Documents API`** (Прикладной интерфейс документов).
- **`Behaviors!`** (Виды поведения).

---

[7] Это характерный пример *модели содержимого заголовочных элементов управления*, подобной рассматривавшимся ранее в настоящей главе модели содержимого элементов управления и модели содержимого многокомпонентных элементов управления.

Примечание. Стремясь внести изменения в текст заголовков, вы мало чего добьетесь, если попытаетесь выбрать элемент `Header` на панели Objects and Timeline, поскольку он является свойством объекта типа `TabItem`, а не выбираемым элементом. Воспользуйтесь лучше панелью Properties для изменения текста.

Следует особенно иметь в виду, что, выбирая любой объект типа `TabItem` на монтажном столе, вы тем самым активизируете соответствующую вкладку для правки, а после этого можете приступать к перетаскиванию элементов управления в корневой диспетчер компоновки. Разумеется, вы сможете добиться аналогичного результата, выбрав объект типа `TabItem` на панели Objects and Timeline. И наконец, введите в текстовом поле свойства `Name` имя `myTabControl` элемента управления типа `TabControl`, а в текстовом поле свойств `Name` объектов типа `TabItem` — имена `tabInk`, `tabDocs` и `tabBehaviors` соответственно.

Работа с сеткой

На первой вкладке пользовательского интерфейса рассматриваемого здесь примера проекта должны отображаться данные, введенные пользователем и зафиксированные средствами прикладного интерфейса Ink API, который поддерживается на обеих платформах, WPF и Silverlight, хотя применяемые на этих платформах модели программирования немного отличаются. Этот прикладной интерфейс оказывается очень полезным для написания программ, выполняемых на вычислительных устройствах с сенсорным экраном, где данные, которые пользователь вводит пером или пальцем, должны правильно фиксироваться. Оказывается, что средствами того же самого прикладного интерфейса можно фиксировать также графические штрихи, наносимые мышью, что и предполагается сделать в данном примере. Для хранения каждого элемента пользовательского интерфейса, отображаемого на данной вкладке, будет использоваться диспетчер компоновки типа `Grid`. Объект типа `Grid` может быть настроен на работу в следующих режимах.

- **Режим компоновки на сетке** — устанавливается по умолчанию. В этом режиме создается ряд строк и столбцов, составляющих сетку, а отдельные компоненты располагаются в ячейках сетки. Регулируя размеры ячеек сетки разделителями строк и столбцов, можно изменить также размеры содержащихся в них объектов, чтобы сохранить поля, определяемые в их свойствах `Margin`, как поясняется ниже.
- **Режим компоновки на полотне** — дает возможность редактирования таким же образом, как и на панели Canvas, где отдельные графические элементы располагаются там, где они были первоначально помещены, независимо от их расположения в ячейках сетки.

В устанавливаемом по умолчанию режиме компоновки на сетке порожденные объекты можно помещать в ячейках сетки и задавать (прямо или косвенно) поля с помощью свойств `Margin` объекта, содержащегося в диспетчере компоновки типа `Grid`. У каждого элемента пользовательского интерфейса, располагаемого на сетке, могут быть четыре поля: верхнее, нижнее, левое и правое. Эти поля определяют положение элемента в ячейке сетки и могут быть отредактированы в области Layout на панели Properties с помощью упомянутых выше свойств `Margin`. Такой исходно устанавливаемый режим компоновки оказывается удобным в том случае, если элемент пользовательского интерфейса, размещаемый в ячейке сетки, должен заполнять все ее пространство и увеличиваться или сокращаться при изменении размеров ячейки.

Режим компоновки на полотне отличается тем, что порожденные объекты сетки имеют *абсолютное расположение*. Поэтому, если изменяются размеры любых строк и

столбцов сетки, размеры порожденных объектов не изменяются автоматически, хотя их свойства `Margin` по-прежнему обновляются. Чаще всего для разработки пользовательского интерфейса достаточно устанавливаемого по умолчанию режима компоновки на сетке, но для перехода из одного режима компоновки в другой следует щелкнуть на кнопке-переключателе в левом верхнем углу выбранного редактора сетки, как показано на рис. 4.25.

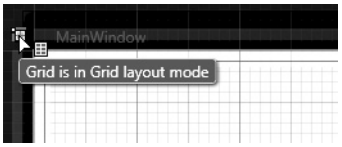


Рис. 4.25. Режимы компоновки на сетке и полотне легко переключаются

Определение строк и столбцов сетки

Ввод строк и столбцов в выбранный объект типа `Grid` осуществляется в среде Expression Blend IDE довольно просто. Для этого активизируйте сначала сам объект типа `Grid` во вкладке `Fun with Ink` на панели Objects and Timeline, а затем разделите этот объект на две строки непосредственно на монтажном столе таким образом, чтобы верхняя строка занимала около четверти всего пользовательского интерфейса. С этой целью поместите курсор на внешнем крае сетки и щелкните кнопкой мыши. В качестве примера на рис. 4.26 показано положение курсора мыши на левом краю редактора сетки.

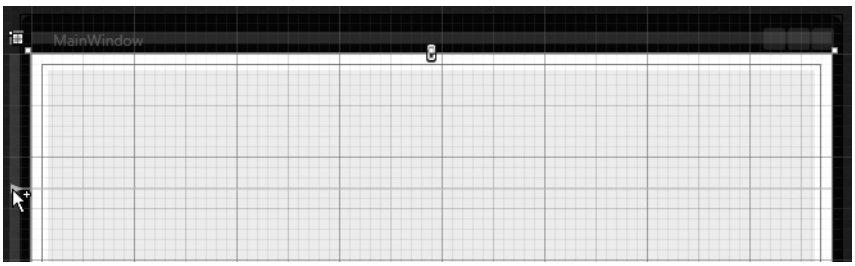


Рис. 4.26. Разделение сетки на две строки

Подобным способом можно ввести любое количество строк или столбцов, а после этого переставить их, перетаскивая соответствующие разделители мышью. Попробуйте ввести еще несколько строк и столбцов и поупражняйтесь в изменении размеров каждой ячейки сетки. По завершении отмените все предыдущие операции, нажимая комбинацию клавиш <Ctrl+Z> до тех пор, пока не вернетесь к виду сетки с двумя строками.

Примечание. Заполнить сетку строками и столбцами можно и с помощью свойств `Column-Definitions` (Определения столбцов) и `RowDefinitions` (Определения строк), находящихся среди дополнительных свойств, доступных в области Layout на панели Properties.

Ввод элементов в ячейки сетки

После определения строк и столбцов данной сетки можете расположить в ее ячейках элементы пользовательского интерфейса, в том числе и геометрические формы. Напомним, что в режиме компоновки на сетке, устанавливаемом для объекта типа `Grid` по умолчанию, элементы как порожденные объекты размещаются в ячейках сетки с

учетом полей, устанавливаемых в свойствах `Margin` данного объекта. В частности, положение порожденного объекта зависит от установок следующих свойств.

- **Выравнивание.** Определяет положение объекта относительно родительского объекта.
- **Поля.** Определяют величину пустого пространства вокруг элемента управления, между внешними краями порожденного объекта и границами ячейки сетки.
- **Высота и ширина.** Фиксированные значения, измеряемые в пикселях (аппаратно-независимых единицах измерения, составляющих около 1/96 дюйма, или 0,26 мм). Эти свойства можно установить в режим **`Auto`**, чтобы размеры порожденных объектов изменялись автоматически в зависимости от размеров родительской панели.

На печатной странице трудно показать, каким образом расположение элементов в ячейках сетки настраивается с помощью упомянутых выше свойств, поэтому введите элемент управления `myInkArea` типа `InkCanvas` в нижней строке сетки, найдя его в библиотеке ресурсов и дважды щелкнув на нем. По умолчанию любой элемент, добавляемый к сетке двойным щелчком, располагается в первой строке и первом столбце. Выделите этот объект, перетащите во вторую строку и растяните его таким образом, чтобы заполнить им всю эту строку, как показано на рис. 4.27.

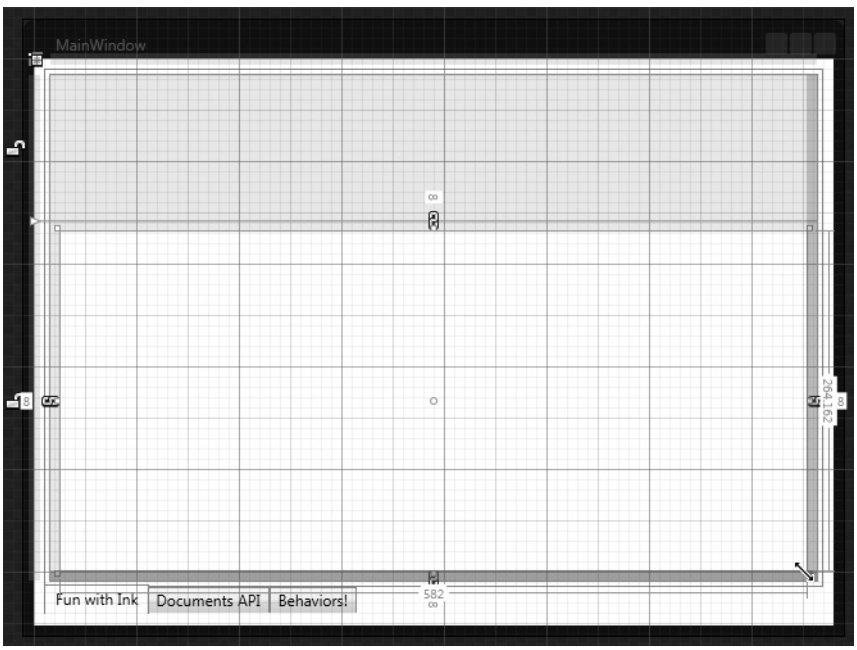


Рис. 4.27. Ввод элемента в строку сетки

Как упоминалось в главе 1, на монтажном столе отображаются красные линии привязки, по которым нетрудно определить, каким образом отдельный элемент использует полезную площадь своего контейнера. В частности, элемент управления `myInkArea` привязан к каждой стороне своей строки, и если изменить размеры строки (или всей сетки), то этот элемент соответственно увеличится или сократится, в чем вы можете убедиться сами. Это же относится к любому элементу, вводимому подобным способом в ячейку сетки.

Прежде чем вводить элементы в верхней строке сетки, рассмотрим свойства выбранного объекта типа `InkCanvas`, доступные в области Layout на панели Properties (рис. 4.28). В этой области можно задавать поля, высоту и ширину (в настоящий момент они установлены в режим `Auto`, поскольку данный объект привязан к своему родительскому объекту), а также указывать вручную строку и столбец для размещения выбранного элемента, если он, конечно, находится на сетке. Попробуйте изменить установки в этих свойствах и обратите внимание на то, каким образом изменяется расположение объекта типа `InkCanvas`, а затем нажмите комбинацию клавиш <Ctrl+Z>, чтобы вернуться к исходной компоновке.

Рис. 4.28. В области Layout на панели Properties можно настроить свойства, определяющие расположение элемента в родительском диспетчере компоновки

Создание разделителя сетки

Многие объекты типа `Grid` становятся невидимыми во время выполнения, а это означает, что пользователь вообще не будет видеть строки и столбцы сетки. Но если вам действительно требуется показать линии сетки, установите логическое значение `true` в свойстве `ShowGridLines` (Показ линий сетки). Несмотря на то что пользователь вообще не видит строки и столбцы сетки в диспетчере компоновки типа `Grid`, очень часто в сетке предоставляются видимые *разделители*, с помощью которых пользователь может изменять размеры строк и столбцов сетки.

Для демонстрации подобной функциональной возможности в текущей компоновке найдите элемент управления типа `GridSplitter` в библиотеке ресурсов и выберите его для дальнейшего применения. Перетаскивая экземпляр объекта типа `GridSplitter` на монтажный стол, привяжите его к той строке или столбцу, размеры которых вам нужно сделать изменяемыми. Ради большей наглядности разделитель сетки (т.е. объект типа `GridSplitter`) показан на рис. 4.29 (а также на цветной вклейке к книге) немного утолщенным и заполненным голубым цветом, тогда как фон полотна (т.е. объекта типа `InkCanvas`) окрашен светлым оттенком оранжевого цвета.

Если вы запустите теперь свое приложение на выполнение, то сможете рисовать мышью на виртуальном полотне, доступном на первой вкладке, а также изменять размеры строк сетки, используя ее разделитель. В этом случае полотно (т.е. объект типа `InkCanvas`) будет изменяться по высоте, как показано на рис. 4.30 (а также на цветной вклейке к книге). Не так уж и плохо для всего лишь нескольких строк разметки!

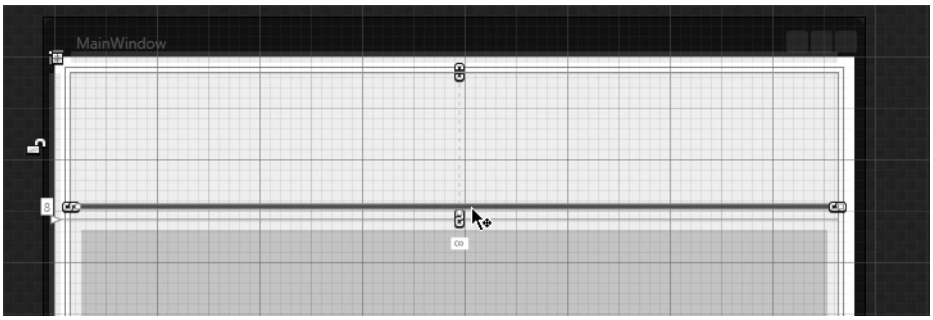


Рис. 4.29. Добавление разделителя к сетке

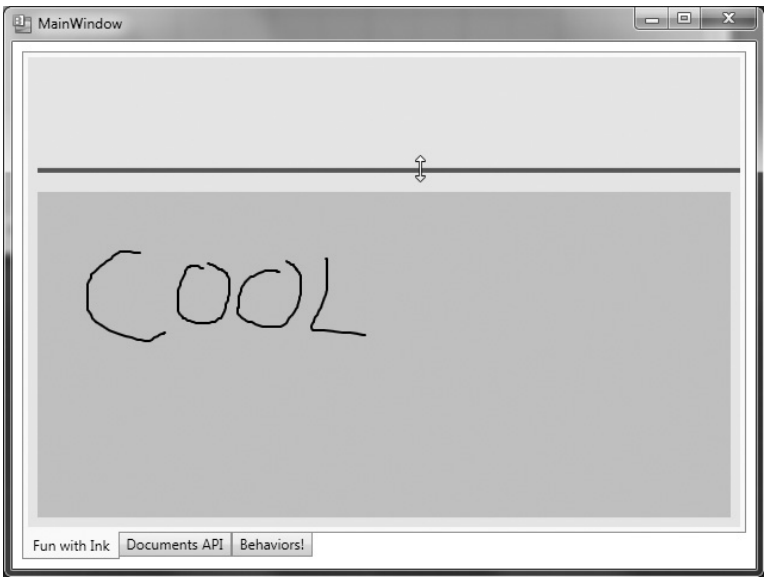


Рис. 4.30. Область виртуального полотна с изменяемыми размерами

Добавление вложенной блочной панели

В завершение компоновки первой вкладки вам предстоит ввести еще несколько элементов управления в верхней области сетки, чтобы пользователь мог изменять цвет обводки и размеры пера. С этой целью добавьте элемент управления типа `StackPanel` в верхней строке сетки, привязав его ко всем четырем сторонам, чтобы заполнить им всю эту область. Затем установите значение **`Horizontal`** в свойстве `Orientation` этого элемента управления и введите в него три объекта типа `Button`, присвоив им имена `btnRed`, `btnGreen` и `btnBlue` соответственно, а также элемент управления типа `TextBox`, назвав его `txtPenSize` (текстовое поле для ввода размера пера), вместе с объектом типа `Label`, содержащим текст метки, описывающей данное текстовое поле. В качестве примера на рис. 4.31 (а также на цветной вклейке к книге) приведен один из возможных вариантов компоновки элементов управления в верхней строке сетки. Элементы управления в этом варианте компоновки расставлены с определенным промежутком, для чего были специально настроены свойства `Margin` каждого из этих элементов на вложенной блочной панели.

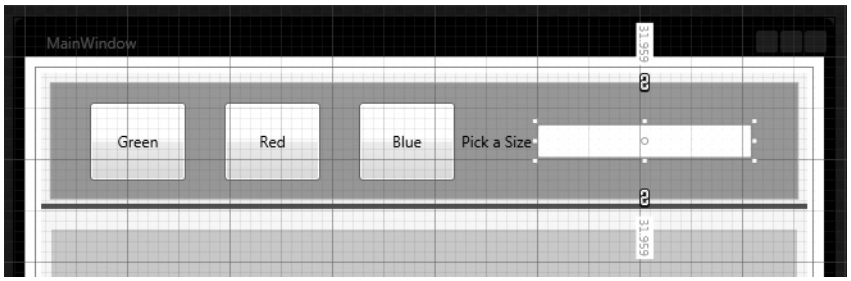


Рис. 4.31. Расположение элементов управления на вложенной блочной панели

Для того чтобы упростить процесс изменения цвета обводки, введите в поле свойства `Content` каждого объекта типа `Button` текст, обозначающий конкретный цвет: `Red` (Красный), **`Green`** (Зеленый) и **`Blue`** (Синий). Затем организуйте на панели Properties обработку события `Click`, наступающего после щелчка на каждом объекте типа `Button`, указав имя `ChangeColor` метода обработки этого события. Ниже приведен исходный код реализации данного метода.

```
private void ChangeColor(object sender, System.Windows.RoutedEventArgs e)
{
  // Получить строковое значение от выбранной щелчком кнопки.
  string colorToUse = ((Button)sender).Content.ToString();

  // Задать цвет, преобразовав строку в сплошной цвет.
  this.myInkArea.DefaultDrawingAttributes.Color =
  (Color)ColorConverter.ConvertFromString(colorToUse);
}
```

И наконец, организуйте на первой вкладке обработку события `LostFocus`, наступающего при потере логического фокуса на объекте типа `TextBox`. Реализуя метод обработки этого события в исходном коде, воспользуйтесь значением, введенным в текстовом поле объекта типа `TextBox`, чтобы задать высоту и ширину пера, как показано ниже.

```
private void txtPenSize_LostFocus(object sender,
System.Windows.RoutedEventArgs e)
{
  try
  {
    // Изменить высоту и ширину пера, исходя из данных,
    // введенных в текстовом поле.
    this.myInkArea.DefaultDrawingAttributes.Height =
    int.Parse(txtPenSize.Text);
    this.myInkArea.DefaultDrawingAttributes.Width =
    int.Parse(txtPenSize.Text);
  }
  catch
  {
    this.Title = "Bad Pen Size Value!";
  }
}
```

Запустите приложение на выполнение. Теперь можете задавать размеры пера и цвет обводки рисуемых штрихов, как показано на рис. 4.32, а также на цветной вклейке к книге.

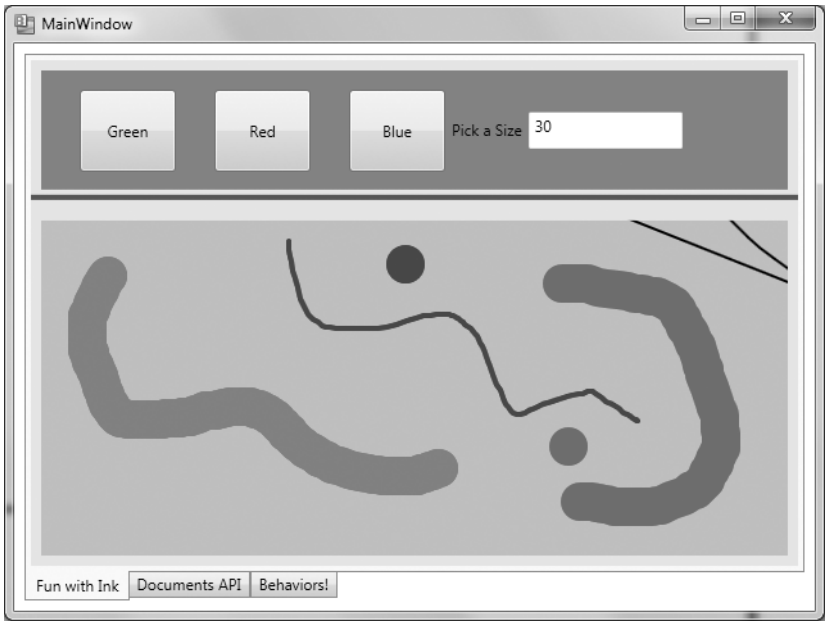


Рис. 4.32. Рисование на виртуальном полотне пером разного цвета и размеров

На этом компоновка первой вкладки пользовательского интерфейса рассматриваемого здесь примера проекта завершается. Надеюсь, теперь вы будете чувствовать себя более уверенно, обращаясь с объектами типа `Grid` и `StackPanel` и заполняя их контейнеры элементами управления. Попутно вы ознакомились поближе с моделью программирования, которая поддерживается в прикладном интерфейсе WPF Ink API[8]. А теперь перейдем к рассмотрению более развитых инструментов редактирования, с которыми вам, скорее всего, придется иметь дело при разработке приложений в среде Expression Blend IDE.

Введение в прикладной интерфейс WPF Document API

На первой вкладке рассматриваемого здесь примера проекта используются элементы управления, которые компонуются и настраиваются довольно просто. Даже если заполнить их специальным, составным содержимым, то это не намного усложнит их разработку в среде Expression Blend IDE. Но имеется целый ряд элементов управления, для разработки которых требуются намного более сложные и развитые редакторы, позволяющие раскрывать коллекции других объектов, присутствующие в подобных элементах управления. Здесь нет никакой практической возможности рассматривать редакторы *каждого* элемента управления *в отдельности*, поэтому функциональные возможности прикладного интерфейса WPF Document API[9] будут продемонстрированы на примере компоновки второй вкладки.

[8] Прикладной интерфейс Ink API обладает намного более богатым набором средств, чем продемонстрированные в этом простом примере. Если хотите освоить этот интерфейс более основательно, обратитесь за справкой к документации на .NET Framework и Silverlight.

[9] Напомним: на платформе Silverlight аналогичный прикладной интерфейс документов не поддерживается.

Для обращения с простыми фрагментами текста вполне пригодны такие элементы управления, доступные на платформе WPF, как, например, `Label`, `TextBox`, `TextBlock` и `PasswordBox`. Несмотря на всю их полезность, в некоторых приложениях WPF требуется применять сложные, тщательно отформатированные текстовые данные, аналогичные тем, которые можно обнаружить в документах формата Adobe PDF. Подобные функциональные возможности предоставляет прикладной интерфейс WPF Document API, хотя он и предполагает применение формата XPS (XML Paper Specification) вместо формата файлов PDF.

Используя различные классы из пространства имен `System.Windows.Documents` прикладного интерфейса WPF Document API, можно создать и подготовить к печати полноценный документ. Для этой цели в прикладном интерфейсе WPF Document API поддерживается целый ряд типов данных, обозначающих фрагменты документа в формате XPS, в том числе классы `List`, `Paragraph`, `Section`, `Table`, `LineBreak`, `Figure`, `Floater` и `Span`.

Блочные и внутристрочные элементы

Формально говоря, элементы, вводимые в документ формата XPS, относятся к одной из двух обширных категорий. К первой категории принадлежат *блочные элементы*, в том числе классы `List`, `Paragraph`, `BlockUIContainer`, `Section` и `Table`. Классы этой категории служат для группирования остального содержимого, например, списка, состоящего из разбитых на абзацы данных; абзаца, состоящего из подпунктов с отформатированным по-разному текстом, и т.д.

Ко второй категории относятся *внутристрочные элементы*, вкладываемые в блочные элементы или блоковые компоненты. К числу наиболее распространенных внутристрочных элементов относятся классы `Run`, `Span`, `LineBreak`, `Figure` и `Floater`.

Нетрудно догадаться, что эти классы носят названия терминов, используемых при создании форматированных документов в профессиональных редакторах. Как и все остальные элементы управления на платформе WPF, эти классы могут быть настроены в разметке XAML или в исходном коде. Следовательно, в разметке можно объявить пустой элемент `<Paragraph>`, заполняемый во время выполнения (в рассматриваемом здесь примере будет показано, как это делается).

Диспетчеры компоновки документов формата XPS

Как ни странно, но поместить внутристрочные и блочные элементы непосредственно в контейнер панели, например, типа `Grid`, нельзя. Вместо этого приходится вкладывать их в элемент разметки `<FlowDocument>` или `<FixedDocument>`.

Элемент типа `FlowDocument` идеально подходит для тех случаев, когда требуется предоставить пользователю возможность изменять порядок вывода данных на экран компьютерного монитора, включая масштабирование и форматирование текста, разбиение длинных страниц на две колонки и т.д. А элемент типа `FixedDocument` лучше всего подходит для представления подготовленных к печати и уже неизменяемых данных документа по принципу WYSIWYG (“что видишь на экране, то и получишь при печати”).

В рассматриваемом здесь примере проекта будет продемонстрировано применение только элемента типа `FlowDocument` для размещения внутристрочных и блочных элементов. Как только внутристрочные и блочные элементы будут введены в элемент типа `FlowDocument`, его можно поместить в один из четырех специализированных диспетчеров компоновки, поддерживающих формат XPS и перечисленных в табл. 4.2.

Таблица 4.2. Диспетчеры компоновки элементов управления в документах формата XPS

Диспетчер компоновки	Назначение
`FlowDocumentReader`	Отображает данные в элементе типа `FlowDocument` и обеспечивает поддержку масштабирования, поиска и компоновки содержимого в самых разных формах
`FlowDocumentScrollViewer`	Отображает данные в элементе типа `FlowDocument`, но сами данные представлены единым документом, просматриваемым с прокруткой. В этом диспетчере компоновки масштабирование, поиск или альтернативные режимы компоновки не поддерживаются
`RichTextBox`	Отображает данные в элементе типа `FlowDocument` и обеспечивает поддержку их редактирования пользователем
`FlowDocumentPageViewer`	Отображает документ только постранично. Допускает масштабирование, но не поиск данных

Наиболее полный в функциональном отношении способ отображения содержимого элемента типа `FlowDocument` состоит в том, чтобы поместить его в диспетчер компоновки типа `FlowDocumentReader`. В этом случае пользователь получает возможность изменять компоновку документа, искать в нем отдельные слова и масштабировать данные предоставляемыми для этих целей средствами пользовательского интерфейса. Единственное ограничение данного диспетчера компоновки, а также `FlowDocumentScrollViewer` и `FlowDocumentPageViewer` заключается в том, что данные в этих контейнерах отображаются только для чтения. Если же требуется предоставить пользователю возможность вводить новую информацию в элемент типа `FlowDocument`, то его следует поместить в диспетчер компоновки типа `RichTextBox`.

Создание панели инструментов

Щелкните на вкладке `Documents API` непосредственно на монтажном столе, чтобы активизировать ее для последующей правки (напомним, что это вторая вкладка в элементе управления типа `TabItem`). У вас уже должен быть диспетчер компоновки типа `Grid`, используемый по умолчанию в качестве прямого потомка элемента управления типа `TabItem`, но вы должны заменить его объектом типа `StackPanel`, перейдя к панели Objects and Timeline, щелкнув правой кнопкой мыши на заменяемом диспетчере компоновки и выбрав команду Change Layout Type из всплывающего контекстного меню.

Первым элементом управления, размещаемым на блочной панели типа `StackPanel`, должна стать специальная панель инструментов с двумя кнопками. Поэтому найдите элемент управления типа `ToolBar` в библиотеке ресурсов и добавьте его экземпляр в выбранный объект типа `StackPanel`. Доступный на платформе WPF элемент управления типа `ToolBar` может быть настроен таким образом, чтобы содержать любое количество элементов управления. Когда же потребуется добавить в него элемент средствами Expression Blend IDE, найдите его свойство `Items` (Элементы) и щелкните на кнопке с меткой ... (рис. 4.33).

Щелкните на кнопке Add another item (Добавить еще один элемент), расположенной в нижней части открывшегося диалогового окна, чтобы открыть другое диалоговое окно и найти в нем нужный элемент по имени. А справа от этой кнопки раскрывается список, из которого можно выбрать наиболее часто применяемые элементы управления (рис. 4.34).

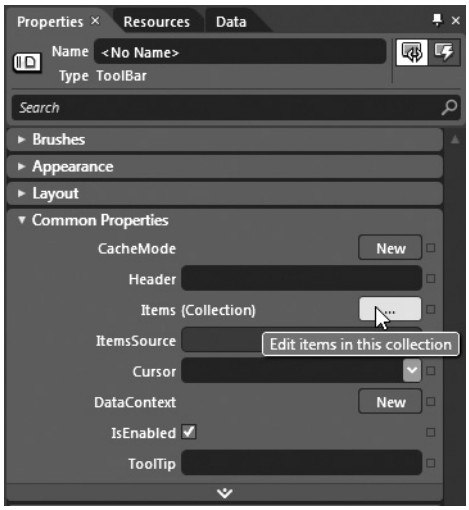


Рис. 4.33. Заполнение панели инструментов, представленной элементом управления типа `ToolBar`

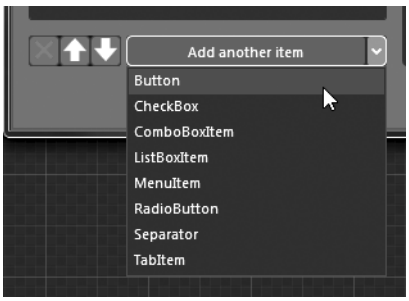


Рис. 4.34. Добавление объектов в коллекцию `Items` панели инструментов

Введите в открывшемся окне редактора коллекций два объекта типа `Button`. Сделав это, вы сразу же заметите, что каждый элемент может быть выбран и отредактирован в данном редакторе (рис. 4.35).

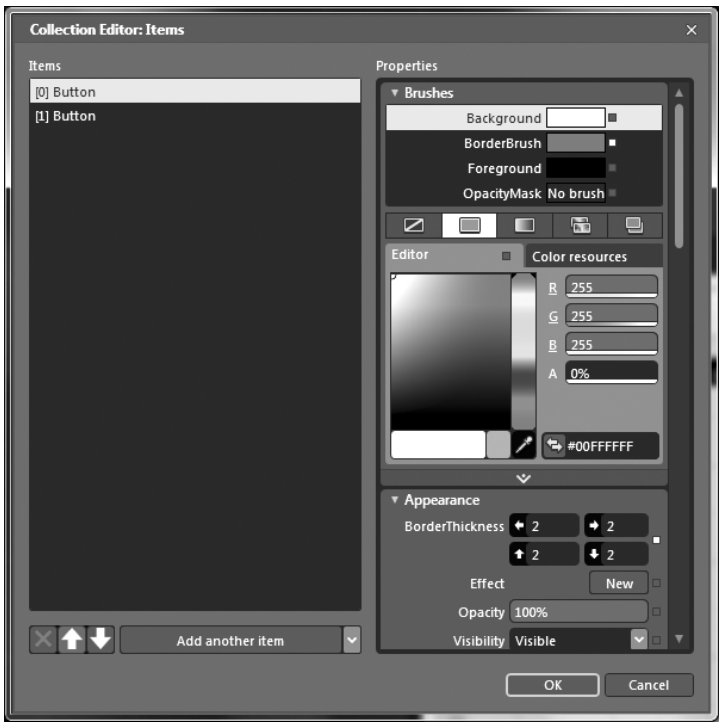


Рис. 4.35. Редактирование объектов из коллекции `Items` панели инструментов

Настройте свойства каждого объекта типа `Button` по своему усмотрению, но для рассматриваемого здесь примера в их свойствах `Content` должны быть установлены такие же значения, как и в приведенных ниже строках разметки XAML.

```
<Button BorderBrush="Green" Content="Save Doc"/>
<Button BorderBrush="Green" Content="Load Doc"/>
```

Вернитесь к визуальному конструктору, откройте библиотеку ресурсов и найдите элемент управления типа `FlowDocumentReader`. Разместите его на блочной панели типа `StackPanel`, присвойте ему имя `myDocumentReader` и растяните по всей поверхности блочной панели типа `StackPanel`. На данном этапе компоновка второй вкладки должна выглядеть так, как показано на рис. 4.36.

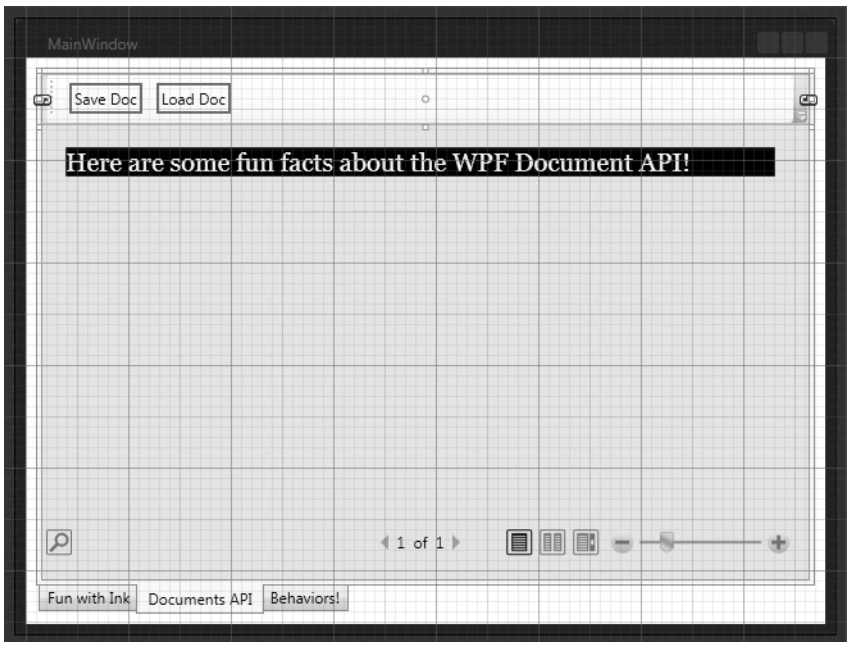


Рис. 4.36. Компоновка вкладки `Documents API`

Выберите элемент управления типа `FlowDocumentReader` на панели Objects and Timeline и перейдите к области Miscellaneous (Разное) на панели Properties. Щелкните на кнопке New рядом со свойством `Document`. В итоге разметка XAML обновится, и в ней появится пустой элемент `<FlowDocument>`:

```
<FlowDocumentReader x:Name="myDocumentReader" Height="269.4">
  <FlowDocument/>
</FlowDocumentReader>
```

В этом элементе можно теперь ввести классы форматирования документа, в том числе `List`, `Paragraph`, `Section`, `Table`, `LineBreak`, `Figure`, `Floater` и `Span`. Именно этим вам и придется заняться далее.

Заполнение элемента типа `FlowDocument`

Как только вы введете новый документ в рассматриваемый здесь контейнер документов, его свойство `Document` станет расширяемым, выявив массу новых свойств, предназначенных для оформления текстового документа. В данном примере

наибольший интерес представляет свойство `Blocks (Collection)` (Блоки/Коллекция), как показано на рис. 4.37.

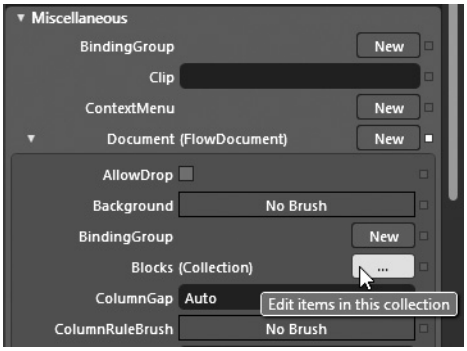


Рис. 4.37. Элемент типа `FlowDocument` может быть заполнен с помощью свойства `Blocks (Collection)`

Щелкните сначала на кнопке с меткой ... справа от данного свойства, а затем на кнопке Add another item в открывшемся диалоговом окне. Выберите классы блоков документа `List`, `Paragraph` и `Section`, чтобы ввести их в текущий контейнер, как показано на рис. 4.38.

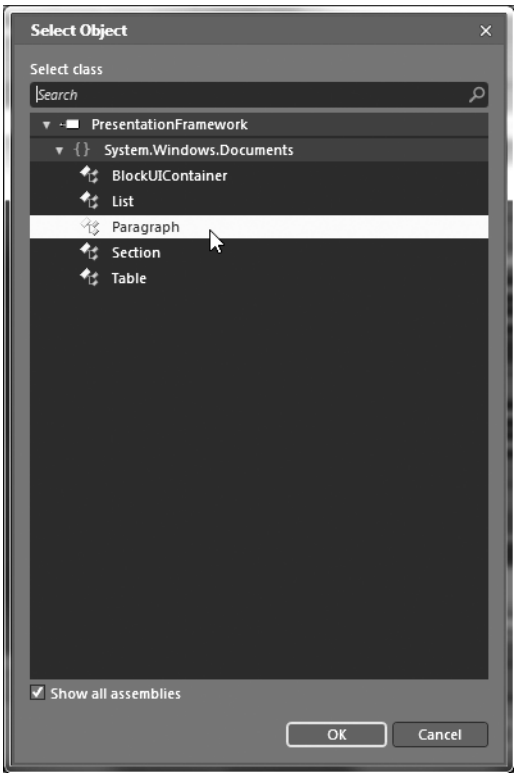


Рис. 4.38. Добавление в документ блоков, представленных отдельными классами

Каждый из блоков документа может быть отредактирован в редакторе блоков. Кроме того, каждый блок может содержать связанные с ним субблоки. Так, если выбрать блок типа `Section`, обозначающий раздел документа, то его можно дополнить субблоком типа `Paragraph`, представляющим абзац. В качестве примера на рис. 4.39 показан блок типа `Section`, настроенный на конкретный цвет фона, цвет переднего плана и размер шрифта и дополненный субблоком типа `Paragraph`.

Рис. 4.39. Настройка блоков документа

Настройте блок типа `Section` по своему усмотрению, но оставьте блок типа `List` и исходный блок типа `Paragraph` пустыми, поскольку манипулировать ими предстоит посредством кода. Ниже приведен один из возможных вариантов настройки элемента типа `FlowDocument` в диспетчере компоновки типа `FlowDocumentReader`.

```
<FlowDocumentReader x:Name="myDocumentReader"
                Height="339.4" Margin="0,0,8,0">
  <FlowDocument>
    <Section Foreground = "Yellow" Background = "Black">
      <Paragraph FontSize = "20">
        Here are some fun facts about the WPF Document API!
      </Paragraph>
    </Section>
    <List />
    <Paragraph />
  </FlowDocument>
</FlowDocumentReader>
```

Если вы запустите теперь свое приложение на выполнение, нажав функциональную клавишу <F5>, то должны уже иметь возможность изменять масштаб документа ползунковым регулятором в правом нижнем углу главного окна, производить поиск по ключевому слову, указываемому в поле Search слева внизу, а также отображать текстовые данные одним из трех способов с помощью соответствующих кнопок компоновки.

В качестве примера на рис. 4.40 показаны результаты поиска фрагмента текста “WPF” и увеличения масштаба документа.

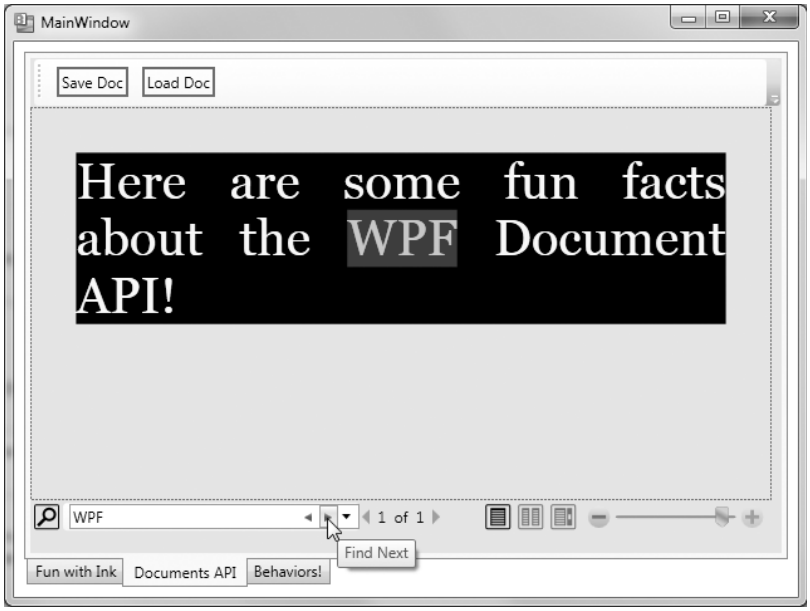


Рис. 4.40. Манипулирование элементом `FlowDocument` в диспетчере компоновки типа `FlowDocumentReader`

Прежде чем перейти к следующему этапу работы над текущим проектом, попробуйте отредактировать разметку XAML, чтобы воспользоваться другим контейнером для элемента типа `FlowDocument`, например `FlowDocumentScrollViewer` или `RichTextBox`, вместо диспетчера компоновки типа `FlowDocumentReader`. Сделав это, еще раз запустите приложение на выполнение и обратите внимание на другие способы и средства обработки текстовых данных в документе. После этого вернитесь к диспетчеру компоновки типа `FlowDocumentReader`.

Заполнение элемента типа `FlowDocument` непосредственно в коде

Эта книга посвящена главным образом инструментальным средствам разработки приложений в среде Expression Blend IDE, тем не менее вы должны знать, что элементами документа можно манипулировать и непосредственно в коде. В качестве упражнения вам предстоит построить в коде блок типа `List` и оставшийся блок типа `Paragraph`. Возможность манипулировать документами непосредственно в коде имеет, безусловно, большое значение, поскольку заполнять элемент типа `FlowDocument` придется с учетом информации, полученной от пользователя, из базы данных, внешних файлов и прочих ресурсов. Но прежде чем сделать это, следует воспользоваться встроенным в Expression Blend редактором XAML, чтобы присвоить блокам типа `List` и `Paragraph` подходящие имена для удобного обращения к ним в исходном коде, как показано ниже.

```
<List x:Name="listOfFunFacts"/>
<Paragraph x:Name="paraBodyText"/>
```

Итак, определите новый метод `PopulateDocument()` в файле исходного кода своего проекта. Сначала в этом методе в блок типа `List` будет добавлен ряд новых элементов списка, представленных объектами типа `ListItems`, причем каждый из них будет

состоять из блока типа Paragraph с одним объектом типа Run. Затем в этом вспомогательном методе с помощью трех отдельных объектов типа Run будет динамически отформатирован абзац. Ниже приведен исходный код данного метода, который можно скопировать и вставить из загружаемого архива примеров проектов к данной книге.

```
private void PopulateDocument()
{
  // Ввести данные в элемент списка.
  this.listOfFunFacts.FontSize = 14;
  this.listOfFunFacts.MarkerStyle = TextMarkerStyle.Circle;
  this.listOfFunFacts.ListItems.Add(new ListItem(new
    Paragraph(new Run("Fixed documents are for WYSIWYG print ready docs!"))));
  this.listOfFunFacts.ListItems.Add(new ListItem(
    new Paragraph(new Run("The API supports tables and embedded figures!"))));
  this.listOfFunFacts.ListItems.Add(new ListItem(
    new Paragraph(new Run("By default Flow documents are read only!"))));
  this.listOfFunFacts.ListItems.Add(new ListItem(
    new Paragraph(new Run
       ("BlockUIContainer allows you to embed WPF controls"))));

  // А теперь ввести данные в абзац Paragraph.
  // Первая часть абзаца с примером предложения.
  Run prefix = new Run("This paragraph was generated ");

  // Средняя часть абзаца.
  Bold b = new Bold();
  Run infix = new Run("dynamically");
  infix.Foreground = Brushes.Red;
  infix.FontSize = 30;
  b.Inlines.Add(infix);

  // Последняя часть абзаца.
  Run suffix = new Run(" at runtime!");

  // Далее ввести каждый фрагмент в коллекцию внутристрочных элементов абзаца.
  this.paraBodyText.Inlines.Add(prefix);
  this.paraBodyText.Inlines.Add(infix);
  this.paraBodyText.Inlines.Add(suffix);
}
```

Вызовите данный метод из конструктора главного окна приложения, как показано ниже

```
public MainWindow()
{
  this.InitializeComponent();
  // Ввести ниже код, выполняемый при создании объекта.
  PopulateDocument();
}
```

Введя приведенный выше исходный код, запустите приложение на выполнение и посмотрите, как новое содержимое документа формируется динамически.

Сохранение и загрузка данных документа

А теперь нужно разобраться с функциями двух кнопок, введенных на панели инструментов, созданной на вкладке Documents API. С этой целью воспользуйтесь панелью Properties, чтобы организовать обработку события Click, наступающего после щелчка на каждой из кнопок, представленных объектами типа Button, указав для каждой из них разные имена метода обработки данного события. Затем импортируйте в файл

исходного кода главного окна приложения (объекта типа `Window`) указанные ниже два пространства имен .NET, предоставляющих доступ к объектам ввода-вывода файлов, а также к объектам типа `XamlReader` и `XamlWriter`, которые потребуются для сохранения и загрузки данных документа.

```
using System.IO;
using System.Windows.Markup;
```

Для сохранения данных документа остается лишь создать XAML-файл, в котором будет храниться содержимое документа. А разметка, описывающая сами данные, предоставляется в свойстве `Document` диспетчера компоновки типа `FlowDocumentReader`, как показано в приведенном ниже исходном коде метода обработки событий `Click`, наступающих после щелчка на кнопке `Save Doc` (Сохранить документ).

```
private void btnSaveDoc_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
  using(FileStream fStream =
    File.Open("documentData.xaml", FileMode.Create))
  {
    XamlWriter.Save(this.myDocumentReader.Document, fStream);
  }
}
```

Загрузка документа выполняется так же просто. Для этого достаточно изменить основную операцию на обратную, как показано в приведенном ниже исходном коде метода обработки событий `Click`, наступающих при выборе кнопки `Load Doc` (Загрузить документ).

```
private void btnLoadDoc_Click(object sender, System.Windows.RoutedEventArgs e)
{
  using(FileStream fStream = File.Open("documentData.xaml", FileMode.Open))
  {
    try
    {
       FlowDocument doc = XamlReader.Load(fStream) as FlowDocument;
       this.myDocumentReader.Document = doc;
    }
    catch(Exception ex) {MessageBox.Show(ex.Message, "Error Loading Doc!");}
  }
}
```

Если вы запустите теперь свое приложение на выполнение и щелкнете на кнопке `Save Doc`, то XAML-документ будет сохранен в каталоге `\bin\Debug` текущего проекта. А если щелкнете на кнопке `Load Doc`, то те же самые данные будут считаны из сохраненного прежде XAML-файла, чтобы заполнить ими текущий документ.

На этом компоновка второй вкладки и введение в прикладной интерфейс WPF Document API завершается. И в заключение назначение объектов поведения будет рассмотрено на примере компоновки третьей и последней вкладки пользовательского интерфейса рассматриваемого здесь проекта.

Введение в объекты поведения, применяемые в Expression Blend

Любое приложение на платформе Silverlight или WPF представляет собой полнофункциональное сочетание разметки XAML, описывающей внешний вид приложения, и кода, определяющего его функциональные возможности. Конкретный код, который

приходится писать вручную, зависит от характера разрабатываемого проекта. Так, в процессе разработки проекта может возникнуть потребность в написании кода для установления связи с удаленной WCF-службой[10], чтения данных из реляционной базы данных, формирования динамического содержимого во время выполнения и т.п. И хотя базовый код каждого приложения по-своему уникален, существует целый ряд ситуаций, типичных для многих приложений, в которых интенсивно используется графика.

В частности, во многих приложениях на платформах WPF и Silverlight требуется предоставлять пользователям возможность переставлять отдельные элементы интерфейса методом перетаскивания или же воспроизводить записи из звуковых файлов в самых разных ситуациях, в том числе при выборе пункта меню, щелчком на графическом элементе и т.д. Допустим, вы создаете пользовательский интерфейс, в котором применяются операции привязки данных, подробно рассматриваемые в главе 6, и вам требуется воспроизводить разные виды анимации по раскадровке при обновлении источника данных. Для решения этой задача вы могли бы, конечно, написать специальный код на C# или VB, но в состав Expression Blend SDK входят разнообразные готовые *объекты поведения*, способные решить подобные задачи программирования в типичных ситуациях.

Объекты поведения представляют собой обычные компоненты, которые могут быть добавлены в иерархическое представление разметки XAML для манипулирования другими, связанными с ними элементами пользовательского интерфейса. По своему внутреннему устройству объекты поведения являются подлинными классами, определенными в библиотеке .NET, а это означает, что любое поведение можно приводить в действие непосредственно в коде[11].

Большинство встроенных объектов поведения включены в состав библиотеки `Microsoft.Expression.Interactivity.dll` и автоматически вводятся в проекты, разрабатываемые в Expression Blend, по ссылке на эту библиотеку, когда требуется воспользоваться ими. (Если же вы хотите воспользоваться объектами поведения в среде Visual Studio 2010, вам придется сделать ссылку на данную библиотеку вручную.) В табл. 4.3 перечислены некоторые, но не все объекты поведения, входящие в состав Expression Blend SDK и сгруппированные по сходным выполняемым функциям.

Таблица 4.3. Некоторые объекты поведения, входящие в состав Expression Blend SDK

Объект поведения	Назначение
`FluidMoveBehavior`	Действует вместе с несколькими связанными с ним объектами и позволяет управлять анимацией и эффектами переходов между элементами пользовательского интерфейса, что иначе называется *плавным поведением*
`ControlStoryboardAction`	Позволяет начинать, останавливать или приостанавливать анимацию по раскадровке, не прибегая к написанию процедурного кода, как демонстрировалось в главе 3

`CallMethodAction`
`ChangePropertyAction`
`InvokeCommandAction`
	Изменяют свойство объекта, вызывают метод для объекта или инициируют выполнение команды в разметке XAML[12]

[10] Windows Communication Foundation (WCF) — это прикладной интерфейс .NET API, позволяющий дистанционно вызывать методы, используя разные сетевые протоколы, в том числе HTTP, TCP, именованные каналы и т.д.

[11] Имеется также возможность создавать классы поведения, но данный вопрос выходит за рамки рассмотрения этой книги.

[12] Объекты команд относятся к тем классам, в которых реализуется интерфейс ICommand. Они могут использоваться для внедрения в приложение часто выполняемых пользователями команд, в том числе копирования, вставки и вырезания, без написания большого объема кода.

Окончание табл. 4.3

Объект поведения	Назначение
`GoToStateAction`	Позволяет переходить в новое состояние, определяемое диспетчером визуальных состояний (Visual State Manager), что характерно для построения специальных шаблонов, рассматриваемых в главе 5
`PlaySoundAction`	Определяет различные условия воспроизведения звуковой записи
`DataStateBahavior` `SetDataStoreValueAction`	Определяет порядок применения визуальных эффектов при выполнении операций привязки данных, подробно рассматриваемых в главе 6
`MouseDragElementBehavior`	Дает пользователю возможность перемещать элемент, содержащийся в диспетчере компоновки
`ActivateStateAction` `NavigateToScreenAction` `NavigateBackAction` `NavigateForwardAction`	Позволяют, наряду с другими объектами поведения, определить навигационную структуру при построении прототипа пользовательского интерфейса с помощью компонента SketchFlow среды Expression Blend, как поясняется в главе 8

Напомним, что каждый из перечисленных выше объектов поведения преобразуется в тип класса, и поэтому в каждом таком объекте поддерживается целый ряд свойств, методов и событий, многие из которых могут быть настроены на панели Properties аналогично элементам управления пользовательского интерфейса. Далее в главе будет показано, каким образом действуют различные объекты поведения, а полное их описание можно найти в справочной системе Expression Blend SDK for WPF User Guide (Руководство пользователя набором инструментальных средств Expression Blend SDK для платформы WPF), доступной в меню Help среды Expression Blend.

Примечание. Справочная система Expression Blend SDK for WPF User Guide отличается от справочной системы Expression Blend User Guide, и поэтому их не следует путать!

Открыв эту справочную систему, вы найдете подробное описание каждого из объектов поведения, а также примеры их настройки (рис. 4.41).

Как и остальные компоненты Expression Blend IDE, объекты поведения можно обнаружить в библиотеке ресурсов. На рис. 4.42 показано содержимое категории Behaviors из этой библиотеки на панели Assets.

Объект поведения типа `MouseDragElementBehavior`

В завершение рассматриваемого здесь примера проекта будет показано, насколько полезными могут оказаться объекты поведения. С этой целью выберите третью вкладку `Behaviors!` на монтажном столе для последующей правки. Затем создайте на монтажном столе любую геометрическую форму, например шестиугольную, доступную в категории Shapes библиотеки ресурсов, настроив ее основные свойства (цвета и прочее). Далее выберите компонент `MouseDragElementBehavior` в библиотеке ресурсов и перетащите его на вновь созданную геометрическую форму.

Примечание. Пиктограмма библиотеки ресурсов со знаком **>>** исчезнет с панели Tools, как только выбранный в этой библиотеке элемент появится в нижней части панели Tools. Как упоминалось ранее, пользоваться панелью Assets удобнее, поскольку она остается на экране после ее выбора. Кроме того, панель Assets значительно упрощает работу с объектами поведения, которые обычно переносятся на монтажный стол методом перетаскивания.

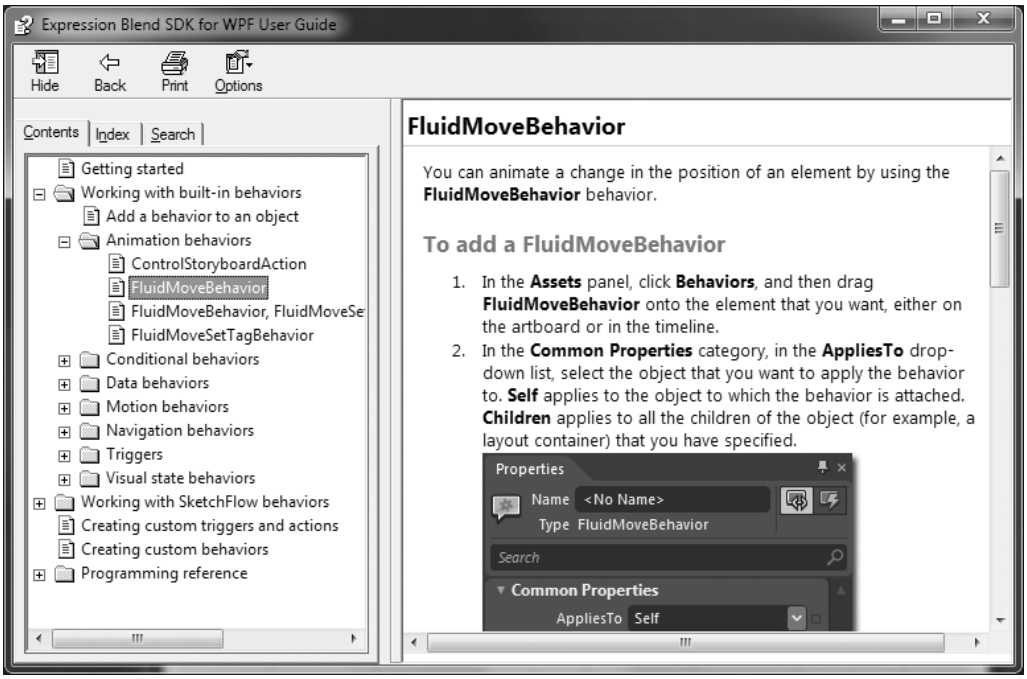


Рис. 4.41. Все объекты поведения подробно описываются в справочной системе Expression Blend SDK for WPF User Guide

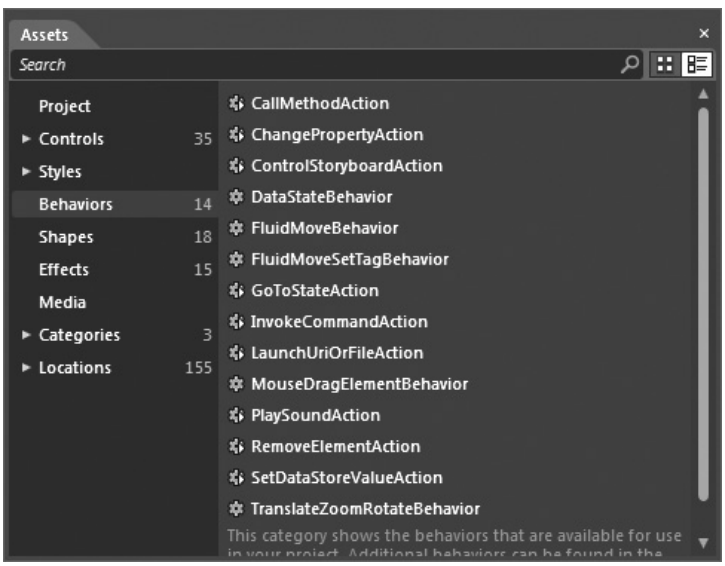


Рис. 4.42. Объекты поведения можно обнаружить в библиотеке ресурсов

Как подразумевает название объекта поведения типа `MouseDragElementBehavior`, его можно использовать для упрощения реализации в приложении функциональных возможностей перетаскивания элементов пользовательского интерфейса, *не* прибегая

к обработке целого ряда событий от мыши, расчетам проверок попадания курсора или написанию сложного кода для перестановки элемента в родительском контейнере. Проанализировав содержимое панели Objects and Timeline, вы обнаружите, что компонент `MouseDragElementBehavior` является потомком целевой геометрической формы.

Итак, выберите узел этого объекта поведения и проанализируйте содержимое панели Properties. У данного объекта поведения имеется единственное свойство `ConstrainToParentBounds` (Соблюдение родительских рамок). Если установить флажок этого свойства, как показано на рис. 4.43, то данный объект поведения автоматически примет к сведению, что перетаскиваемый элемент не может быть перемещен за пределы своего родительского диспетчера компоновки.

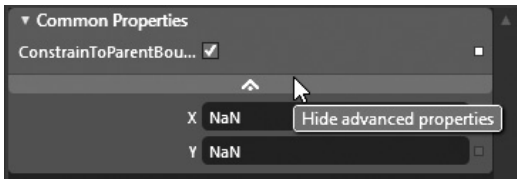


Рис. 4.43. У каждого объекта поведения свои особые настраиваемые свойства

Запустите свое приложение на выполнение и проверьте, сможете ли вы перемещать геометрическую форму по сетке. Нетрудно себе представить, насколько полезным может оказаться такой объект поведения в самых разных проектах. В частности, на платформе Silverlight можно построить приложение электронной коммерции и воспользоваться данным объектом поведения, чтобы предоставить пользователю возможность перетаскивать закупаемые товары в тележку для покупок с автоматическим подсчетом их стоимости. Этот объект можно также использовать при создании интерактивных видеоигр, мультимедийных и прочих приложений.

В отношении данного конкретного объекта поведения необходимо сделать следующую оговорку: его нельзя применять к объектам, поддерживающим событие типа `Click`. Так, если вам требуется предоставить пользователю возможность перетаскивать элемент интерфейса, например, кнопку, представленную объектом типа `Button`, для решения этой задачи вам придется приложить дополнительные усилия. В частности, придется заключить этот элемент пользовательского интерфейса в объект типа `Border`, а затем применить компонент `MouseDragElementBehavior` *именно* к объекту типа `Border`, а *не* к его порожденному элементу управления.

Для того чтобы убедиться в этом, добавьте объект типа `Button` на монтажном столе, щелкните на нем правой кнопкой мыши и выберите команду Group Into из всплывающего контекстного меню, чтобы поместить этот элемент пользовательского интерфейса в контейнер элемента управления типа `Border` (рис. 4.44). Затем перейдите к панели Properties, установите значение 3 в свойстве `BorderThickness` (Толщина границы) по всем четырем сторонам и выберите особый цвет для свойства `BorderBrush` (Кисть для обводки границы). Как только вы добавите новый компонент `MouseDragElementBehavior` в контейнер элемента управления типа `Border`, то сможете перемещать кнопку по сетке во время выполнения своего приложения (рис. 4.45).

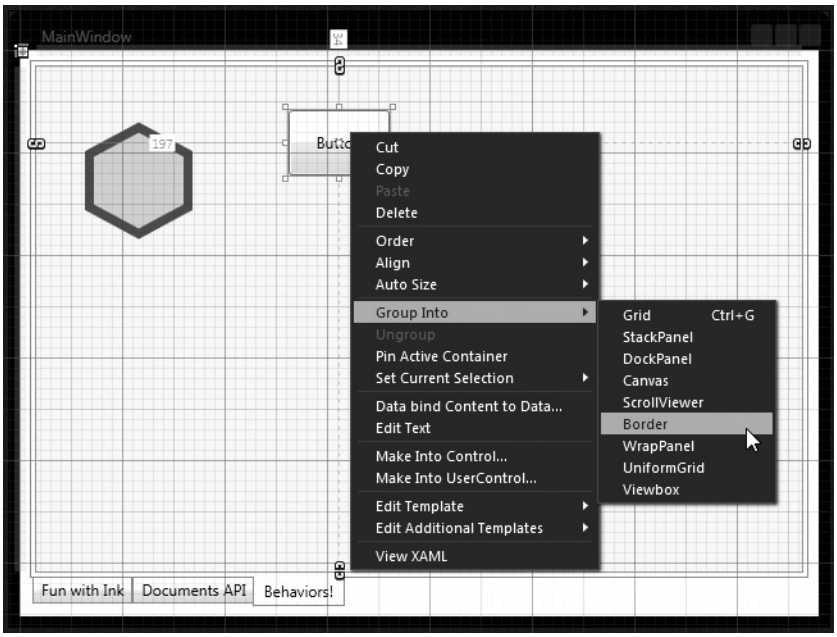


Рис. 4.44. Для того чтобы выбираемые щелчком элементы пользовательского интерфейса можно было перетаскивать, их нужно заключить в контейнер элемента управления типа `Border`

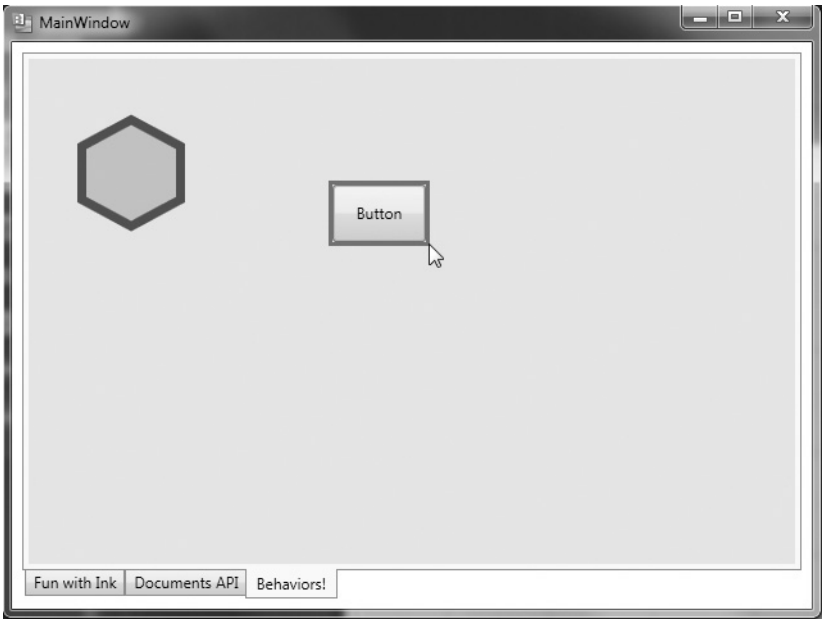


Рис. 4.45. Перемещение элементов пользовательского интерфейса, вообще не требующее написания кода!

Элемент управления типа `Border` можно также настроить на отображение пиктограммы руки, появляющейся при наведении курсора мыши на этот элемент. Тем самым пользователю дается ясно понять, что объект типа `Button`, содержащийся в данном элементе управления, можно переместить. Для этой цели достаточно выбрать в свойстве `Cursor` элемента управления типа `Border` один из возможных вариантов отображения курсора.

Исходный код. Исходный код примера проекта `WpfControlsApp` находится в папке `Ch 4 Code` загружаемого архива примеров проектов к данной книге.

На этом завершается рассмотрение средств и способов построения в среде Expression Blend IDE систем компоновки элементов пользовательского интерфейса, а также внедрения элементов управления и объектов поведения в приложения. Применение других объектов поведения и элементов управления на практике будет продемонстрировано в последующих главах, а до тех пор вы, надеюсь, почувствуете себя более уверенно, пользуясь средствами Expression Blend для разработки графического пользовательского интерфейса своих приложений.

Резюме

В этой главе были представлены две модели программирования на платформах WPF и Silverlight: модель содержимого элементов управления и модель содержимого многокомпонентных элементов управления. Как следует из данной главы, обе модели позволяют создавать элементы пользовательского интерфейса, содержащие специально настраиваемые данные.

В первых примерах проектов, рассматривавшихся в этой главе, было продемонстрировано применение нескольких диспетчеров компоновки. Затем были подробно рассмотрены инструментальные средства Expression Blend IDE, упрощающие построение компоновок со сложным содержимым. В частности, на монтажном столе можно построить объект типа `Grid` и расположить по строкам и столбцам его сетки самые разные элементы пользовательского интерфейса. Кроме того, были продемонстрированы различные способы группирования выбранных элементов в новые диспетчеры компоновки, показано, каким образом диспетчеры компоновки настраиваются на панели Objects and Timeline, а также разъяснено назначение элемента управления типа `GridSplitter`.

В остальной части главы был рассмотрен пример построения приложения WPF, на котором было продемонстрировано применение целого ряда новых элементов управления (`TabControl` “со товарищи”), а также ряда полезных прикладных интерфейсов, в том числе Ink API для фиксации данных, вводимых мышью, пером и касанием сенсорного экрана и WPF Document API для построения довольно сложных документов, подготавливаемых к печати в формате, аналогичном Adobe PDF.

И в заключение были рассмотрены не менее важные вопросы применения объектов поведения. Эти компоненты Expression Blend позволяют быстро внедрять типичные виды поведения во время выполнения приложений практически без написания процедурного кода. Здесь были продемонстрированы функциональные возможности объекта поведения типа `MouseDragElementBehavior`, а другие объекты данной категории будут рассмотрены в последующих главах.

ГЛАВА 5

Стили, шаблоны и классы UserControl

В предыдущей главе было показано, каким образом пользовательский интерфейс создается средствами Expression Blend IDE и библиотек элементов управления на платформах WPF и Silverlight. В настоящей главе рассматривается целый ряд взаимосвязанных вопросов, разъяснение которых позволяет глубже понять, каким образом элементы управления специально настраиваются на окончательный визуальный вывод. Сначала будет рассмотрен механизм стилевого оформления приложений на платформах WPF и Silverlight. Вам, возможно, известно, что стили оформления служат той же цели, что и вложенные таблицы стилей, применяемые в разработке веб-приложений: и те и другие обеспечивают общее стилевое оформление (шрифты, цвета, размеры и пр.) всех экземпляров отдельного элемента управления (типа `Button`, `TextBox` и т.д.).

Далее в главе рассматривается назначение шаблонов элементов управления. Как будет показано в ней, каждый элемент управления на платформах WPF и Silverlight содержит стандартный набор инструкций визуализации, называемый *используемым по умолчанию шаблоном* и применяемый для визуализации готового вывода. Но вы вольны видоизменить или полностью заменить этот шаблон специально подобранным набором инструкций визуализации. Используя шаблоны, можно коренным образом изменить порядок визуализации элементов управления, причем сделать это, вообще не прибегая к написанию процедурного кода. Попутно вы ознакомитесь с применением триггеров и диспетчера визуальных состояний (VSM — Visual State Manager) для определения условий изменения внешнего вида шаблона при переходе в разные визуальные состояния, когда, например, он получает логический фокус, курсор мыши находится в его пределах, на нем щелкают кнопкой мыши и т.д.

И в заключение главы будет рассмотрено назначение специальных классов `UserControl`, с помощью которых можно создавать новые элементы управления на платформах WPF и Silverlight из уже имеющихся элементов пользовательского интерфейса. И нет ничего удивительного в том, что для полноты реализации в специальных классах `UserControl` также применяются стили, шаблоны, триггеры и диспетчер VSM.

Примечание. Для создания стилей оформления, шаблонов и классов `UserControl` нужно свободно разбираться в таких вопросах, как графика, анимация и ресурсы объектов (см. главы 2 и 3).

Назначение стилей оформления

При построении пользовательского интерфейса приложения на платформе WPF или Silverlight зачастую требуется, чтобы у однотипных элементов управления был какой-то

общий внешний вид. В частности, все кнопки, представленные объектами типа Button, должны иметь одинаковую высоту, ширину, цвет фона и размер шрифта текстовых надписей. Если все элементы пользовательского интерфейса находятся в одном контейнере, например в объекте типа Window или UserControl, то их внешний вид можно быстро определить, выбрав их на монтажном столе (щелчком на каждом из них по очереди при одновременно нажатой клавише <Ctrl>) и настроив их общие свойства на панели Properties.

Подобным способом можно, конечно, одинаково настроить отдельные свойства объектов типа Button, но труднее будет вносить изменения впоследствии, когда настройку целого ряда свойств многих объектов придется начинать с начала при каждом изменении[1]. Недостаток такого подхода становится совершенно очевидным после установки в некоторых свойствах таких сложных объектов, как специальные кисти. Достаточно, например, представить, насколько усложнится дело, если потребуется размножить подобные установки на десять разных элементов управления типа Button в многооконном приложении WPF, как показано в приведенном ниже фрагменте разметки.

```
<!-- Слишком много повторяющейся разметки! -->
<Button Content="OK" Width="100" Height="30"
       FontFamily="Arial Rounded MT Bold" FontSize="16">
  <Button.Background>
    <LinearGradientBrush EndPoint="0,1" StartPoint="0,0">
      <GradientStop Color="#FFF3F3F3" Offset="0"/>
      <GradientStop Color="#FFEBEBEB" Offset="0.5"/>
      <GradientStop Color="#FFDDDDDD" Offset="0.5"/>
      <GradientStop Color="#FF3620CE" Offset="1"/>
    </LinearGradientBrush>
  </Button.Background>
</Button>
```

Правда, на обеих платформах, WPF и Silverlight, предоставляется довольно простой способ соблюдения внешнего вида связанных вместе элементов управления с помощью стилей оформления. Проще говоря, *стиль* представляет собой объект, содержащий совокупность пар "свойство–значение". С точки зрения программирования отдельный стиль оформления представлен классом System.Windows.Style. У этого класса имеется свойство Setters (Установщики), раскрывающее одноименную строго типизированную коллекцию объектов типа Setter. С помощью этих объектов и определяются пары "свойство–значение".

Помимо коллекции Setters, в классе Style определен также ряд других важных компонентов, позволяющих накладывать определенные ограничения на применение стиля и даже создавать новый стиль из уже существующего (это своего рода наследование стилей). Как и следовало ожидать, в среде Expression Blend IDE предоставляются средства, помогающие автоматизировать процесс создания и редактирования стилей оформления. Но прежде чем рассматривать эти вспомогательные средства, покажем на небольшом примере, каким образом самый элементарный стиль создается вручную.

Создание простейшего стиля вручную

Если хотите последовать рассматриваемому здесь примеру, запустите Expression Blend на выполнение и создайте новый проект приложения WPF, присвоив ему имя

[1] Данный способ малопригоден и в том случае, если настраиваемые элементы управления находятся в разных местах, например, в пяти разных окнах (объектах типа Window) приложения WPF.

WpfStyleByHand[2]. Созданный вручную стиль можно, конечно, встроить непосредственно в элемент управления, но зачастую объект типа Style включается в компоновку элемента управления как ресурс объекта (см. главу 2). Подобно любому другому ресурсу объекта, стиль можно внедрить на уровне объекта типа Window или UserControl, на уровне приложения (в файл разметки приложения App.xaml), а также ввести его в специальный словарь ресурсов, что очень удобно, поскольку делает объекты типа Style легко доступными во многих проектах.

Напомним, что конечная цель создания стиля — определить объект типа Style, как минимум, заполняющий коллекцию Setters парами "свойство–значение". В данном примере будет создан стиль, в котором определяются основные характеристики шрифтового оформления отдельного элемента управления из разрабатываемого приложения. С этой целью откройте файл разметки приложения App.xaml и введите приведенное ниже описание стиля, определяемого ключом BasicControlStyle (Элементарный стиль оформления элемента управления), в редакторе XAML, интегрированном в среду Expression Blend IDE[3].

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

  <!-- Это очень простой стиль, применяемый к любому элементу управления. -->
  <Application.Resources>
    <Style x:Key ="BasicControlStyle">
      <Setter Property = "Control.FontSize" Value ="14"/>
      <Setter Property = "Control.Height" Value = "40"/>
      <Setter Property = "Control.Cursor" Value = "Hand"/>
    </Style>
  </Application.Resources>
</Application>
```

Как видите, в этом стиле три объекта типа Setter вводятся во внутреннюю коллекцию Setters. В данном примере стиль обеспечивает принятие элементом управления следующего внешнего вида: размер шрифта — **14** пунктов, высота — **40** пикселей и отображение пиктограммы руки, когда курсор мыши находится в пределах данного элемента управления. Этот стиль будет далее применен к нескольким элементам управления на монтажном столе.

Присваивание стиля свойству Style элемента управления

Откройте монтажный стол исходного окна, представленного объектом типа Window, и разместите элементы управления типа Label и Button в любом месте в диспетчере компоновки типа Grid. Введите в свойстве Content каждого из этих элементов управления свой особый текст, как в приведенном ниже примере разметки.

---

[2] Разметка для этого примера приложения будет похожей, но не точно такой же, как и для приложения Silverlight. Следовательно, разметка XAML, используемая для стилей оформления приложений WPF и Silverlight, не является совершенно одинаковой. Правда, при создании стилей в редакторах Expression Blend соответствующая разметка XAML формируется автоматически в зависимости от выбранной платформы и ее прикладного интерфейса API.

[3] Если вы создаете данный пример проекта на платформе Silverlight, добавьте атрибут TargetType="Control" в элемент <Style>, который открывает объявление стиля. Подробнее об этом речь пойдет ниже, в разделе "Ограничение стиля типом целевого объекта".

```
<Grid x:Name="LayoutRoot">
  <Button Content="My Button" ... />
  <Label Content="Some Simple Text" ... />
</Grid>
```

Далее выберите оба элемента управления на монтажном столе, нажав клавишу <Ctrl> и щелкнув на них по очереди, и найдите свойство `Style` в категории Miscellaneous на панели Properties. (Напомним: вы всегда сможете воспользоваться полем Search на панели Properties для быстрого поиска нужного свойства или события.) Найдя свойство `Style`, щелкните на кнопке Advanced options с изображением маленького квадратика справа от текстового поля свойства, и найдите ресурс `BasicControlStyle`, применяемый на уровне приложения (рис. 5.1).

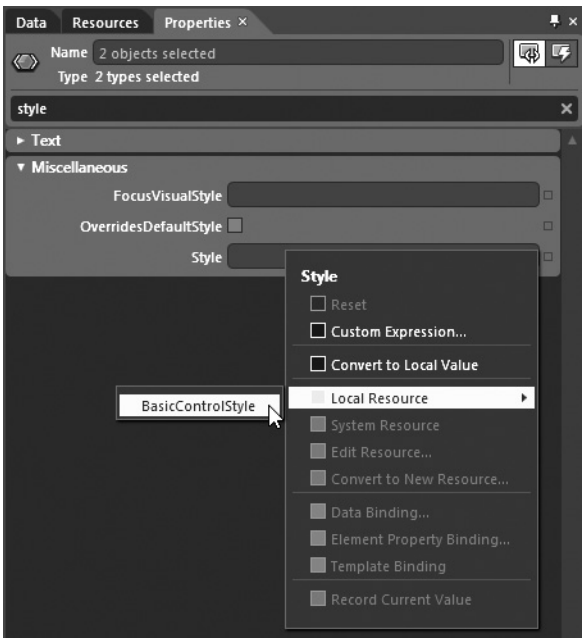


Рис. 5.1. Свойство `Style` находится в области Miscellaneous на панели Properties, а нужный стиль выбирается как ресурс, применяемый на уровне приложения

Как только вы выберете ресурс `BasicControlStyle`, оба элемента управления соответственно обновятся на монтажном столе. Если вы запустите теперь свое приложение на выполнение, то заметите, что курсор мыши изменяет свой вид на изображение руки, когда оказывается в пределах любого из двух элементов управления. А если вы проанализируете автоматически сформированную разметку, то обнаружите, что выбранный вами стиль задается в свойстве `Style` посредством расширения разметки `{DynamicResource}`[4], тогда как в приложениях Silverlight для этой же цели служит расширение разметки `{StaticResource}`.

```
<Grid x:Name="LayoutRoot">
  <Button Content="My Button" ...
```

[4] Расширение разметки `{DynamicResource}` гарантирует, что если ресурс изменяется во время выполнения, то объекты, использующие его, автоматически обновляются. Напомним также, что расширение разметки `{DynamicResource}` поддерживается только на платформе WPF.

```
        Style="{DynamicResource BasicControlStyle}"/>
  <Label Content="Some Simple Text" ...
        Style="{DynamicResource BasicControlStyle}"/>
</Grid>
```

Уделите немного времени просмотру свойства `Style` и на панели Properties. Обратите внимание на то, что это свойство теперь обрамлено зеленой ограничивающей рамкой, как показано на рис. 5.2, а также на цветной вклейке к книге. По принятому соглашению все свойства, которым присвоен именованный ресурс объекта (в данном случае — стиль оформления), выделяются подобным образом для удобного и наглядного напоминания о данном обстоятельстве в процессе разработки проекта.

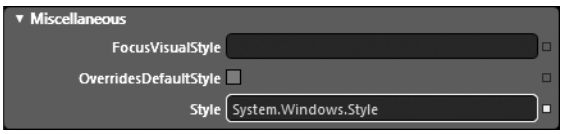


Рис. 5.2. Свойства, связанные с ресурсами объектов, обрамляются зеленой ограничивающей рамкой

Переопределение установок стиля

В настоящий момент в обоих элементах управления, `Button` и `Label`, действуют ограничения, накладываемые применяемым в них стилем. Разумеется, если после применения стиля в элементе управления требуется изменить некоторые его настройки, то к этому не должно быть никаких препятствий. В частности, стиль элемента управления типа `Button` можно обновить, с тем чтобы изменить в нем вид курсора с изображения руки на изображение знака вопроса, обозначающего справку, как показано ниже.

```
<Button Content="My Button" Cursor="Help" ...
      Style="{DynamicResource BasicControlStyle}"/>
```

Однако стили обрабатываются перед установками отдельных свойств элементов управления с помощью стилей. Поэтому эти установки могут время от времени переопределяться в элементах управления.

Ограничение стиля типом целевого объекта

В настоящий момент рассматриваемый здесь простейший стиль определен таким образом, чтобы его можно было применить к *любому* элементу управления, установив соответственно его свойство `Style`. И происходит это потому, что каждое свойство, указываемое в элементе разметки `<Style>`, определено в классе `Control`, как в приведенном ниже примере[5].

```
<Setter Property = "Control.Height" Value = "40"/>
```

Для стиля, в котором определяются десятки установок, подобный подход влечет за собой немалый объем повторяющейся разметки. Один из способов упростить такой стиль состоит в применении атрибута `TargetType`. Добавив этот атрибут в начальном дескрипторе элемента разметки `<Style>`, можно явно указать, где именно должен применяться данный стиль. В качестве примера ниже приведен переделанный вариант разметки, описывающей рассматриваемый здесь стиль `BasicControlStyle`.

[5] Класс `Control` является общим родительским классом для всех элементов графического пользовательского интерфейса. Это справедливо для приложений на обеих платформах, WPF и Silverlight.

```
<Style x:Key ="BasicControlStyle" TargetType="Control">
  <Setter Property = "FontSize" Value ="14"/>
  <Setter Property = "Height" Value = "40"/>
  <Setter Property = "Cursor" Value = "Hand"/>
</Style>
```

Примечание. На платформе Silverlight стили всегда должны создаваться с использованием атрибута `TargetType`. И только на платформе WPF допускается указывать имя класса перед именем свойства в элементе разметки `<Setter>`.

В какой-то степени это выход из положения, но у нас по-прежнему остается стиль, который можно применять к *любому* элементу управления. А использование атрибута `TargetType` приносит большую пользу в том случае, если действительно требуется определить стиль, который может быть применен к элементу управления конкретного типа. В качестве упражнения попробуйте ввести новый стиль под названием `BigGreenButton` (Крупная зеленая кнопка) в словарь ресурсов текущего проекта. Ниже приведена разметка, описывающая этот стиль.

```
<!-- Этот стиль можно применять только к элементам управления типа Button -->
<Style x:Key ="BigGreenButton" TargetType="Button">
  <Setter Property = "FontSize" Value ="20"/>
  <Setter Property = "Height" Value = "100"/>
  <Setter Property = "Width" Value = "100"/>
  <Setter Property = "Background" Value = "DarkGreen"/>
  <Setter Property = "Foreground" Value = "Yellow"/>
</Style>
```

Данный стиль подходит только для элементов управления типа `Button` (или объектов подкласса `Button`)[6], и если применить его к несовместимому элементу, то это повлечет за собой ошибки разметки и компиляции. Но если применить этот новый стиль к элементу управления типа `Button`, как показано ниже, то последний будет выглядеть аналогично приведенному на рис. 5.3.

```
<Button Content="Button!" ...
Style="{DynamicResource BigGreenButton}"/>
```

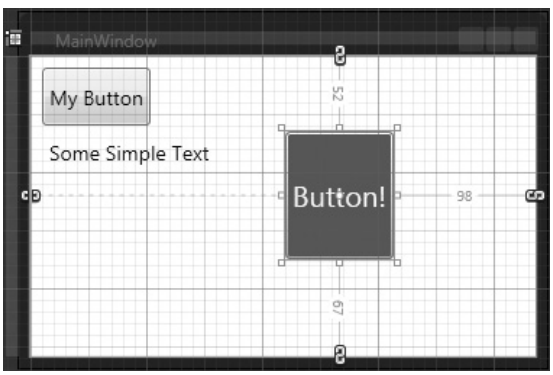


Рис. 5.3. Несколько иной и более привлекательный стиль оформления кнопок

[6] Если требуется стиль применить ко всем видам кнопок, т.е. к объектам типа `Button`, `ToggleButton`, `RepeatButton` и так далее, установите в атрибуте `TargetType` значение **`ButtonBase`**.

Примечание. При создании стиля для целевого объекта конкретного типа совершенно не обязательно присваивать значение свойству, которое не поддерживается целевым объектом. Ведь если какое-то свойство не поддерживается в целевом объекте данного типа, то оно просто игнорируется.

Подклассификация существующих стилей

Новые стили можно создавать из уже существующих, используя последние в качестве отправной точки и опираясь на свойство `BasedOn`. Расширяемому стилю должен быть присвоен соответствующий ключ в словаре с помощью атрибута `x:Key`, поскольку производный стиль будет ссылаться на него по имени в расширении разметки `{StaticResource}`.

Примечание. Свойство `BasedOn` не может быть установлено в расширении разметки `{DynamicResource}`. Для создания производного стиля в проектах на платформах WPF и Silverlight следует использовать расширение разметки `{StaticResource}`.

Ниже приведен пример разметки, описывающей новый стиль `TiltButton` (Наклонная кнопка), созданный на основании стиля `BigGreenButton` и разворачивающий элемент управления типа `Button` на 20 с помощью элемента разметки `<RotateTransform>`.

```
<!-- Этот стиль основывается на стиле BigGreenButton -->
<Style x:Key ="TiltButton" TargetType="Button"
      BasedOn = "{StaticResource BigGreenButton}">
  <Setter Property = "Foreground" Value = "White"/>
  <Setter Property = "RenderTransform">
    <Setter.Value>
      <RotateTransform Angle = "20"/>
    </Setter.Value>
  </Setter>
</Style>
```

Если вы определите этот новый стиль в файле разметки приложения `App.xaml file`, то в конечном итоге получите такой же результат, как и на рис. 5.4, где данный стиль применен к элементу управления типа `Button`.

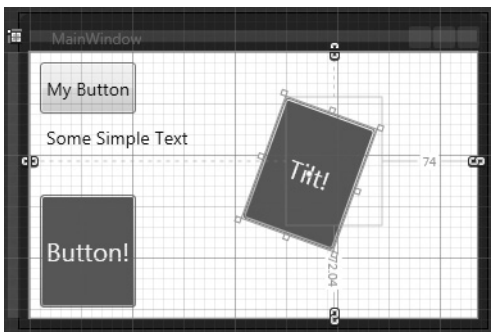


Рис. 5.4. Еще более привлекательный стиль оформления кнопок

Определение используемых по умолчанию стилей

А теперь допустим, что требуется обеспечить одинаковый вид всех элементов управления типа `TextBox` в текущем проекте и что для этой цели уже определен стиль в качестве ресурса, применяемого на уровне приложения, т.е. доступного во всех его окнах.

Несмотря на всю привлекательность такого решения, у него имеется один существенный недостаток: при наличии в приложении целого ряда окон с многочисленными элементами управления типа `TextBox` свойство `Style` придется устанавливать неоднократно!

Стили на платформах WPF и Silverlight можно применять неявным образом ко всем элементам управления в заданной области действия разметки XAML. Для создания такого стиля необходимо воспользоваться атрибутом `TargetType`, но при этом *опустить* присваивание значения `x:Key` ресурсу типа `Style`. Такой "безымянный" стиль может быть далее автоматически применен ко всем элементам управления типа `TextBox` в текущем приложении, как показано в приведенном ниже примере разметки.

```
<!-- Стиль, применяемый по умолчанию ко всем текстовым полям -->
  <Style TargetType="TextBox">
  <Setter Property = "FontSize" Value ="14"/>
  <Setter Property = "Width" Value = "100"/>
  <Setter Property = "Height" Value = "30"/>
  <Setter Property = "BorderThickness" Value = "5"/>
  <Setter Property = "BorderBrush" Value = "Red"/>
  <Setter Property = "FontStyle" Value = "Italic"/>
</Style>
```

Данный стиль определен без значения `x:Key` и внедрен на уровне приложения, поэтому в среде Expression Blend IDE он автоматически распознается как применяемый по умолчанию стиль оформления внешнего вида всех текстовых полей, добавляемых в качестве объектов типа `TextBox` на монтажном столе. Убедитесь в этом сами, опробовав новый стиль в своем приложении. При этом вы непременно обнаружите, что данный стиль автоматически применяется без всяких усилий с вашей стороны. В самом деле, если вы проанализируете автоматически сформированную разметку, описывающую данный стиль, то не обнаружите никаких следов установки свойства `Style` для объектов типа `TextBox`. В конце концов, это *применяемый по умолчанию стиль*!

Примечание. Если вам требуется определить в своем проекте элемент управления, не получающий применяемый по умолчанию стиль, воспользуйтесь редактором XAML, чтобы установить в свойстве `Style` данного элемента строковое значение **`{x:Null}`**. Это расширение разметки XAML, по существу, уведомляет исполняющую систему о необходимости полностью проигнорировать любой стиль, применяемый по умолчанию в вашем проекте, и визуализировать данный элемент управления так, как он определен. Кроме того, можете просто присвоить интересующему вас элементу управления другой, снабженный ключом стиль.

Нетрудно себе представить, что специально выделенный словарь ресурсов можно заполнить целым набором особых стилей оформления, чтобы определить используемые по умолчанию установки свойств большого числа элементов управления. Если же принять меры к тому, чтобы ни один из этих ресурсов не был именован посредством атрибута `x:Key`, то в новых проектах WPF или Silverlight элементы управления будут автоматически настраиваться на нужные стили оформления. Для этого достаточно ввести XAML-файл ресурсов со стилями оформления в проект и объединить его с файлом разметки приложения в области его действия, как пояснялось в главе 2.

Управление существующими стилями в среде Expression Blend IDE

И в завершение первого примера проекта откройте в среде Expression Blend IDE панель Resources. Обратите внимание на то, что все созданные вами стили перечислены на этой панели наряду с любыми другими специальными ресурсами объектов, которые вам довелось определить. На рис. 5.5 показаны ресурсы, применяемые на уровне приложения в рассматриваемом здесь примере проекта.

Рис. 5.5. Стили, перечисляемые на панели Resources

Если вы щелкнете на кнопке Edit Resource (Править ресурс) справа от выбранного стиля, то откроется интегрированный в Expression Blend редактор стилей. После этого можете беспрепятственно воспользоваться панелью Properties, чтобы видоизменить установки свойств выбранного стиля. В качестве примера на рис. 5.6 (а также на цветной вклейке к книге) приведен результат правки в этом редакторе стиля `TiltButton`, в котором цвет фона был изменен на лиловый, а в свойстве `Width` установлено значение 200 на панели Properties.

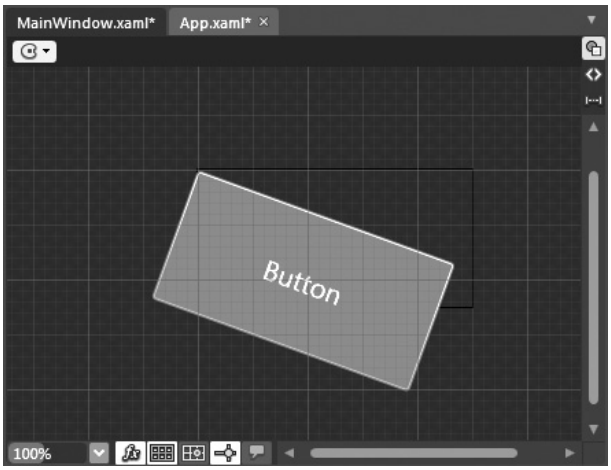


Рис. 5.6. Существующие стили могут быть выбраны для правки в среде Expression Blend IDE

Попутно заметим, что если просмотреть категорию Styles в библиотеке ресурсов, то можно заметить, что в ней перечислены стили, специально созданные для текущего проекта, кроме любых стилей, определенных в нем как используемые по умолчанию (рис. 5.7). Если перетащить любой из этих стилей на монтажный стол, то автоматически сформируется элемент управления, в свойстве `Style` которого будет установлен связанный с ним ресурс.

А теперь, когда у вас сложилось более или менее полное представление о назначении и применении стилей, перейдем к рассмотрению способов и средств Expression Blend IDE, помогающих в создании новых стилей.

Исходный код. Исходный код примера проекта `WpfStyleByHand` находится в папке `Ch 5 Code` загружаемого архива примеров проектов к данной книге.

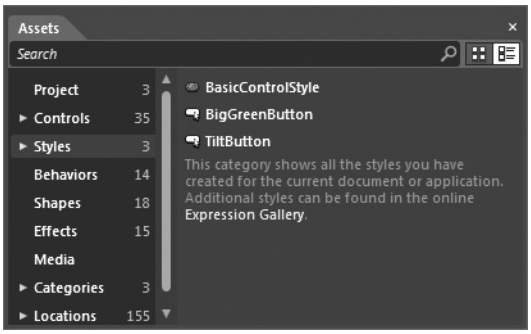


Рис. 5.7. Стили можно перетаскивать из библиотеки ресурсов для быстрого определения стилизованного элемента управления

Создание новых стилей в Expression Blend

В следующем примере проекта вам предстоит создать несколько стилей в среде Expression Blend IDE и заодно освоить более совершенные способы обращения со стилями. И в этом случае речь пойдет о приложении WPF, поскольку в завершение данного примера в разделе "Работа с простыми стилями на платформе WPF" будут рассмотрены некоторые способы, которые *не* поддерживаются на платформе Silverlight. Даже если вас интересует, главным образом, платформа Silverlight и ее прикладной интерфейс API, прочитайте весь приведенный ниже материал, поскольку в нем описан целый ряд способов и средств Expression Blend, пригодных для применения на обеих платформах.

Создайте новый проект приложения WPF, присвоив ему имя `WpfStylesWithBlend`. Создание специального стиля в среде Expression Blend IDE обычно начинается с размещения экземпляра стилизуемого элемента управления на монтажном столе. Вы вольны выбрать из библиотеки ресурсов любой элемент, но для целей данного примера рекомендуется выбрать самый простой элемент управления типа `Button`. Если же вы выберете более сложный элемент управления (например, типа `Calendar` или `TreeView`), то сформированная первоначальная разметка, описывающая стиль оформления такого элемента управления, окажется довольно пространной.

Создание нового пустого стиля

Выберите элемент управления на монтажном столе или на панели Objects and Timeline (опять же в данном примере предполагается, что это простой элемент типа `Button`). Затем выберите команду меню Object⇨Edit Style (Объект⇨Править стиль), как показано на рис. 5.8.

По команде меню Object⇨Edit Style доступны три основные операции со стилями, которые *могут* или *не могут* оказаться доступными в зависимости от того, каким образом выбранный в настоящий момент элемент управления определен в разметке XAML. По существу, у вас имеются три варианта выбора.

- Edit Current **(Править текущий стиль).** При выборе этой команды можно поправить стиль, применяемый к выбранному в настоящий момент элементу пользовательского интерфейса. Эта команда меню оказывается недоступной, если вы выбрали элемент управления, в котором пока еще не установлено свойство `Style`.
- Edit a Copy **(Править копию).** При выборе этой команды можно получить копию применяемого в настоящий момент стиля вместе с текущими свойствами выбранного элемента пользовательского интерфейса.

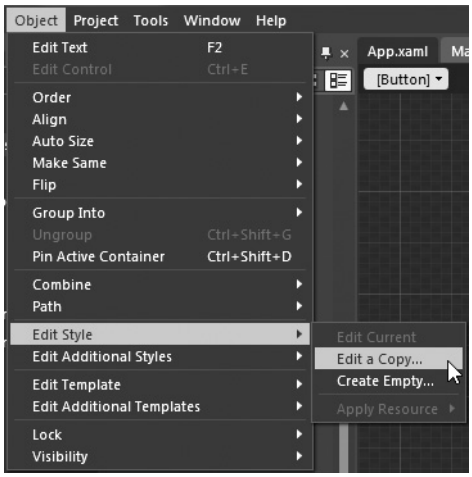


Рис. 5.8. Меню Object служит отправной точкой для создания стилей в среде Expression Blend IDE

- Create Empty **(Создать пустой стиль).** При выборе этой команды можно создать пустой стиль, в котором атрибуту `TargetType` присваивается значение, соответствующее типу выбранного элемента пользовательского интерфейса.

Примечание. В меню Object⇨Edit Style доступна также команда Apply Resource (Применить ресурс), но при условии, что в текущем проекте содержатся ресурсы существующих стилей. Это еще один способ применения стилей к элементам, выбранным на монтажном столе.

У элемента управления типа `Button`, с которым вы работаете в настоящий момент, пока еще отсутствует заданный специальный стиль, поэтому вам доступны для выбора только две команды: Edit a Copy и Create Empty. Выберите команду Create Empty, чтобы открыть диалоговое окно, в котором вы сможете ввести имя нового стиля или сделать его используемым по умолчанию, не снабженным ключом стилем, выбрав вариант Apply to all (Применить ко всем элементам). Кроме того, можете указать место для хранения нового стиля в качестве ресурса объекта. Для рассматриваемого здесь примера проекта выберите вариант Application, чтобы создать специальный стиль `firstButtonStyle` (стиль первой кнопки), сохраняемый на уровне приложения (рис. 5.9).

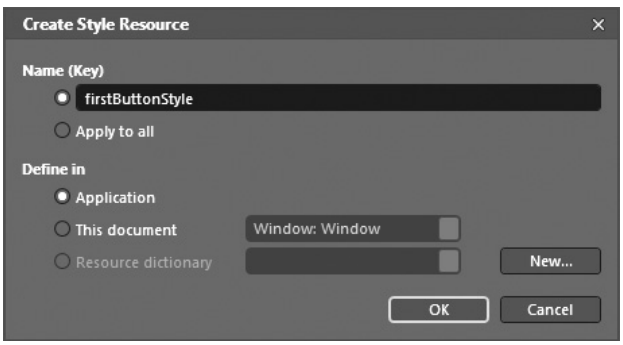


Рис. 5.9. Создание нового именованного стиля для оформления кнопки, представленной объектом типа `Button`

Как только вы щелкнете на кнопке OK, появится новый визуальный конструктор для вновь созданного стиля. В настоящий момент на нем находится нестилизованная кнопка, представленная объектом типа `Button` (рис. 5.10).

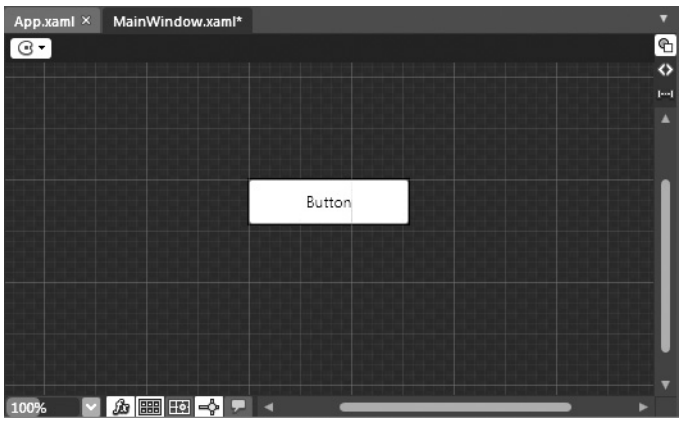


Рис. 5.10. Стили можно править как на монтажном столе, так и на панели Properties

Примечание. Не забывайте, что вы всегда сможете выбрать стиль для правки, найдя его на панели Resources и щелкнув на кнопке Edit resource справа от выбранного стиля.

Ваша задача — установить на панели Properties различные свойства стиля аналогично настройке свойств отдельного экземпляра элемента управления в главном окне (объект типа `Window`) или на начальной веб-странице (объект типа `UserControl`). Для этой цели можете, в частности, воспользоваться редактором кистей, инструментами графических преобразований, редакторами заполнения и полей и т.д. Более того, можете создать стиль, в котором применяются фрагменты анимации, что вам и предстоит сделать далее в главе; шаблоны привязки данных (см. главу 6), а также другие интересующие вас элементы графического оформления.

Примечание. Как упоминалось в главе 1, содержимое монтажного стола можно масштабировать колесиком мыши или с помощью соответствующих элементов управления монтажного стола. Такая возможность оказывается весьма кстати при построении сложных стилей и шаблонов.

Вы вольны исследовать все варианты и средства для создания стиля и выбрать наиболее подходящие для вас, но ради простоты рассматриваемого здесь примера проекта ниже приводится первоначальная разметка, которая может быть затем отредактирована. Она размещается в файле `App.xaml`, поскольку данный стиль определен как ресурс, применяемый на уровне приложения.

```
<Style x:Key="firstButtonStyle" TargetType="{x:Type Button}"/>
```

На рис. 5.11 (а также на цветной вклейке к книге) приведен результат правки стиля `firstButtonStyle` различными средствами, доступными на панели Properties. Вам совсем не обязательно следовать этому примеру, чтобы добиться точно такого же стиля оформления кнопки. В частности, на вашем компьютере может отсутствовать шрифт, использованный в тексте надписи кнопки. Просто уделите немного времени правке данного стиля, внося в него изменения по своему усмотрению и ради собственного эстетического удовольствия.

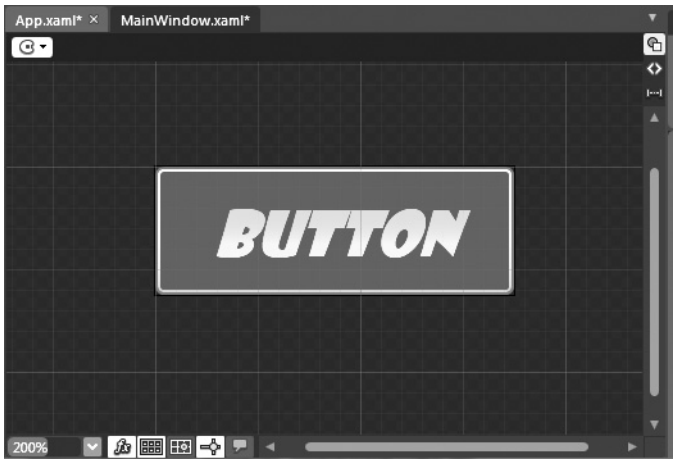


Рис. 5.11. Специальный стиль оформления кнопки, созданный в редакторе стилей, интегрированном в среду Expression Blend

Завершив создание специального стиля, уделите немного времени анализу разметки, автоматически сформированной в коде XAML для описанного этого стиля. Объем этой разметки зависит от сложности внесенных вами правок. Но даже в упрощенном варианте стиля `firstButtonStyle`, малопригодном для практического применения, приведенная ниже разметка, сформированная в коде XAML на основании сделанных установок, выглядит довольно пространной.

```
<Style x:Key="firstButtonStyle" TargetType="{x:Type Button}">
  <Setter Property="Background" Value="#FF154BD0"/>
  <Setter Property="BorderBrush" Value="#FFE20C0C"/>
  <Setter Property="FontSize" Value="24"/>
  <Setter Property="FontFamily" Value="Showcard Gothic"/>
  <Setter Property="Foreground">
    <Setter.Value>
      <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
       <GradientStop Color="#FF26D096" Offset="0"/>
       <GradientStop Color="#FFEFE710" Offset="1"/>
      </LinearGradientBrush>
    </Setter.Value>
  </Setter>
  <Setter Property="BorderThickness" Value="5"/>
  <Setter Property="Height" Value="50"/>
  <Setter Property="Width" Value="140"/>
  <Setter Property="FontStyle" Value="Italic"/>
</Style>
```

После внесенных правок первоначальный внешний вид элемента управления в исходном окне типа `Window` должен автоматически измениться. Если же этого не происходит, убедитесь в том, что сохранили правки, внесенные в новый стиль. А если хотите добавить больше кнопок в пользовательский интерфейс своего проекта, перетащите стиль их оформления из категории Styles библиотеки стилей на монтажный стол, как было показано ранее на рис. 5.7. В качестве примера на рис. 5.12 (а также на цветной вклейке к книге) приведен внешний вид главного окна приложения (объекта типа `Window`) с тремя кнопками (объектами типа `Button`), оформленными в одном и том же стиле `firstButtonStyle` и снабженными текстом, поясняющим, что в них применяется этот стиль.

Рис. 5.12. Несколько кнопок, оформленных в одном и том же стиле

Работа с простыми стилями на платформе WPF

Итак, вы научились выбирать элемент управления на монтажном столе и создавать пустой стиль его оформления (или же копировать уже имеющийся стиль) для последующей правки. В некоторых случаях такой подход оказывается практически идеальным, но как быть, если требуется определить стили для оформления двадцати разных элементов управления в своем новом проекте на платформе WPF. Допустим, вам нужно разработать новую "тему" для своего приложения, где каждый элемент управления должен изменять свой вид по сезону, т.е. зимнюю, летнюю и пр. Нетрудно себе представить, насколько трудоемкой окажется процедура переноса каждого элемента управления на монтажный стол для извлечения копии нужного стиля оформления.

К счастью, в проектах на платформе WPF, разрабатываемых в среде Expression Blend IDE, имеется возможность пользоваться полным набором простых стилей в качестве отправной точки для графического оформления пользовательского интерфейса приложения. Вместе с простыми стилями в текущий проект автоматически вводится целый ряд новых ресурсов объектов, которые можно править по своему усмотрению. Благодаря такому подходу существенно упрощается процесс создания стилей, поскольку в формируемую для них разметку включаются многочисленные и очень важные установки, требующиеся для того, чтобы элементы пользовательского интерфейса функционировали именно так, как и предполагалось.

Попробуйте открыть библиотеку ресурсов (или панель Assets) и развернуть категорию Styles. В этой категории вы обнаружите раздел Simple Styles (Простые стили), как показано на рис. 5.13.

Выберите один из простых стилей, например `SimpleSlider` (Простой ползунок), и перетащите его на монтажный стол. Выделите данный объект и измените его размеры, чтобы растянуть его по ширине окна. В итоге объект, оформленный простым стилем `SimpleSlider`, должен стать похожим на обычный ползунковый элемент управления (рис. 5.14).

Но если вы проанализируете разметку, описывающую данный объект, то обнаружите, что в его свойстве `Style` задан простой стиль `SimpleSlider`:

```
<Slider Margin="128,0,129.507,55" Style="{DynamicResource SimpleSlider}"
      VerticalAlignment="Bottom"/>
```

Откройте панель Resources в своем проекте. Обратите внимание на то, что в разделе ресурсов иерархической структуры файла разметки приложения `App.xaml` появился новый узел `Linked To`, обозначающий связь с новым XAML-файлом `Simple Styles.xaml` (рис. 5.15).

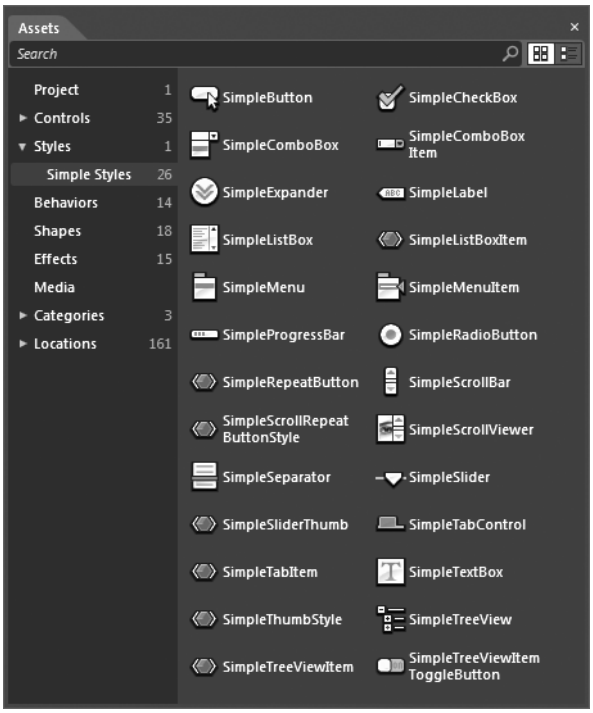


Рис. 5.13. Простые стили, применяемые на платформе WPF и доступные в библиотеке ресурсов

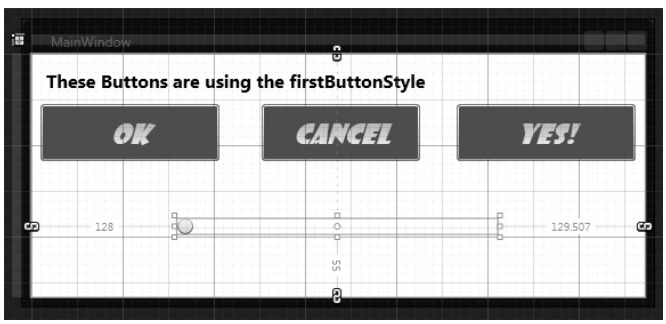


Рис. 5.14. На первый взгляд простой стиль `SimpleSlider` напоминает обычный ползунковый элемент управления

Рис. 5.15. Узел `Linked To` обозначает, что с приложением объединен новый XAML-файл

С точки зрения разметки XAML узел Linked To появился в результате объединения нового XAML-файла с приложением в контейнере ресурсов последнего. Открыв файл разметки приложения App.xaml в редакторе XAML, вы обнаружите в нем новые строки разметки, приведенные ниже.

```
<ResourceDictionary.MergedDictionaries>
  <ResourceDictionary Source="Simple Styles.xaml"/>
</ResourceDictionary.MergedDictionaries>
```

Но еще важнее следующее обстоятельство: если вы вернетесь к панели Resources, предварительно убедившись в том, что окно монтажного стола активизировано в среде Expression Blend IDE, иначе панель Resources окажется пустой, а затем развернете узел файла Simple Styles.xaml, то обнаружите, что в него сдублированы все основные элементы управления на платформе WPF. Кроме того, в этом файле определен целый ряд дополнительных ресурсов, в том числе кисти, используемые по умолчанию для раскраски разных составляющих элементов управления (заднего, переднего плана и т.д.). Как показано на рис. 5.16 (а также на цветной вклейке к книге), используемые по умолчанию кисти могут быть отредактированы в редакторе кистей, интегрированном в среду Expression Blend IDE.

Так, если изменить оттенок цвета раскраски обычной кистью (NormalBrush), соответственно изменится и окраска ползунка в элементе управления типа Slider, как показано на рис. 5.17, а также на цветной вклейке к книге.

И разумеется, если выбрать конкретный стиль на панели Resources для последующей правки, этот простой ресурс можно видоизменить еще больше. На рис. 5.18 показан простой стиль SimpleSlider, выбранный для правки.

Примечание. В Интернете можно найти немало ресурсов для обмена стилями среди разработчиков и художников-оформителей приложений на платформах WPF и Silverlight. К числу наиболее предпочтительных ресурсов подобного рода относится веб-сайт, доступный по адресу www.wpfstyles.com. На нем представлены десятки стилей, которые можно свободно загружать и применять в своих проектах, что очень полезно для тех, у кого, как и у меня, нет особых художественных талантов!

Просмотр разметки простого стиля

Откройте файл Simple Styles.xaml для просмотра в редакторе XAML и уделите немного времени анализу его содержимого. Для примера найдите стиль SimpleSliderThumb. Обратите внимание на то, что он содержит встраиваемый элемент разметки <ControlTemplate>, в котором отдельно определяется целый ряд других специальных средств, включая триггеры. Ниже приведен фрагмент разметки из данного файла.

```
<Style x:Key="SimpleSliderThumb" d:IsControlPart="True"
      TargetType="{x:Type Thumb}">
...
  <ControlTemplate TargetType="{x:Type Thumb}">
    <Grid>
      <Ellipse x:Name="Ellipse" Fill="{DynamicResource NormalBrush}"
             Stroke="{DynamicResource NormalBorderBrush}"
             StrokeThickness="1"/>
    </Grid>
    <ControlTemplate.Triggers>
      <Trigger Property="IsMouseOver" Value="True">
        <Setter Property="Fill" Value="{DynamicResource MouseOverBrush}"
              TargetName="Ellipse"/>
```

```
    </Trigger>
...
  </ControlTemplate.Triggers>
...
</Style>
```

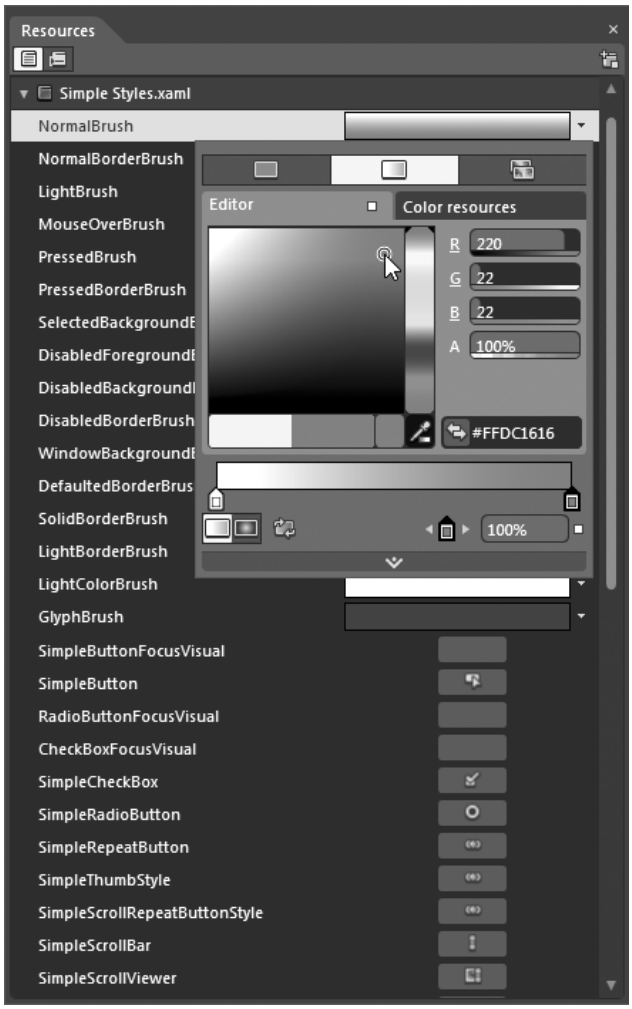


Рис. 15.16. В файле `Simple Styles.xaml` определены простые стили и связанные с ними ресурсы для каждого основного элемента управления на платформе WPF

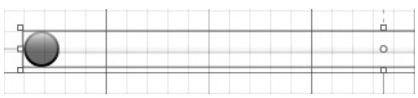


Рис. 15.17. В результате правки кистей, включенных в состав файла `Simple Styles.xaml`, изменяется внешний вид элементов управления, оформляемых данным простым стилем

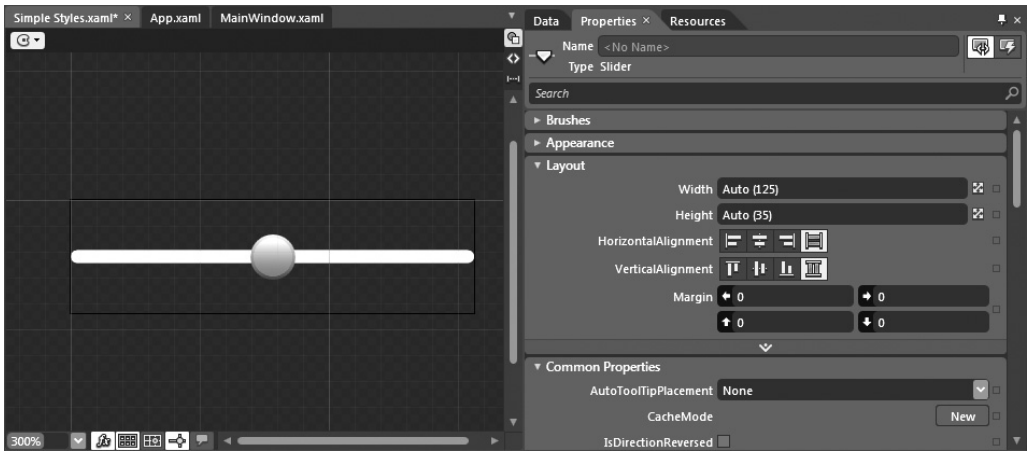


Рис. 5.18. Простые стили можно править таким же образом, как и любые другие стили

Как видите, используемые по умолчанию простые стили сложнее, чем кажется на первый взгляд! Для того чтобы эта разметка стала яснее, рассмотрим назначение шаблонов элементов управления. Но прежде следует заметить, что в руководстве пользователя Expression Blend имеется целый раздел, посвященный внесению типичных коррективов в простые стили на платформе WPF. Этот раздел называется "Рекомендации по оформлению простых стилей на платформе WPF" (Styling tips for WPF Simple Styles). В качестве примера на рис. 5.19 приведено окно справочной системы Expression Blend, в котором поясняется, каким образом в простом стиле `SimpleSlider` осуществляется специальная настройка оформления ползунка.

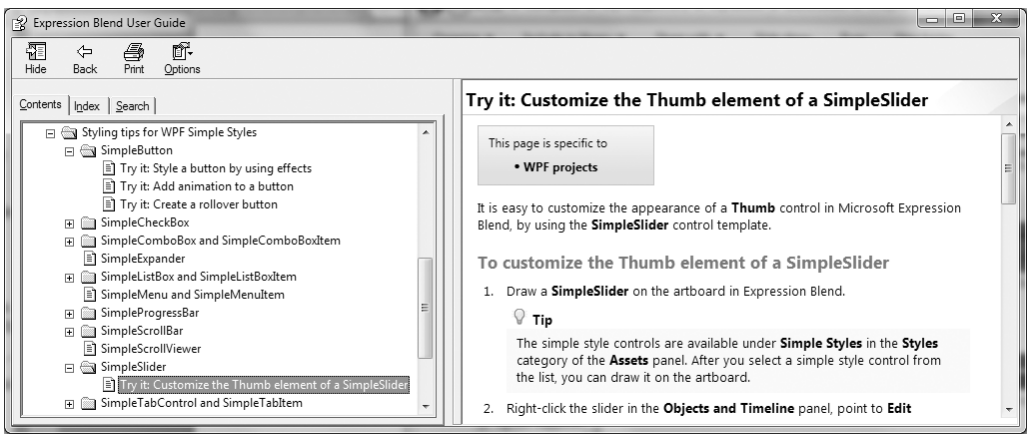


Рис. 5.19. В руководстве пользователя Expression Blend предоставляется немало учебного материала по специальной настройке простых стилей на платформе WPF

Исходный код. Исходный код примера проекта `WpfStylesWithBlend` находится в папке `Ch 5 Code` загружаемого архива примеров проектов к данной книге.

Назначение шаблонов элементов управления в стилях

Даже самый простой стиль в действительности служит не более чем контейнером для установок общих свойств элемента управления определенного типа. Несмотря на всю пользу, которую стили приносят для упрощения настройки многих свойств (цвета, размера шрифта и т.д.) элементов управления целевого типа, общий порядок визуализации элементов управления остается прежним. Поэтому, независимо от характера стилевого оформления элемента управления типа `Button`, он все равно имеет форму прямоугольной, а возможно, и квадратной кнопки.

Иногда возникает потребность создать стиль, который не только изменяет ряд установок общих свойств целевого элемента управления, но и переопределяет форму самого элемента управления. Так, в конкретном проекте может потребоваться круглая форма кнопки или произвольная многоугольная форма, нарисованная инструментом Pen или Pencil.

Когда требуется создать стиль, способный изменить геометрическую форму визуализируемого элемента управления, этот стиль приходится определять таким образом, чтобы он содержал встроенный шаблон *элемента управления*, описываемый в разметке XAML элементом `<ControlTemplate>`. А в этом элементе разметки можно определить внешний вид нового элемента управления, воспользоваться диспетчерами компоновки для доступа к их содержимому и выполнить ряд других типичных операций.

Примечание. Несмотря на то что шаблон является наиболее часто используемой составляющей более крупного стиля, имеется все же возможность определить автономный ресурс объекта в элементе разметки `<ControlTemplate>`. Настраивая свойство `Template` (Шаблон) элементов управления, можно назначать для них заданные шаблоны. В следующем примере проекта будет показано, как это делается.

По умолчанию внешний вид каждого элемента управления на платформах WPF и Silverlight визуализируется с помощью связанного с ними шаблона, используемого *по умолчанию*. Например, шаблон, используемый по умолчанию для элемента управления типа `Button`, содержит инструкции по визуализации так хорошо знакомой вам прямоугольной формы кнопки. По этому же принципу визуализируются все остальные элементы управления на платформах WPF и Silverlight. Используемые по умолчанию шаблоны входят в состав библиотек на платформах WPF и Silverlight в качестве встраиваемых XAML-ресурсов, и поэтому не могут быть изменены *непосредственно*. Но, как будет показано далее, в среде Expression Blend предоставляется ряд механизмов для извлечения копии используемого по умолчанию шаблона с целью его последующей правки.

Помимо определения различных инструкций по визуализации общего внешнего элемента управления, шаблон содержит также инструкции по реагированию элемента управления на действия пользователя и прочие изменения состояния. Например, в шаблоне, используемом по умолчанию для элемента управления типа `Button`, описывается, каким образом должна выглядеть кнопка, когда она получает логический фокус, когда производится щелчок в пределах ее границ, когда она становится неактивной, и в прочих состояниях.

Для определения нового шаблона (или видоизменения уже имеющегося), как правило, требуется предоставлять специальные элементы разметки XAML, описывающие реакцию элемента управления на те же самые изменения состояния. Как будет показано далее, в шаблоне на платформе WPF можно определить визуальные подсказки, используя два следующих средства: триггеры или диспетчер визуальных состояний

(VSM — Visual State Manager), тогда как на платформе Silverlight это можно сделать только с помощью диспетчера VSM. Далее будет также показано, что в среду Expression Blend интегрированы специальные редакторы, предназначенные для работы с каждым из этих двух средств.

Примечание. Диспетчер VSM был впервые введен в состав прикладного интерфейса Silverlight API и служит для внедрения визуальных подсказок в шаблон на платформе Silverlight. Исторически сложилось так, что программирующие на платформе WPF применяли аналогичный подход посредством триггеров. Но после выпуска версии .NET 4.0 платформа WPF была обновлена, чтобы поддерживать диспетчер VSM, а, следовательно, программирующие на этой платформе получили возможность пользоваться двумя разными средствами (триггерами и диспетчером VSM) для реализации визуальных подсказок. Применение обоих этих средств демонстрируется в рассматриваемых далее примерах.

Построение специального шаблона элемента управления вручную

Прежде чем переходить к рассмотрению различных способов и средств Expression Blend, упрощающих построение специальных шаблонов, покажем, каким образом простой шаблон создается вручную. Ведь лучшего способа, чем этот, для уяснения особенностей создания шаблонов трудно придумать. Когда же вы усвоите эти особенности на примере построения шаблона вручную, вам будет легче создавать другие шаблоны в среде Expression Blend IDE.

Итак, создайте новый проект приложения WPF, присвоив ему имя `WpfTemplatesByHand`. После этого добавьте одну кнопку (объект типа `Button`) на монтажном столе в исходном окне типа `Window`. В настоящий момент кнопка визуализируется с помощью связанного с ней шаблона, используемого по умолчанию. Как упоминалось ранее, этот шаблон является ресурсом, встраиваемым в соответствующую библиотеку на платформе WPF или Silverlight. Определяя свой шаблон, вы, по существу, заменяете набор инструкций из используемого по умолчанию шаблона собственным набором инструкций.

Для того чтобы начать создание своего шаблона, откройте редактор XAML в исходном окне типа `Window` и обновите определение кнопки в элементе разметки `<Button>`, указав в нем новый встраиваемый шаблон[7], который будет постепенно видоизменен и усовершенствован. Обратите внимание на то, что в приведенной ниже разметке свойствам `Height` и `Width` присвоено значение **100**, а свойство `Content` элемента управления типа `Button` вообще удалено. Остальные атрибуты в элементе разметки `<Button>`, открывающем определение кнопки, оставлены без изменения.

```
<Button Width="100" Height="100" ... >
  <Button.Template>
    <ControlTemplate>
      <Grid x:Name="controlLayout">
        <Ellipse x:Name="buttonSurface" Fill = "LightBlue"/>
        <Label x:Name="buttonCaption" VerticalAlignment = "Center"
               HorizontalAlignment = "Center"
               FontWeight = "Bold" FontSize = "20" Content = "OK!"/>
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```

[7] В процессе работы над рассматриваемым здесь примером проекта вам придется перенести этот шаблон в словарь ресурсов своего приложения.

В приведенной выше разметке определен шаблон, состоящий из именованного элемента управления типа `Grid`, содержащего, в свою очередь, элементы управления типа `Ellipse` и `Label`. А поскольку строки или столбцы сетки не определены, то каждый последующий потомок объекта типа `Grid` располагается над предыдущим, что дает возможность без труда отцентровать содержимое данного объекта с помощью его свойств `VerticalAlignment` (Выравнивание по вертикали) и `HorizontalAlignment` (Выравнивание по горизонтали). Обратите также внимание на то, что все содержимое элемента разметки `<ControlTemplate>` присваивается свойству `Template` элемента управления типа `Button` в элементе разметки `<Button.Template>`.

Для последующей проверки своего шаблона организуйте на панели Properties обработку события `Click`, наступающего после щелчка на кнопке, представленной объектом типа `Button`. (Напомним, что для выбора событий, наступающих для отдельного элемента управления, служит кнопка со знаком молнии на пиктограмме.) Затем введите в сформированный обработчик событий строку кода, в которой организуется вывод на экран простого сообщения `You clicked the button!` (Вы щелкнули на кнопке), как показано ниже.

```
private void Button_Click(object sender, System.Windows.RoutedEventArgs e)
{
  MessageBox.Show("You clicked the button!");
}
```

Если вы запустите теперь свое приложение на выполнение, то обнаружите, что событие `Click` наступает только в том случае, если курсор мыши оказывается в пределах эллипса или круга (объекта типа `Ellipse`), но не в углах по его краям. Это очень ценное свойство архитектуры шаблонов на платформах WPF и Silverlight, поскольку оно избавляет от необходимости выполнять перерасчет проверки попадания курсора, проверку границ или другие мелкие операции низкого уровня. Так, если в вашем шаблоне объект типа `Path` используется для визуализации неправильной геометрической формы, можете быть уверены в том, что мелкие операции проверки попадания курсора будут выполняться относительно формы элемента управления, а не более крупного ограничивающего ее прямоугольника.

Сохранение шаблонов в виде ресурсов

В настоящий момент ваш шаблон встроен в конкретный элемент управления типа `Button`, что, безусловно, ограничивает его повторное применение. В идеальном случае можете поместить свой шаблон круглой кнопки в словарь ресурсов, чтобы неоднократно использовать его в разных проектах, или хотя бы перенести его в контейнер ресурсов приложения, чтобы еще раз воспользоваться им в рамках текущего проекта.

Итак, переместите ресурс локального объекта типа `Button` на уровень приложения. Для этого найдите сначала свойство `Template` данного объекта типа `Button` на панели Properties. Затем откройте меню дополнительных параметров, щелкнув на белом квадратике справа от текстового поля данного свойства. Выберите из этого меню команду Convert to New Resource, как показано на рис. 5.20.

Определите в открывшемся диалоговом окне новый шаблон, сохраняемый под именем `RoundButtonTemplate` (Шаблон круглой кнопки) в качестве ресурса приложения. На данной стадии работы над рассматриваемым здесь проектом вы обнаружите в файле `App.xaml` следующую разметку.

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

```
  StartupUri="MainWindow.xaml">
  <Application.Resources>
    <ControlTemplate x:Key="RoundButtonTemplate"
                     TargetType="Button">
      <Grid x:Name="controlLayout">
        <Ellipse x:Name="buttonSurface" Fill = "LightBlue"/>
        <Label x:Name="buttonCaption" VerticalAlignment = "Center"
              HorizontalAlignment = "Center"
              FontWeight = "Bold" FontSize = "20" Content = "OK!"/>
      </Grid>
    </ControlTemplate>
  </Application.Resources>
</Application>
```

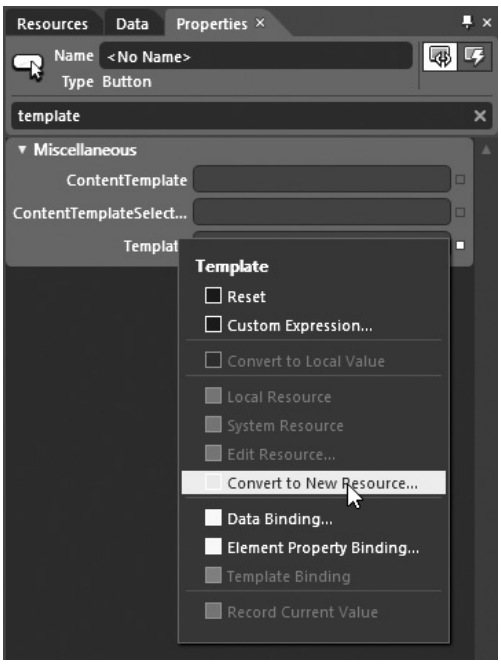


Рис. 5.20. Извлечение локального ресурса

Обратите также внимание на то, что разметка, описывающая исходный объект типа `Button`, видоизменена. Как показано ниже, в свойстве `Template` этого объекта ваш шаблон установлен в качестве специального именованного ресурса.

```
<Button HorizontalAlignment="Left" Margin="44,30,0,0"
      VerticalAlignment="Top" Width="100" Height="100"
      Click="Button_Click"
      Template="{DynamicResource RoundButtonTemplate}"/>
```

Этот ресурс теперь доступен для всего приложения, а следовательно, в данном приложении можно определить сколько угодно круглых кнопок. Итак, добавьте еще два элемента управления типа `Button` на монтажном столе исходного окна типа `Window`. Затем перейдите к панели Properties и установите в свойстве `Template` каждого из этих элементов управления ресурс `RoundButtonTemplate`, как показано на рис. 5.21.

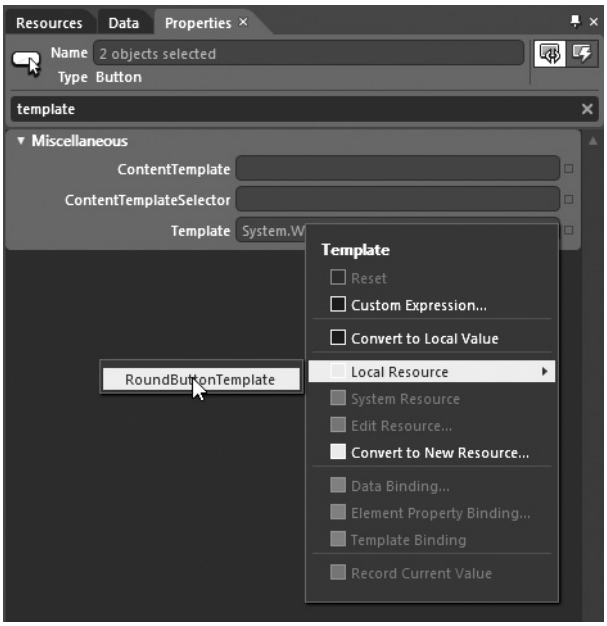


Рис. 5.21. Установка специального шаблона в свойстве `Template` объектов типа `Button`

Конечный результат приведен на рис. 5.22.

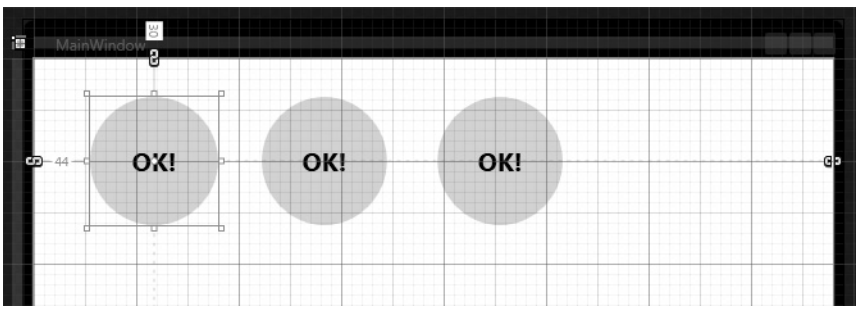


Рис. 5.22. Три круглые кнопки, оформленные по единому шаблону

Внедрение визуальных подсказок с помощью триггеров на платформе WPF

Когда вы определяете специальный шаблон, удаляются также все визуальные подсказки, определенные в используемом по умолчанию шаблоне. Напомним, что используемый по умолчанию шаблон содержит инструкции по разметке, в которых уведомляется, каким образом должен выглядеть элемент управления при наступлении в пользовательском интерфейсе определенных событий, например, когда этот элемент получает логический фокус, когда на нем производится щелчок кнопкой мыши, когда он активизируется (или же делается неактивным) и т.д. Пользователи уже привыкли к подобного рода визуальным подсказкам, поскольку они создают хоть какую-то иллюзию реакции элемента пользовательского интерфейса на его касание.

В рассматриваемом здесь шаблоне `RoundButtonTemplate` подобная разметка не определена, а следовательно, независимо от действий пользователя по отношению к элементу управления, оформляемому по этому шаблону, внешний вид самого элемента не меняется. В идеальном случае внешний вид элемента управления должен хотя бы немного изменяться, когда на нем производится щелчок кнопкой мыши. Например, он может менять свой цвет или уменьшаться в размерах, чтобы дать пользователю наглядно знать о перемене своего состояния.

После первого выпуска платформы WPF для реализации подобных визуальных подсказок в шаблон вводилось любое количество триггеров, которые обычно должны были изменять значения свойств объекта или начинать анимацию по раскадровке либо делать и то и другое, когда становилось истинным условие срабатывания триггера.

В качестве упражнения по реализации подобного подхода на практике обновите шаблон `RoundButtonTemplate`, как показано в приведенной ниже разметке. После внесения указанных ниже изменений цвет фона элемента управления будет становиться голубым, а цвет переднего плана — желтым при наведении курсора мыши на этот элемент.

```
<ControlTemplate x:Key="RoundButtonTemplate" TargetType="Button">
  <Grid x:Name="controlLayout">
    <Ellipse x:Name="buttonSurface" Fill="LightBlue" />
    <Label x:Name="buttonCaption" Content="OK!" FontSize="20"
          FontWeight="Bold" HorizontalAlignment="Center"
          VerticalAlignment="Center" />
  </Grid>

  <ControlTemplate.Triggers>
    <Trigger Property = "IsMouseOver" Value = "True">
      <Setter TargetName = "buttonSurface"
            Property = "Fill" Value = "Blue"/>
      <Setter TargetName = "buttonCaption" Property = "Foreground"
            Value = "Yellow"/>
    </Trigger>
  </ControlTemplate.Triggers>
</ControlTemplate>
```

Если вы еще раз запустите свое приложение на выполнение, то обнаружите изменение цвета кнопки, когда курсор мыши оказывается в пределах ее границ. Обратите также внимание на то, что данный конкретный элемент `<Trigger>` определен таким образом, что при установке логического значения `true` в свойстве `IsMouseOver` (Курсор мыши наведен) два целевых элемента (`buttonSurface` и `buttonCaption`), указываемых в свойстве `TargetName` (Имя целевого объекта), соответственно изменяют свое состояние. Приведенная выше разметка не должна вас особенно смущать. Ведь большинство программирующих на платформе WPF согласны с тем, что устанавливать триггеры вручную — занятие малопривлекательное. Поэтому далее будет показано, каким образом панель Triggers используется в Expression Blend для автоматического формирования логики работы триггеров.

Ниже приведен еще один пример разметки триггера, сокращающего размеры объекта типа `Grid`, а следовательно, и всех его потомков, когда на элементе управления производится щелчок кнопкой мыши. Добавьте эту разметку в коллекцию `<ControlTemplate.Triggers>`, а затем запустите свое приложение на выполнение и проверьте его работоспособность с учетом нового триггера, запускающего еще одну визуальную подсказку.

Итак, у вас теперь имеется специальный шаблон с несколькими визуальными подсказками, внедренными с помощью триггеров на платформе WPF. В следующем далее примере будет продемонстрирован альтернативный способ внедрения визуальных

подсказок с помощью диспетчера визуальных состояний (VSM). Но прежде необходимо рассмотреть назначение расширения разметки {TemplateBinding} и элемента <ContentPresenter>.

Назначение расширения разметки {TemplateBinding}

Рассматриваемый здесь шаблон можно применять только к элементам управления типа Button, и поэтому имеет смысл установить его свойства в элементе разметки <Button> таким образом, чтобы шаблон проявлял себя особым образом. В частности, свойство Fill элемента управления типа Ellipse жестко закодировано на голубой цвет, тогда как свойство Content элемента управления типа Label — на строковое значение "OK!". Но ведь в пользовательском интерфейсе нужны кнопки разного цвета и разные текстовые надписи меток, поэтому можно попытаться определить следующие кнопки в главном окне приложения, представленного объектом типа Window.

```
<Grid x:Name="LayoutRoot">
  <Button Background="Red" Content="Howdy!" ...
        Template="{DynamicResource RoundButtonTemplate}"/>
  <Button Background="LightGreen" Content="Cancel!" ...
        Template="{DynamicResource RoundButtonTemplate}" />
  <Button Background="Yellow" Content="Format" ...
        Template="{DynamicResource RoundButtonTemplate}"/>
</Grid>
```

Но, несмотря на то, что каждый объект типа Button определен и настроен на применение особого цвета фона и текстовой надписи в свойствах Background и Content, все три кнопки по-прежнему имеют голубой цвет фона и текстовую надпись **OK!**. Дело в том, что свойства элементов управления типа Button, в которых *применяется* данный шаблон, не совпадают в точности со свойствами, задаваемыми *в самом* шаблоне, например, со свойством Fill элемента управления типа Ellipse. А значение свойства Content элемента управления типа Label, определенное в области действия элемента разметки <Button>, не направляется автоматически к внутреннему потомку шаблона.

Подобные затруднения могут быть разрешены с помощью расширения разметки {TemplateBinding} при построении специального шаблона. Это дает возможность захватывать значения свойств, определенных в элементе управления с помощью шаблона, чтобы использовать их для установки свойств в самом шаблоне. Ниже приведен переработанный вариант разметки шаблона RoundButtonTemplate, в котором расширение разметки {TemplateBinding} теперь используется для привязки свойства Background элемента управления типа Button к свойству Fill элемента управления типа Ellipse, а также для четкой передачи свойства Content элемента управления типа Button аналогичному свойству элемента управления типа Label.

```
<ControlTemplate x:Key="RoundButtonTemplate" TargetType="Button" >
  <Grid x:Name="controlLayout">
    <Ellipse x:Name="buttonSurface"
           Fill="{TemplateBinding Background}"/>
    <Label x:Name="buttonCaption"
          Content="{TemplateBinding Content}"
          FontSize="20" FontWeight="Bold"
          HorizontalAlignment="Center"
          VerticalAlignment="Center" />
  </Grid>
...
</ControlTemplate>
```

После такого обновления можно создать кнопки с разными цветами фона и текстовыми надписями, как показано на рис. 5.23, также на цветной вклейке к книге.

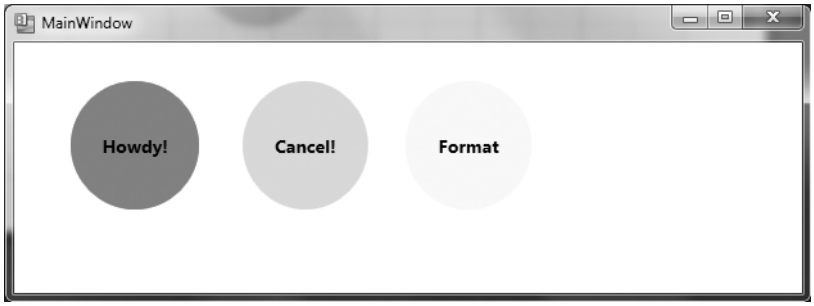


Рис. 5.23. Благодаря привязкам шаблона значения свойств могут передаваться элементам управления, определенным в шаблоне

Представление о назначении элемента разметки <ContentPresenter>

При разработке рассматриваемого здесь шаблона элемент управления `Label` был выбран для отображения текстовой надписи на кнопке. У этого элемента управления имеется такое же свойство, как и у элемента управления типа `Button`. Поэтому с помощью расширения разметки `{TemplateBinding}` можно определить элемент управления типа `Button` с более сложным содержимым, чем простая текстовая строка. В качестве примера на рис. 5.24 показана кнопка со сложным содержимым, описываемым в приведенной ниже разметке.

```
<Button Width="100" Height="100" Background="Yellow"
      Template="{StaticResource RoundButtonTemplate}"
      HorizontalAlignment="Left" ...
>

  <ListBox Height="50" Width="75">
    <ListBoxItem Content="Hello"/>
    <ListBoxItem Content="Hello"/>
    <ListBoxItem Content="Hello"/>
  </ListBox>
</Button>
```

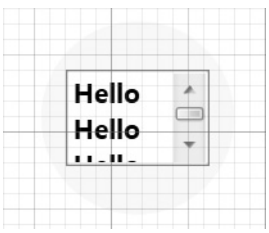


Рис. 5.24. Кнопка со сложным содержимым, оформленная по текущему шаблону

Данный конкретный элемент управления выглядит внешне и действует так, как и предполагалось. Но что, если требуется передать его сложное содержимое компоненту шаблона, не имеющему свойства `Content`? Когда в шаблоне требуется определить

обобщенную область отображения содержимого, то для этой цели можно воспользоваться элементом разметки класса <ContentPresenter> вместо того, чтобы указывать конкретный тип элемента управления (Label или TextBlock).

И хотя этого не требуется в рассматриваемом здесь примере проекта, ниже приведена разметка, в которой показано, каким образом можно создать специальный шаблон, в котором элемент разметки <ContentPresenter> используется для отображения конкретного значения свойства Content элемента управления по заданному шаблону.

```
<!-- По этому шаблону отображается все, что задано в свойстве Content
    оформляемой с его помощью кнопки -->
<ControlTemplate x:Key="NewRoundButton" TargetType="Button">
  <Grid>
    <Ellipse Fill="{TemplateBinding Background}"/>
     <ContentPresenter HorizontalAlignment="Center"
                  VerticalAlignment="Center"/>
  </Grid>
</ControlTemplate>
```

Внедрение шаблонов в стили

В настоящий момент в рассматриваемом здесь шаблоне определен внешний вид, который может принимать любой элемент управления типа Button. Но ответственность за установку основных свойств элемента управления (содержимого, размера и начертания шрифта и т.д.) по-прежнему возлагается на сам элемент управления типа Button, как показано в приведенной ниже разметке.

```
<!-- В настоящий момент значения основных свойств должны устанавливаться
    в самой кнопке, а не в шаблоне ее оформления -->
<Button Content="Yo!" Foreground ="Black" FontSize ="20"
       FontWeight ="Bold"
       Template ="{StaticResource RoundButtonTemplate}"
       Height="100" Width="100"/>
```

По желанию можете встроить шаблон в более крупный стиль. Это даст возможность присвоить значения различным свойствам элемента управления целевого типа, а также определить его особый внешний вид. Ниже приведен окончательный вариант разметки рассматриваемого здесь шаблона, который теперь встроен в стиль оформления кнопок. Обратите внимание на то, что в элементе разметки <ControlTemplate> удален атрибут x:Key, а вместо этого в элемент разметки <Style> добавлен стиль, переименованный в RoundButtonStyle.

```
<!-- Стиль, содержащий шаблон -->
<Style x:Key ="RoundButtonStyle" TargetType ="Button">
  <Setter Property ="Foreground" Value ="Black"/>
  <Setter Property ="FontSize" Value ="14"/>
  <Setter Property ="FontWeight" Value ="Bold"/>
  <Setter Property="Width" Value="100"/>
  <Setter Property="Height" Value="100"/>
  <!-- А это сам шаблон! -->
  <Setter Property ="Template">
    <Setter.Value>
      <ControlTemplate TargetType ="Button">
        <Grid x:Name="controlLayout">
          <Ellipse x:Name="buttonSurface"
                  Fill="{TemplateBinding Background}"/>
          <Label x:Name="buttonCaption"
                Content ="{TemplateBinding Content}"
                HorizontalAlignment="Center"
```

```
            VerticalAlignment="Center" />
      </Grid>
     <ControlTemplate.Triggers>
      <Trigger Property = "IsMouseOver" Value = "True">
       <Setter TargetName = "buttonSurface" Property = "Fill"
             Value = "Blue"/>
       <Setter TargetName = "buttonCaption"
             Property = "Foreground" Value = "Yellow"/>
      </Trigger>
      <Trigger Property = "IsPressed" Value="True">
       <Setter TargetName="controlLayout"
             Property="RenderTransformOrigin" Value="0.5,0.5"/>
       <Setter TargetName="controlLayout"
             Property="RenderTransform">
        <Setter.Value>
         <ScaleTransform ScaleX="0.8" ScaleY="0.8"/>
        </Setter.Value>
       </Setter>
      </Trigger>
     </ControlTemplate.Triggers>
    </ControlTemplate>
   </Setter.Value>
  </Setter>
</Style>
```

После такого обновления шаблона элементы управления типа `Button` создаются, как и прежде, а стиль их оформления указывается в свойстве `Style`:

```
<Button Background="Red" Content="Howdy!"
      Style="{StaticResource RoundButtonStyle}"/>
```

Несмотря на то что внешний и поведение кнопок остаются такими же, как и до подобного обновления шаблона, преимущество встраивания шаблонов в стили заключается в том, что с их помощью можно предоставлять готовый набор значений для общих свойств. На этом рассмотрение особенностей создания шаблонов вручную завершается. А далее основное внимание будет уделено построению шаблонов инструментальными средствами Expression Blend.

Исходный код. Исходный код примера проекта `WpfTemplatesByHand` находится в папке `Ch 5 Code` загружаемого архива примеров проектов к данной книге.

Создание шаблонов элементов управления средствами Expression Blend

Несмотря на всю простоту рассмотренного выше шаблона для оформления круглой кнопки, для его построения потребовалась немалая порция разметки XAML. Если бы вам пришлось создавать вручную крупномасштабные шаблоны, содержащие сложные кисти, виды анимации и прочее, то у вас свело бы от судороги пальцы из-за большого объема набираемого исходного текста. К счастью, в среде Expression Blend IDE имеется целый ряд средств, упрощающих процесс разработки шаблонов элементов управления.

Создание копии используемого по умолчанию шаблона

Для того чтобы упростить построение специальных шаблонов в Expression Blend, можно сначала попытаться извлечь копию используемого по умолчанию шаблона и видоизменить его по собственному усмотрению. И хотя такой подход дает больше всего

возможностей для приспосабливания стандартного шаблона под конкретные нужды, он довольно сложен, поскольку в этом случае приходится непосредственно иметь дело со всеми мелкими операциями низкого уровня. Но прежде чем переходить к некоторым довольно эффективным и в то же время простым альтернативным подходам, рассмотрим основные особенности видоизменения используемого по умолчанию шаблона.

Примечание. При создании шаблонов применять рассматриваемый здесь подход все же не рекомендуется, поскольку для этого нужно очень хорошо разбираться во внутреннем устройстве шаблонов на платформах WPF и Silverlight и уметь аккуратно обращаться с ними. Ведь если вы проявите небрежность, то можете очень легко удалить те части копии шаблона, которые требуются для нормального функционирования оформляемого с его помощью элемента управления!

Итак, создайте новый проект приложения на платформе WPF, присвоив ему имя `WpfTemplatesWithBlend`. А затем разместите ради простоты рассматриваемого здесь примера один элемент управления типа `Button` на монтажном столе исходного окна типа `Window`.

Пример. В данном примере рассматривается проект на платформе WPF. Если же вы разрабатываете свои проекты на платформе Silverlight, то формируемая в коде XAML разметка будет совсем иной, хотя основные приемы правки шаблонов остаются прежними.

Щелкните правой кнопкой мыши на вновь созданном элементе пользовательского интерфейса и выберите команду Edit Template⇨Edit a Copy (Править шаблон⇨Править копию) из всплывающего контекстного меню, как показано на рис. 5.25.

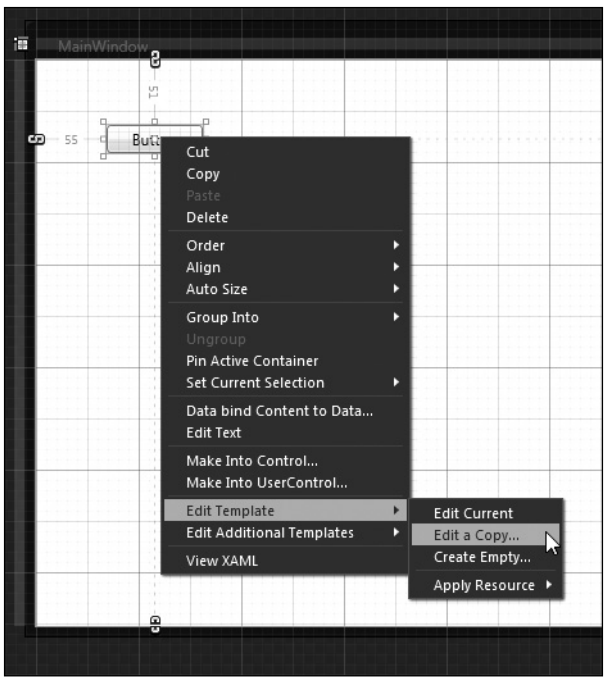


Рис. 5.25. Создание копии используемого по умолчанию шаблона элемента управления

После выбора упомянутой выше команды из контекстного меню откроется диалоговое окно Create Template Resource (Создание ресурса для шаблона), аналогичное диалоговому окну Create Style Resource (Создание ресурса стиля), представленному ранее на рис. 5.9. Присвойте новому стилю имя `rawButtonTemplate` (исходный шаблон кнопки) и сохраните его в качестве ресурса, применяемого на уровне приложения.

Исследование свойств стиля в используемом по умолчанию шаблоне

Откройте файл разметки приложения `App.xaml` в редакторе XAML и обратите внимание на то, что в элементе разметки `<Style>` по умолчанию устанавливается целый ряд значений таких общих для всех элементов управления свойств, как `Background` (Фон), `BorderBrush` (Кисть обводки границы), `Padding` (Заполнение) и пр. Но значения, присваиваемые этим элементам, не являются жестко запрограммированными. Напротив, они обнаруживаются во время выполнения в зависимости от количества заранее определенных ресурсов, причем одни из этих ресурсов определены в контейнере ресурсов приложения, а другие зависят от системных настроек. В приведенной ниже разметке дается частичное определение стиля `rawButtonTemplate`.

```
<Style x:Key="rawButtonTemplate" TargetType="{x:Type Button}">
  <Setter Property="FocusVisualStyle"
        Value="{StaticResource ButtonFocusVisual}"/>
  <Setter Property="Background"
        Value="{StaticResource ButtonNormalBackground}"/>
  <Setter Property="BorderBrush"
        Value="{StaticResource ButtonNormalBorder}"/>
  <Setter Property="BorderThickness" Value="1"/>
  <Setter Property="Foreground"
        Value="{DynamicResource
              {x:Static SystemColors.ControlTextBrushKey}}"/>
  <Setter Property="HorizontalContentAlignment" Value="Center"/>
  <Setter Property="VerticalContentAlignment" Value="Center"/>
  <Setter Property="Padding" Value="1"/>
...
</Style>
```

Если требуется изменить любые из приведенных выше установок основных свойств, можете сделать это непосредственно на монтажном столе. Перейдя в режим конструирования, выберите свойство `Style` редактируемого шаблона по навигационной цепочке в левом верхнем углу рабочего пространства монтажного стола (рис. 5.26).

После этого воспользуйтесь панелью Properties обычным образом, чтобы изменить значения выбранных свойств. Делается это аналогично видоизменению простого стиля, в котором вообще отсутствуют свойства шаблона элемента управления.

Исследование самого шаблона

Помимо исследованных выше установок простых свойств, рассматриваемый здесь стиль содержит элемент разметки `<ControlTemplate>`, в котором определяется целый ряд подчиненных элементов. Прежде всего, вы обнаружите в нем элемент, связывающий используемый по умолчанию шаблон с официально принятой в корпорации Microsoft "темой для кнопок" под названием `ButtonChrome`, как показано ниже.

```
<ControlTemplate TargetType="{x:Type Button}">
  <Microsoft_Windows_Themes:ButtonChrome x:Name="Chrome"
    BorderBrush="{TemplateBinding BorderBrush}"
    Background="{TemplateBinding Background}"
    RenderMouseOver="{TemplateBinding IsMouseOver}"
    RenderPressed="{TemplateBinding IsPressed}"
    RenderDefaulted="{TemplateBinding IsDefaulted}"
```

```
    SnapsToDevicePixels="true">
    <ContentPresenter
      HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
      Margin="{TemplateBinding Padding}"
      RecognizesAccessKey="True"
      SnapsToDevicePixels="{TemplateBinding SnapsToDevicePixels}"
      VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>
  </Microsoft_Windows_Themes:ButtonChrome>
...
</ControlTemplate>
```

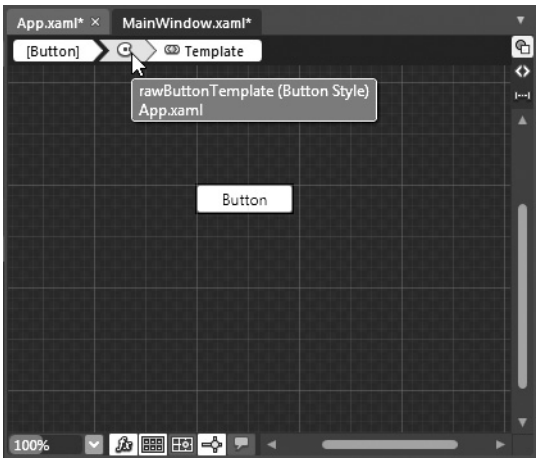


Рис. 5.26. Выбор свойств стиля из используемого по умолчанию шаблона для последующего редактирования

Обратите внимание на то, что целому ряду приведенных выше свойств присваивается значение, возвращаемое из расширения разметки `{TemplateBinding}`. Эта конкретная лексема XAML оказывается очень полезной при построении шаблонов, поскольку она позволяет связать значения свойств, установленных с помощью стиля в элементе управления, с соответствующими свойствами в самом шаблоне. Обратите также внимание на то, что и в данном шаблоне используется элемент разметки `<ContentPresenter>`, допускающий применение произвольного содержимого в конкретном стиле оформления.

В используемом по умолчанию шаблоне кнопки определяется также целый ряд триггеров на платформе WPF. На первый взгляд, триггеры в этом шаблоне не особенно нужны, но на самом деле они организуют обращение к дополнительным ресурсам, встраиваемым в библиотеки на платформе WPF, для получения более полной и подробной информации о том, как нужно визуализировать данный элемент управления.

```
<ControlTemplate.Triggers>
  <Trigger Property="IsKeyboardFocused" Value="true">
    <Setter Property="RenderDefaulted" TargetName="Chrome" Value="true"/>
  </Trigger>
  <Trigger Property="ToggleButton.IsChecked" Value="true">
    <Setter Property="RenderPressed" TargetName="Chrome" Value="true"/>
  </Trigger>
  <Trigger Property="IsEnabled" Value="false">
    <Setter Property="Foreground" Value="#ADADAD"/>
  </Trigger>
</ControlTemplate.Triggers>
```

Примечание. В рассматриваемой здесь копии шаблона имеется также пустая разметка для описания диспетчера Visual State Manager на платформе .NET 4.0. О роли данного компонента в построении шаблонов речь пойдет ниже.

Инструменты, применяемые для правки копии шаблона

А теперь, когда стала понятнее структура используемого по умолчанию шаблона, покажем, как пользоваться различными средствами Expression Blend для его правки. Как пояснялось выше, можно щелкнуть на области `Style` в навигационной цепочке (в левом верхнем углу рабочего пространства монтажного стола), чтобы внести изменения в основные свойства стиля. Но если требуется углубиться в такие детали построения шаблона, как, например, элемент разметки `<ContentPresenter>`, то придется выбрать элемент пользовательского интерфейса, оформляемый по данному шаблону, на панели Objects and Timeline или непосредственно на монтажном столе. Как показано на рис. 5.27, каждый атрибут элемента разметки `<ControlTemplate>` может быть выбран по отдельности для последующей правки.

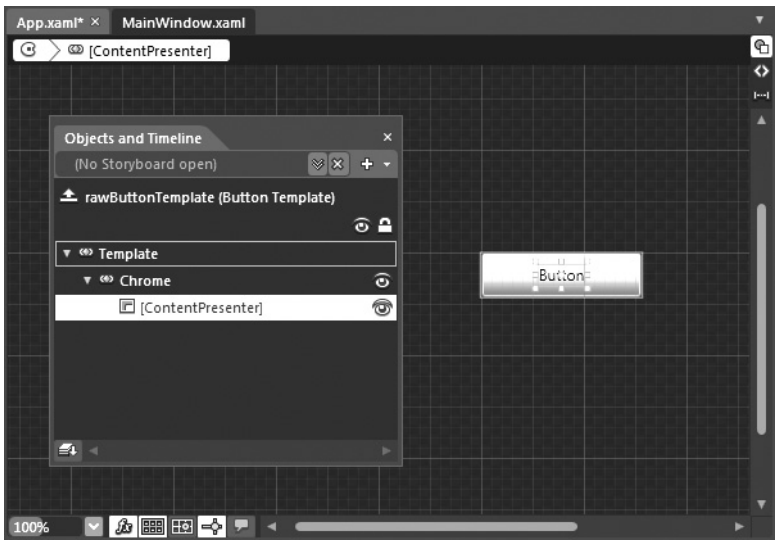


Рис. 5.27. На панели Objects and Timeline можно выбрать нужную часть шаблона для последующей правки

В рассматриваемом здесь примере проекта совсем не обязательно изменять установки в данной копии шаблона оформления кнопки. Дело в том, что в этом шаблоне содержится слишком много жестко закодированных установок, привязывающих оформление элемента управления к внешнему виду, стандартному для программных продуктов корпорации Microsoft. Вы, конечно, вольны внести изменения в этот стандартный внешний вид по своему усмотрению, но сделать это будет совсем не просто. Впрочем, далее будет представлен другой, более простой способ добиться желаемого результата в оформлении элементов пользовательского интерфейса по специальным шаблонам.

Попутно следует также заметить, что панель Triggers можно использовать в Expression Blend для ввода, удаления или видоизменения отдельных триггеров на платформе WPF. В качестве примера на рис. 5.28 показаны текущие установки триггеров в

рассматриваемой здесь копии шаблона кнопки. Но и в этом случае не стоит особенно вдаваться в детали, поскольку в следующем примере будет продемонстрирован намного более простой способ правки специальных шаблонов в среде Expression Blend IDE.

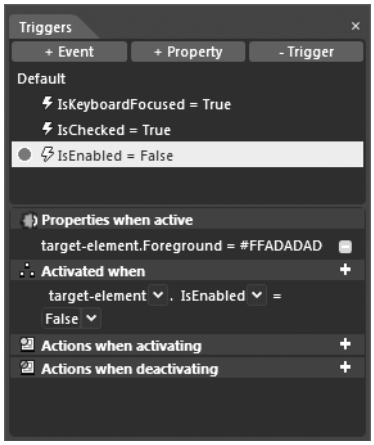


Рис. 5.28. На панели Triggers имеется возможность изменять триггеры, устанавливаемые в текущей копии шаблона

Исходный код. Исходный код примера проекта `WpfTemplatesWithBlend` находится в папке `Ch 5 Code` загружаемого архива примеров проектов к данной книге.

Создание стилизованного шаблона из графики

Как пояснялось выше, всякая попытка видоизменить копию существующего шаблона — дело безнадежное, поскольку это весьма трудоемкий процесс — даже в Expression Blend. Правда, имеется другой, более простой способ создания шаблонов с помощью команды меню Tools⇨Make Into Control (Сервис⇨Преобразовать в элемент управления). Действие этой команды опирается на то обстоятельство, что большинство специальных шаблонов создается на основании существующей графики.

Допустим, что доступными в Expression Blend инструментами была создана идеальная графика или же она была импортирована из Expression Design, как пояснялось в главе 2. Эту графику можно выбрать в качестве отправной точки для создания нового шаблона элемента управления на платформе WPF или Silverlight. А после этого во вновь созданный шаблон можно добавить триггеры, чтобы внедрить визуальные подсказки. Преимущество такого подхода, в отличие от правки копии существующего шаблона, заключается в том, что разработку шаблона можно начинать практически с чистого листа, не особенно вдаваясь в многочисленные установки внешнего вида, стандартного для программных продуктов корпорации Microsoft. Ниже будет показано, как это делается.

Создание исходной графики

Прежде всего создайте новый проект приложения на платформе WPF, присвоив ему имя `StyleTemplateFromGraphic`. Затем создайте произвольную геометрическую форму доступными в Expression Blend инструментами рисования (Pen или Pencil) или же выберите готовую геометрическую форму из категории Shapes в библиотеке ресурсов. В качестве примера на рис. 5.29 показана выбранная форма звезды, в свойстве `PointCount` (Число оконечностей) которой установлено значение **8**.

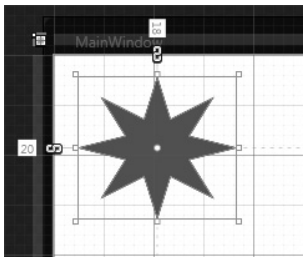


Рис. 5.29. Исходная графика, преобразуемая далее в шаблон элемента управления

Перед тем как продолжить работу над данным примером проекта, уделите немного времени анализу разметки, автоматически сформированной в коде XAML. В этой разметке вы обнаружите более или менее подробное описание созданной геометрической формы, как показано ниже, хотя это зависит от самой геометрической формы и способа ее построения.

```
<ed:RegularPolygon Fill="#FF5050B4" HorizontalAlignment="Left"
   Height="113" InnerRadius="0.47211" Margin="20,18,0,0" PointCount="8"
   Stretch="Fill" Stroke="Black" VerticalAlignment="Top" Width="126"/>
```

Извлечение стилизованного шаблона

Выберите сначала вновь созданную геометрическую форму на монтажном столе, а затем команду меню Tools⇨Make Into Control[8]. Или щелкните правой кнопкой мыши на геометрической форме и выберите команду Make Into Control из всплывающего контекстного меню (рис. 5.30).

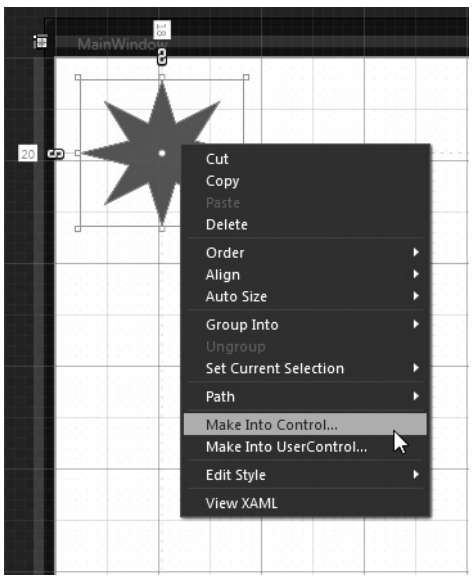


Рис. 5.30. Выбор команды меню Make Into Control в среде Expression Blend IDE

[8] Ни в коем случае не выбирайте команду меню Tools⇨Make Into UserControl, поскольку у нее совсем иное назначение, как поясняется ниже, в разделе "Формирование объектов типа `UserControl` в Expression Blend".

В открывшемся диалоговом окне вы сможете указать имя нового стилизуемого шаблона, место его сохранения как нового ресурса, а самое главное — целевой элемент управления. Для целей рассматриваемого здесь примера определите новый шаблон как ресурс, называемый `starButtonStyle` (стиль кнопки в форме звезды) и применяемый на уровне приложения, а в качестве его целевого элемента управления — объект типа `Button` (рис. 5.31).

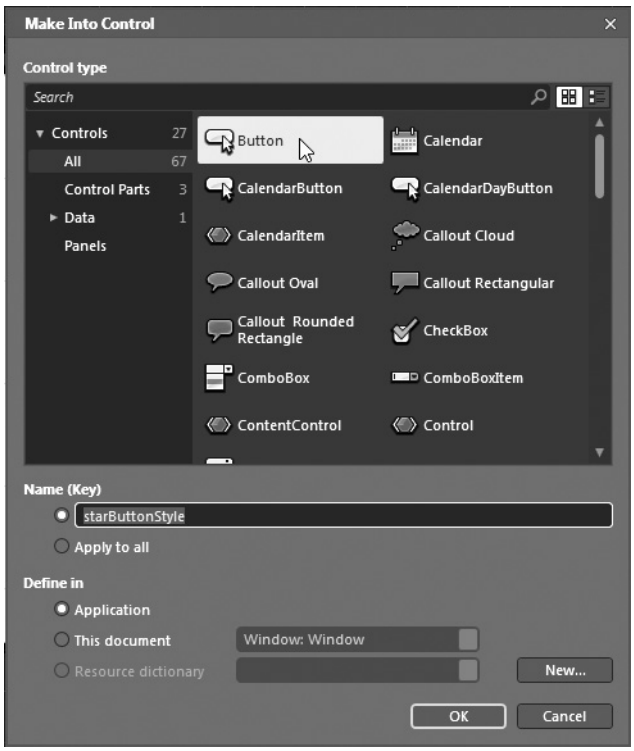


Рис. 5.31. Создание нового шаблона кнопки из геометрической формы звезды

Как только вы щелкнете на кнопке OK, новый стиль станет доступным для правки (рис. 5.32).

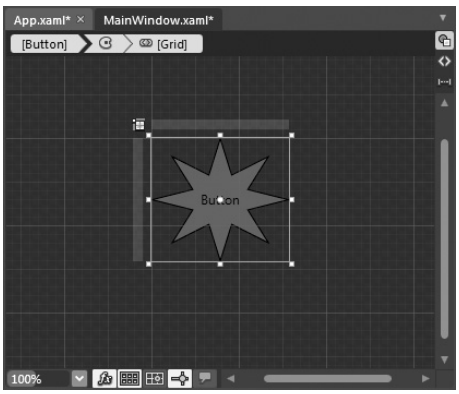


Рис. 5.32. Извлеченный стилизованный шаблон

Проанализировав разметку, автоматически сформированную в коде XAML, вы найдете ее довольно знакомой, поскольку у вас уже имеется некоторый опыт создания шаблона элемента управления вручную ранее в этой главе. В качестве примера ниже приведена разметка, сформированная в коде XAML и сохраненная в файле `App.xaml` при извлечении шаблона из геометрической формы звезды.

```
<Style x:Key="starButtonStyle" TargetType="{x:Type Button}">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type Button}">
        <Grid>
          <ed:RegularPolygon Fill="#FF5050B4" InnerRadius="0.47211"
              PointCount="8" Stretch="Fill" Stroke="Black"/>
          <ContentPresenter HorizontalAlignment=
                "{TemplateBinding HorizontalContentAlignment}"
              RecognizesAccessKey="True"
              SnapsToDevicePixels=
                "{TemplateBinding SnapsToDevicePixels}"
              VerticalAlignment=
                "{TemplateBinding VerticalContentAlignment}"/>
        </Grid>
      <ControlTemplate.Triggers>
        <Trigger Property="IsFocused" Value="True"/>
        <Trigger Property="IsDefaulted" Value="True"/>
        <Trigger Property="IsMouseOver" Value="True"/>
        <Trigger Property="IsPressed" Value="True"/>
        <Trigger Property="IsEnabled" Value="False"/>
      </ControlTemplate.Triggers>
     </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Как видите, в свойстве `TargetType` нового стиля задан целевой элемент управления типа `Button`. Кроме того, в автоматически сформированном элементе разметки `<ControlTemplate>` определен элемент `<ContentPresenter>`, а также ряд пустых триггеров на платформе WPF, которые будут заполнены далее. Обратите внимание на то, каким образом исходная геометрическая форма была удивительным образом превращена в элемент разметки `<Button>`, в котором применяется новый стиль! А при анализе содержимого файла `MainWindow.xaml` вам встретится следующая разметка.

```
<Button Content="Button" HorizontalAlignment="Left" Height="113"
      Margin="20,18,0,0" Style="{DynamicResource starButtonStyle}"
      VerticalAlignment="Top" Width="126"/>
```

Не такой уж и плохой результат всего лишь для нескольких действий мышью, не так ли? Теперь можете обработать любое событие, связанное с кнопкой, внести поправки в остальные свойства и выполнить ряд других полезных операций.

Построение стилизованного шаблона спискового окна

Прежде чем вводить интерактивные средства в рассматриваемый здесь шаблон, необходимо определить еще один шаблон из вспомогательной геометрической формы. С этой целью перейдите на монтажный стол исходного окна типа `Window` и нарисуйте любую вспомогательную геометрическую форму. В качестве примера на рис. 5.33 (а также на цветной вклейке) показана геометрическая форма прямоугольника с волнистыми сторонами, нарисованная инструментом Pencil.

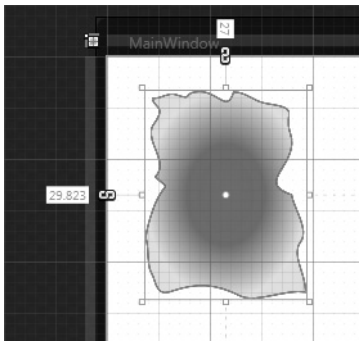


Рис. 5.33. Вспомогательная геометрическая форма, нарисованная инструментом Pencil

Выберите сначала вновь созданный графический элемент, а затем команду меню Make Into Control. Но на этот раз определите на уровне приложения стиль, именуемый `wigglyListBoxStyle` (стиль волнистого списочного окна), а в качестве его цели — элемент управления типа `ListBox`, как показано на рис. 5.34.

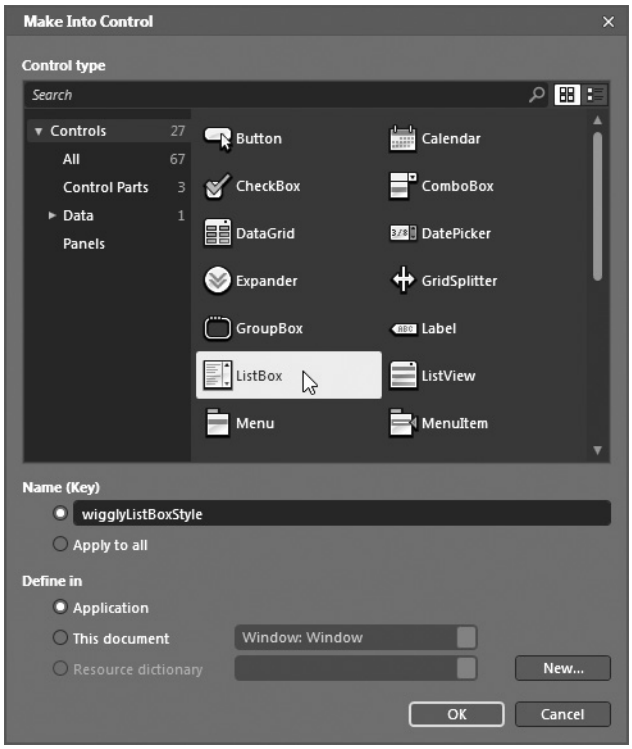


Рис. 5.34. Выбор элемента управления типа `ListBox` в качестве целевого для нового шаблона

Во вновь созданном шаблоне используется элемент разметки `<ItemsPresenter>` вместо элемента `<ContentPresenter>`. Ведь у элементов управления типа `ListBox`, как правило, имеется ряд отдельных элементов списка, которые можно просматривать в

режиме прокрутки. Ниже приведен фрагмент разметки, автоматически сформированной в коде XAML при создании нового шаблона.

```
<Style x:Key="wigglyListBoxStyle" TargetType="{x:Type ListBox}">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type ListBox}">
        ...
        <ScrollViewer>
          <ItemsPresenter/>
        </ScrollViewer>
        ...
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Прежде чем вводить элементы списка во вновь определенный элемент управления типа `ListBox` на монтажном столе главного окна приложения, необходимо обновить элемент разметки `<ItemsPresenter>`, чтобы обеспечить выравнивание каждого элемента списка типа `ListItem` по центру. С этой целью откройте окно визуального конструктора для правки нового стиля `wigglyListBoxStyle` и его шаблона, а затем выберите элемент разметки `<ItemsPresenter>` на панели Objects and Timeline, как показано рис. 5.35.

Далее воспользуйтесь панелью Properties, чтобы установить режим выравнивания по центру (**`Center`**) в свойствах `HorizontalAlignment` и `VerticalAlignment` (рис. 5.36).

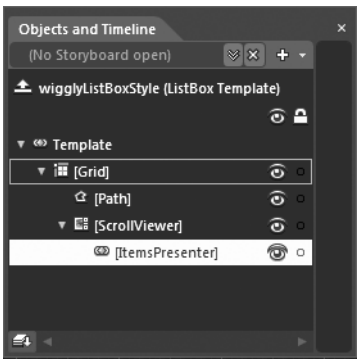


Рис. 5.35. Выбор элемента шаблона `ItemsPresenter` для правки

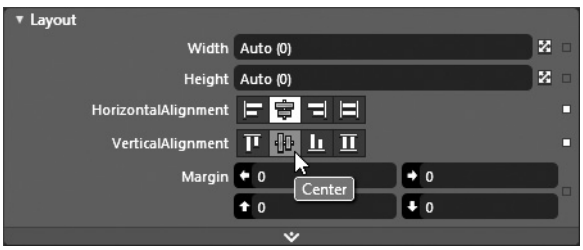


Рис. 5.36. Центровка элементов списка в элементе шаблона `ItemsPresenter`

Сохраните внесенные изменения и заполните несколькими объектами типа `ListBoxItem` элемент управления типа `ListBox`, находящийся на монтажном столе исходного окна типа `Window`. Как пояснялось в главе 4, для этой цели можно воспользоваться редактором, доступным с помощью кнопки ... в области Common Properties на панели Properties, или же просто набрать разметку в редакторе XAML. В качестве примера ниже приведена разметка, в которой описывается списочное окно с тремя введенными элементами и дополнительными установками его свойств.

```
<ListBox HorizontalAlignment="Left" Height="162.101"
       Margin="29.823,27,0,0"
       Style="{DynamicResource wigglyListBoxStyle}"
       VerticalAlignment="Top" Width="125.177"
       FontSize="16" FontStyle="Italic">
  <ListBoxItem Content="Item One"/>
```

```
  <ListBoxItem Content="Item Two"/>
  <ListBoxItem Content="Item Three"/>
</ListBox>
```

Запустите свое приложение на выполнение и убедитесь сами в том, что вы создали для элемента управления типа `ListBox` ни на что не похожий шаблон! Теперь можете выбрать каждый элемент списка по отдельности, как и предполагалось и как показано на рис. 5.37, а также на цветной вклейке.

Поэкспериментируйте с другими средствами преобразования графики в элементы управления. Применяя рассмотренные выше способы, вы быстро сформируете специальные шаблоны для любого числа элементов управления. Более того, сможете поправить и усовершенствовать извлеченный шаблон (а также содержащийся в нем стиль) самыми разными способами, чтобы придать элементам управления совершенно особый внешний вид.

Итак, упражняйтесь в создании шаблонов из геометрических форм столько, сколько душа пожелает. А по завершении перейдите к следующему разделу, посвященному вводу интерактивных средств в шаблон элемента управления с помощью триггеров на платформе WPF.

Рис. 5.37. Специальные шаблоны элементов управления в действии

Ввод интерактивных средств в шаблон с помощью триггеров свойств на платформе WPF

В своем нынешнем состоянии созданные выше шаблоны представляют собой статические изображения, не изменяющие свой внешний вид, когда пользователь, например, помещает курсор в пределах геометрической формы элемента управления, определяемого шаблоном, щелкает на нем кнопкой мыши или выполняет иные действия. Ранее в этой главе для организации подобного взаимодействия с пользователем вам пришлось вручную ввести в шаблон ряд триггеров, доступных на платформе WPF. А теперь настало время обратиться к средствам Expression Blend, упрощающим данный процесс. Итак, перейдите к панели Resources и выберите стиль `starButtonStyle` для последующей правки, как показано на рис. 5.38.

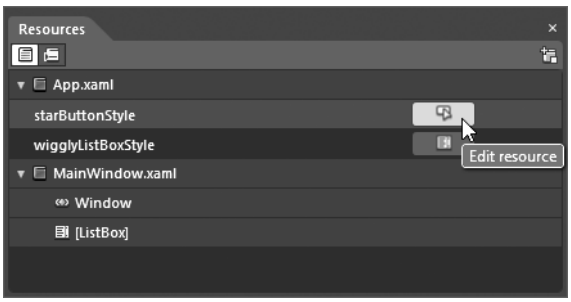


Рис. 5.38. Выбор стиля для правки

Выберите текущий внутренний шаблон элемента управления по навигационной цепочке в левом верхнем углу редактора стилей (рис. 5.39).

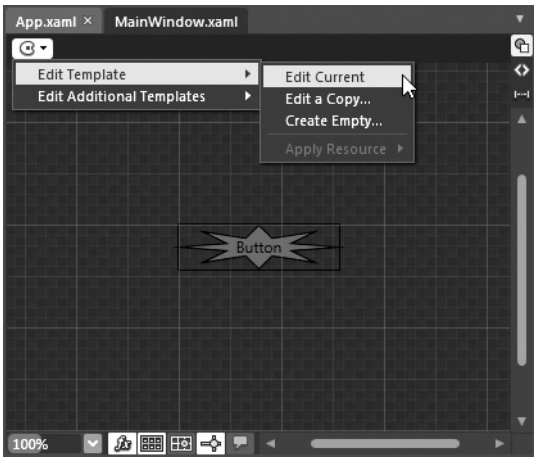


Рис. 5.39. Выбор внутреннего шаблона элемента управления по навигационной цепочке в редакторе стилей

Выбрав шаблон, можете активизировать панель Triggers в среде Expression Blend IDE. (Напомним: если вы не можете найти нужную панель, воспользуйтесь меню Window, из которого открывается и скрывается искомая панель.) Как показано на рис. 5.40, в шаблоне `starButtonStyle` имеются пустые триггеры для пяти свойств на платформе WPF: `IsFocused`, `IsDefaulted`, `IsMouseOver`, `IsPressed`, и `IsEnabled`.

Ваша задача — воспользоваться панелью Triggers, чтобы определить внешний вид элемента управления, оформляемого по данному шаблону, когда каждое из перечисленных выше свойств имеет логическое значение `true` (напомним, что эти свойства не устанавливаются в элементе управления при наступлении соответствующего события). Итак, выберите свойство `IsMouseOver`. В итоге монтажный стол с выбранным на нем шаблоном перейдет в режим регистрации, как в редакторе анимации (см. главу 3).

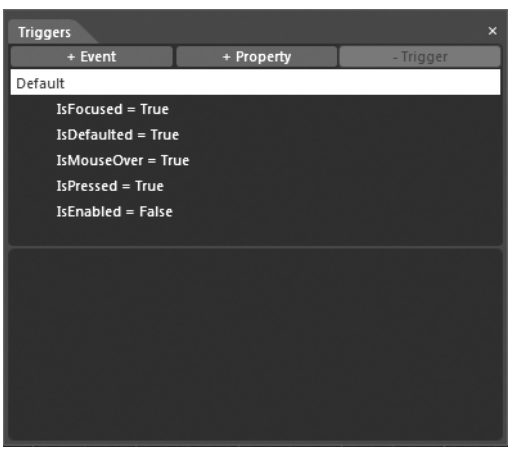


Рис. 5.40. Пустые триггеры в шаблоне, требующие наполнения интерактивными средствами

А теперь воспользуйтесь панелью Properties, чтобы внести в шаблон ряд изменений, которые должны проявляться при наведении курсора на элемент управления, определяемый этим шаблоном. Выберите объект геометрической формы на панели Objects and

Timeline и подберите другой цвет заполнения этой формы в свойстве `Fill`. После этого можете выйти из режима регистрации, щелкнув на красной кнопке остановки в левом верхнем углу монтажного стола или рядом с триггером, редактируемым на панели Triggers. В любом случае после выхода из режима регистрации панель Triggers будет выглядеть так, как показано на рис. 5.41.

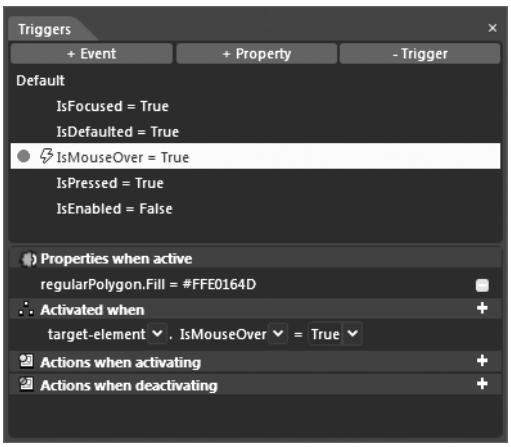


Рис. 5.41. Изменение цвета заполнения геометрической формы при наступлении события `IsMouseOver`

Если вы проанализируете разметку, автоматически сформированную в коде XAML, то обнаружите, что коллекция триггеров в данном шаблоне обновилась следующим образом.

```
<ControlTemplate.Triggers>
  <Trigger Property="IsFocused" Value="True"/>
  <Trigger Property="IsDefaulted" Value="True"/>
  <Trigger Property="IsMouseOver" Value="True">
    <Setter Property="Fill" TargetName="regularPolygon"
          Value="#FFFDFF11"/>
  </Trigger>
  <Trigger Property="IsPressed" Value="True"/>
  <Trigger Property="IsEnabled" Value="False"/>
</ControlTemplate.Triggers>
```

А теперь измените таким же точно способом внешний вид геометрической формы (цвет, размеры или иные свойства), когда в свойстве `IsPressed` устанавливается логическое значение `true`, т.е. при наступлении события `IsPressed`. В частности, можете поправить шаблон таким образом, чтобы при наступлении события `IsPressed`, т.е. при нажатии кнопки мыши, когда курсор находится на геометрической форме, определяемой данным шаблоном, размеры этой формы немного сокращались. С этой целью перейдите на вкладку Scale в области Transform на панели Properties и внесите соответствующие коррективы в свойства, определяющие размеры геометрической формы. Ниже приведена разметка, формируемая при этом в коде XAML.

```
<ControlTemplate.Triggers>
...
  <Trigger Property="IsPressed" Value="True">
    <Setter Property="RenderTransform" TargetName="regularPolygon">
      <Setter.Value>
```

```
        <TransformGroup>
          <ScaleTransform ScaleX="0.9" ScaleY="0.9"/>
          <SkewTransform/>
          <RotateTransform/>
          <TranslateTransform/>
        </TransformGroup>
       </Setter.Value>
      </Setter>
    </Trigger>
   <Trigger Property="IsEnabled" Value="False"/>
</ControlTemplate.Triggers>
```

Аналогичным образом можно внести и другие коррективы в оставшиеся свойства `IsDefaulted`, `IsEnabled` и `IsFocused` на панели Triggers, но общий принцип должен быть вам ясен. А теперь запустите свое приложение на выполнение, наведите курсор на кнопку в форме звезды и щелкните на ней. Состояние этой кнопки должно измениться!

Дополнительные ресурсы для изучения триггеров на платформе WPF

В этой главе было показано, как пользоваться панелью Triggers в среде Expression Blend IDE для внедрения визуальных подсказок в специальный шаблон. К редактору этой панели вам придется еще не раз обращаться для правки своих шаблонов элементов управления на платформе WPF. Но теперь самое время сменить тему и рассмотреть назначение диспетчера визуальных состояний, который считается более предпочтительным (и единственным на платформе Silverlight) средством для внедрения визуальных подсказок в шаблоны посредством разметки. Поэтому если у вас есть интерес к дальнейшему и более углубленному изучению триггеров на платформе WPF и желание проработать дополнительный учебный материал по шаблонам, обратитесь к разделу “Стилевое оформление элементов управления, поддерживающих шаблоны” (Styling a control that supports templates) руководства пользователя Expression Blend (рис. 5.42). В этом разделе содержатся дополнительные сведения о среде триггеров на платформе WPF.

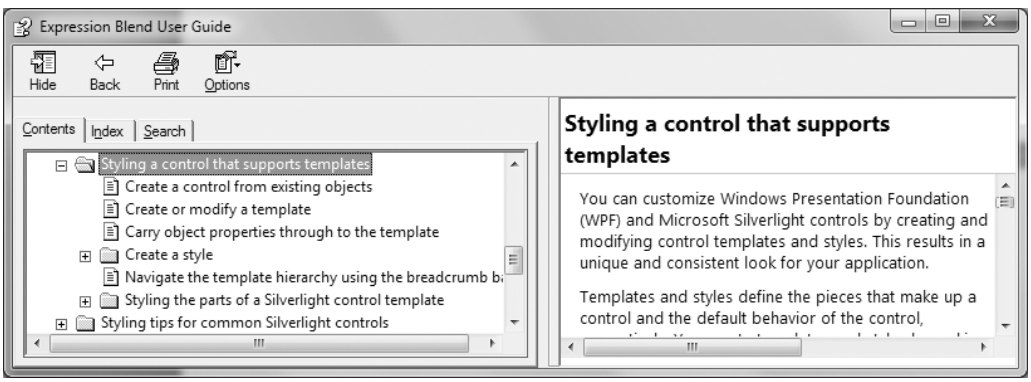


Рис. 5.42. В руководстве пользователя Expression Blend предоставляется немало учебного материала по триггерам на платформе WPF

Исходный код. Исходный код примера проекта `StyleTemplateFromGraphic` находится в папке `Ch 5 Code` загружаемого архива примеров проектов к данной книге.

Создание шаблонов средствам прикладного интерфейса Silverlight API

Как уже не раз упоминалось в этой главе, в прикладном интерфейсе Silverlight API традиционные для платформы WPF триггеры находят весьма ограниченную поддержку, но того же самого конечного результата можно добиться и с помощью диспетчера визуальных состояний (VSM — Visual State Manager). Напомним также, что после выпуска версии платформы .NET 4.0 в прикладной интерфейс WPF API была внедрена собственная версия диспетчера VSM, а следовательно, в прикладных интерфейсах на обеих платформах теперь поддерживается единообразный способ внедрения визуальных подсказок в специальные шаблоны и стили. Принимая во внимание это обстоятельство, в следующем примере проекта вам предстоит создать специальный шаблон средствами прикладного интерфейса Silverlight API, но имейте в виду, что рассматриваемые далее способы можно применять и в проектах, разрабатываемых на платформе WPF.

По существу, в данном примере будет воссоздан рассмотренный ранее шаблон круглой кнопки, но на этот раз — с помощью диспетчера VSM, а не триггеров, доступных на платформе WPF. Прежде всего создайте новый проект приложения Silverlight, присвоив ему имя `SLControlTemplate`. Затем нарисуйте произвольную форму на монтажном столе инструментом Pen или Pencil либо выберите готовую форму из категории Shapes в библиотеке ресурсов. И в этом случае можете выбрать простую форму эллипса, заполнив ее зеленым цветом. Щелкните правой кнопкой мыши на этой геометрической форме и выберите, как и прежде, команду Make Into Control из контекстного меню (рис. 5.43).

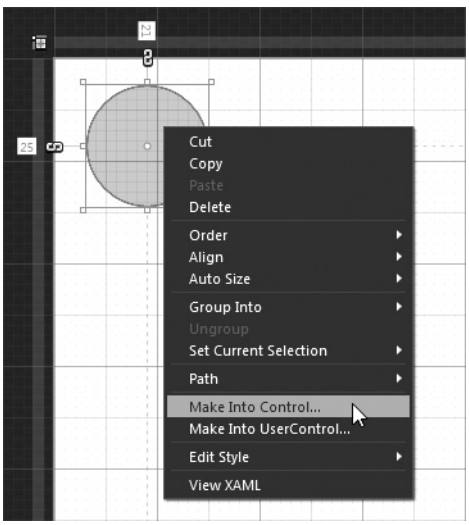


Рис. 5.43. Команда Make Into Control поддерживается и в проектах на платформе Silverlight

Определите новый ресурс, именуемый `SilverlightRoundButtonTemplate`, доступный на уровне приложения и предназначенный для стилевого оформления элементов управления типа `Button`, в открывшемся диалоговом окне. Как только вы щелкнете на кнопке OK, появится окно визуального конструктора круглой кнопки, оформляемой по новому шаблону. Этот шаблон можно, как и прежде, поправить и видоизменить на панели Properties, Objects and Timeline и в редакторе, выбираемом по навигационной цепочке (рис. 5.44).

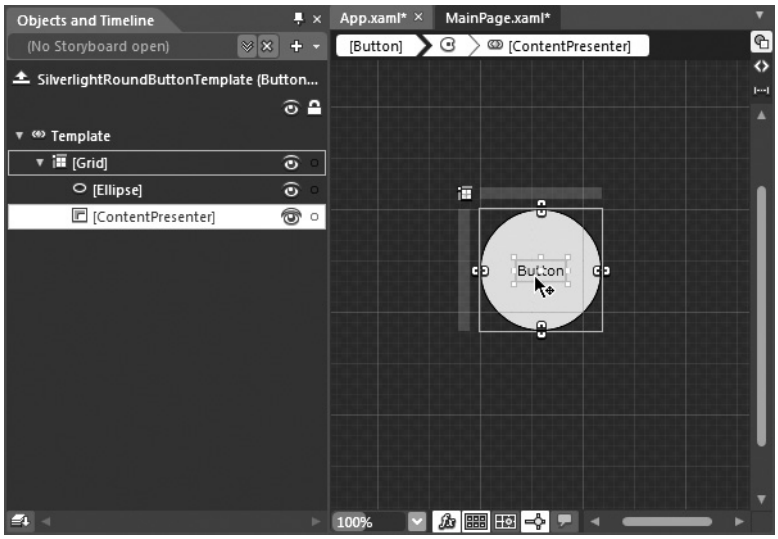


Рис. 5.44. Новый шаблон на платформе Silverlight готов к правке

Но если вы внимательно проанализируете разметку, автоматически сформированную в коде XAML, то не обнаружите в ней ни одного устанавливаемого по умолчанию триггера. Дело в том, что внедрение визуальных подсказок в шаблон на платформе Silverlight осуществляется на панели States (Состояния).

Работа с диспетчером VSM на панели States

Подобно панели Triggers в проектах, разрабатываемых на платформе WPF, панель States служит для внесения изменений визуального состояния в выбранный шаблон, но делается это совершенно по-другому. Если окно с монтажным столом активизировано для выбранного шаблона, разрабатываемого на платформе Silverlight, перейдите к панели States, обратив внимание на исходные установки (рис. 5.45).

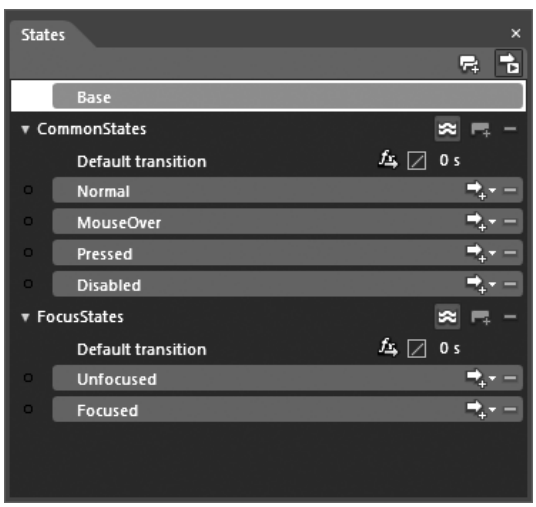


Рис. 5.45. Вид панели States в Expression Blend

По умолчанию в пустом шаблоне вообще отсутствуют визуальные состояния. Но как только вы перейдете к панели States, то обнаружите на ней ряд общедоступных визуальных состояний (`Normal`, `MouseOver`, `Pressed`, `Disabled`, `Unfocused` и `Focused`), разделенных на две группы: `CommonStates` (Общие состояния) и `FocusStates` (Состояния фокуса)[9]. Как будет показано далее, к этим двум группам можно также добавить группы специальных состояний.

Эти группы стандартных состояний удобны тем, что в элементах управления на платформах WPF и Silverlight автоматически реализуются переходы в эти состояния, когда происходит соответствующее действие. Иными словами, если пользователь щелкнет кнопкой мыши на элементе управления, последний автоматически перейдет в состояние `Pressed` (Нажато). Если же элемент окажется вне логического фокуса и каких-либо действий мышью, то он автоматически перейдет в состояние `Normal` (Обычное) и т.д.

Для большей наглядности переход элемента управления в состояние `MouseOver` (Наведен курсор мыши) можно дополнить соответствующей анимацией, описываемой в разметке. С этой целью выберите состояние `MouseOver` на панели States. В итоге произойдет автоматический переход в режим регистрации. Как и при создании обычной анимации (см. главу 3), в данном случае вы сможете внести любые изменения в свойства выбранного элемента управления на панели Properties. Что же касается состояния `MouseOver`, то выберите более темный оттенок цвета заполнения формы эллипса в свойстве `Fill`, как показано на рис. 5.46, а также на цветной вклейке к книге.

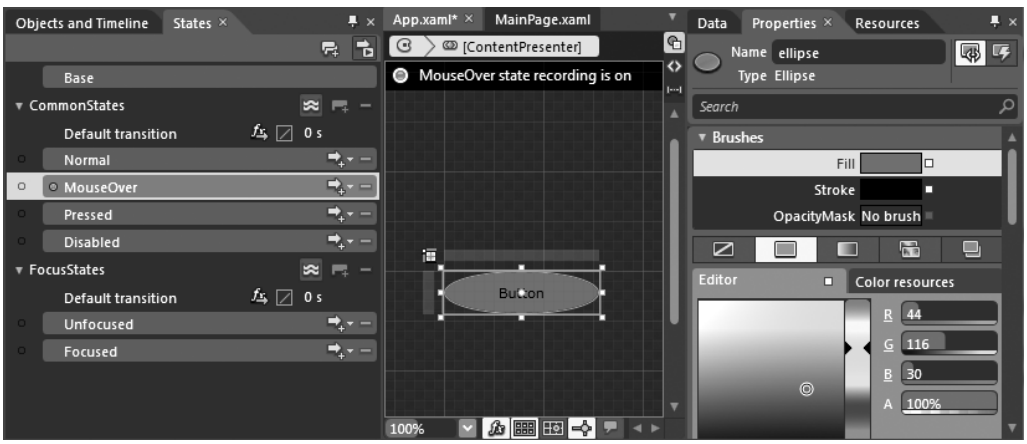


Рис. 5.46. Регистрация изменений, происходящих при наведении курсора мыши на кнопку в форме эллипса

А теперь выберите состояние `Pressed` на панели States. Выполните графическое преобразование формы эллипса таким образом, чтобы размеры соответствующего элемента управления немного сокращались при переходе в данное состояние. С этой целью перейдите к области Transform на панели Properties (напомним, что на вкладке Scale в этой области доступны преобразования масштабированием, как показано на рис. 5.47).

[9] Элементы управления на платформах WPF и Silverlight предназначены для поддержки целого ряда общих состояний, организованных в группы. Это очень удобно, поскольку можно быть уверенным в том, что правки, вносимые в одинаково именуемые состояния, поддерживаются в большинстве элементов управления.

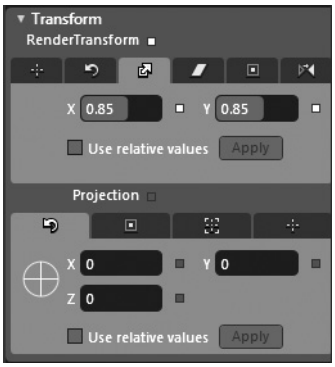


Рис. 5.47. Преобразование формы эллипса после щелчка на кнопке, определяемой этой формой

По завершении выйдите из режима регистрации, щелкнув на красной кнопке остановки в левом верхнем углу монтажного стола (подробнее о работе с редактором анимации см. в главе 3). Запустите свое приложение на выполнение и проверьте внесенные в него изменения. Теперь вы должны наблюдать переходы элемента управления (типа `Button`) в разные визуальные состояния при взаимодействии с ним мышью! А о том, как это происходит, речь пойдет далее.

Просмотр разметки, сформированной в коде XAML

Если вы откроете файл разметки приложения `App.xaml` в редакторе XAML, то заметите, что в элементе разметки `<Style>` появился целый ряд новых инструкций, определяющих изменения, внесенные вами в различные визуальные состояния. Как показано в приведенной ниже разметке, вся логика работы диспетчера VSM заключена в элементе разметки `<VisualStateManager.VisualStateGroups>`, где представлена и учитывается каждая группа стандартных состояний, многие из которых оказываются пустыми. Так, если у отдельного состояния отсутствуют инструкции его визуализации, внешний вид соответствующего элемента управления определяется его текущими настройками.

```
<VisualStateManager.VisualStateGroups>
  <VisualStateGroup x:Name="CommonStates">
    <VisualState x:Name="Normal"/>
    <VisualState x:Name="MouseOver">
      <Storyboard>
        <ColorAnimation Duration="0" To="#FF2C741E"
          Storyboard.TargetProperty="(Shape.Fill).(SolidColorBrush.Color)"
          Storyboard.TargetName="ellipse" d:IsOptimized="True"/>
      </Storyboard>
    </VisualState>
    <VisualState x:Name="Pressed">
      <Storyboard>
        <DoubleAnimation Duration="0" To="0.85"
          Storyboard.TargetProperty=
          "(UIElement.RenderTransform).(CompositeTransform.ScaleX)"
          Storyboard.TargetName="ellipse" d:IsOptimized="True"/>
        <DoubleAnimation Duration="0" To="0.85"
          Storyboard.TargetProperty=
          "(UIElement.RenderTransform).(CompositeTransform.ScaleY)"
          Storyboard.TargetName="ellipse" d:IsOptimized="True"/>
      </Storyboard>
```

```
    </VisualState>
    <VisualState x:Name="Disabled"/>
  </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

Установка времени перехода для групп состояний

Когда происходит переход в определенное состояние, по умолчанию выполняется код в элементе разметки `<Storyboard>`, связанном с этим состоянием. По желанию можете определить новое время перехода для всех состояний из отдельной группы, откорректировав устанавливаемое по умолчанию нулевое значение этого параметра. В качестве примера на рис. 5.48 показано, каким образом устанавливаемое по умолчанию нулевое время перехода изменяется до **2** секунд для всех состояний из группы `CommonStates`.

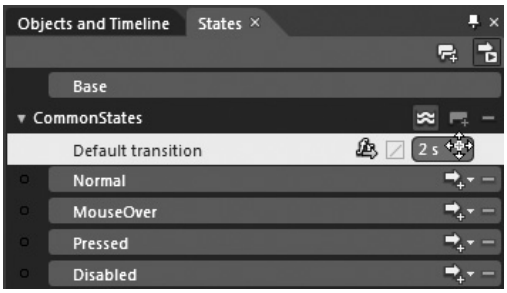


Рис. 5.48. Изменение устанавливаемого по умолчанию времени перехода для всех состояний из выбранной группы

Если вы вновь запустите свое приложение на выполнение, то заметите, что переходы в общие состояния происходят намного медленнее и даже слишком медленно, что, конечно, сделано для большей наглядности! Но вы вольны изменить время перехода в эти состояния по своему усмотрению.

Определение эффектов перехода в разные состояния

Справа от наименования каждой группы состояний на панели States находится небольшая кнопка, обозначенная буквами `fx` и стрелкой, направленной вправо. С помощью этой кнопки можно определить графический эффект, применяемый во время перехода в отдельное состояние из данной группы. Щелкните на данной кнопке в группе состояний `CommonStates` и обратите внимание на появляющиеся варианты выбора эффектов перехода в состояния из этой группы. В качестве примера на рис. 5.49 показано применение эффекта перехода `Smooth Swirl Grid` (Плавное завихрение сетки) для состояний из группы `CommonStates`.

Примечание. Если установлено нулевое время перехода, то анимация эффекта перехода окажется незаметной. Для того чтобы увидеть проявление любого эффекта перехода, следует установить время перехода равным как минимум 1 секунде.

Следует также иметь в виду, что после выбора конкретного эффекта перехода можно поправить отдельные его установки, еще раз щелкнув на кнопке **`fx`**, как показано на рис. 5.50.

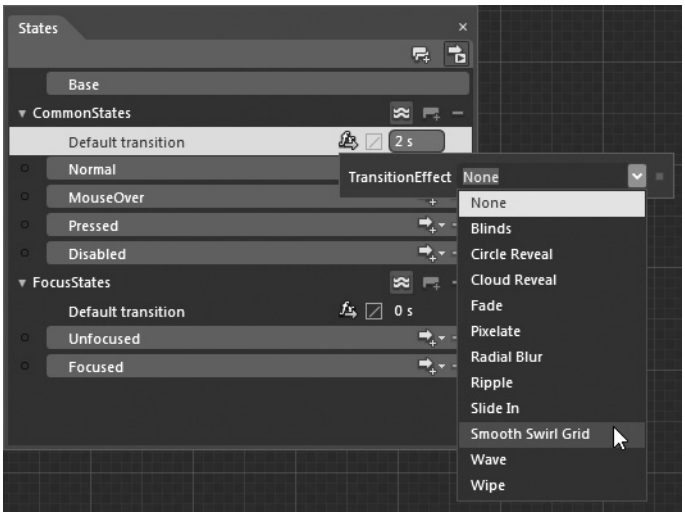


Рис. 5.49. Добавление эффекта перехода

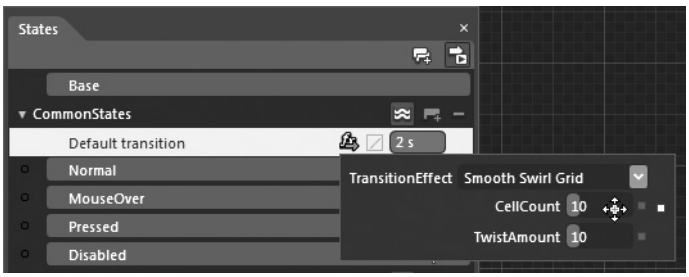


Рис. 5.50. Правка эффекта перехода

Кроме того, отдельную группу состояний можно настроить на применение рассматривавшихся в главе 3 эффектов инерционности движения в анимации переходов между визуальными состояниями. Для этого достаточно выбрать один из вариантов, доступных при активизации кнопки со знаком молнии на пиктограмме, расположенной рядом с кнопкой **fx** (рис. 5.51).

Рекомендуется уделить немного времени экспериментированию с эффектами перехода в разные состояния, изменяя их установки. Ведь на печатной странице просто невозможно воспроизвести подобные эффекты в их динамике[10].

Настройка отдельных переходов

Устанавливая время и настраивая эффекты перехода для отдельной группы состояний, вы должны иметь в виду, что эти установки и настройки распространяются на *все* состояния из данной группы. Так, если вы настраиваете эффект отскакивания для выбранной группы состояний, то должны ясно осознавать, что подобное отскакивание будет происходить независимо от текущего состояния и того состояния, переход в которое должен совершиться. Но, как правило, именно это и требуется. Тем не менее на

[10] Должен признаться, что я потратил больше времени, чем обычно себе позволяю, на экспериментирование с установками эффектов перехода в разные состояния.

панели States предоставляются средства для намного более тонкой настройки переходов в разные состояния.

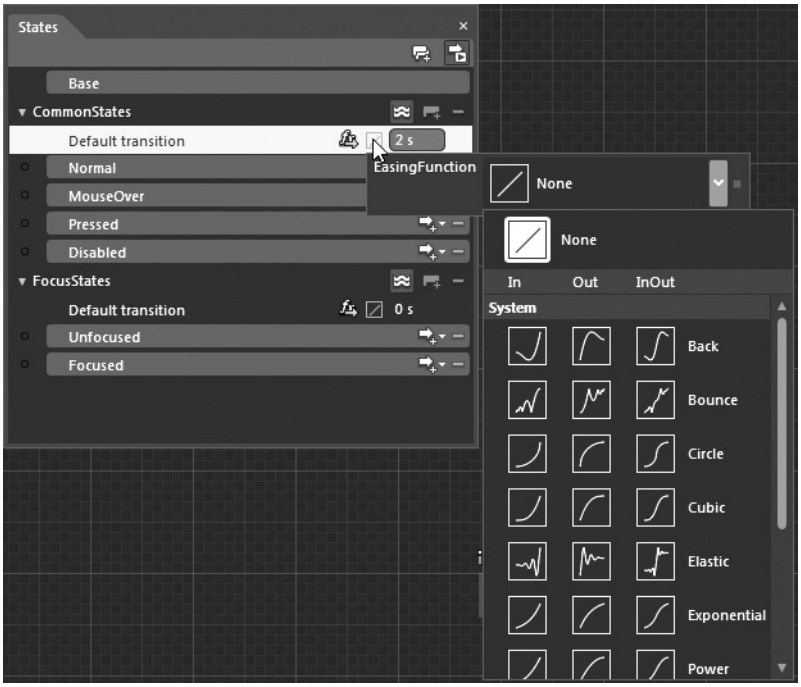


Рис. 5.51. Добавление эффектов инерционности движения в анимацию смены состояний

В качестве примера выберите состояние `Focused` (В фокусе) из группы `FocusedStates` (Состояния в фокусе). Обратите внимание на кнопку справа от наименования этого и других состояний из данной группы. (Она обозначена стрелкой, направленной вправо, и мелким знаком +.) Если щелкнуть на этой кнопке, которая официально называется Add Transition (Добавить переход), отобразится ряд редакторов переходов в состояния. Как показано на рис. 5.52, с помощью этих редакторов можно задавать эффекты перехода из текущего состояния в состояние `Focused`, из состояния `Focused` — в состояние `Unfocused` (Вне фокуса) и обратно в состояние `Focused`, но не спешите задавать какие-либо переходы.

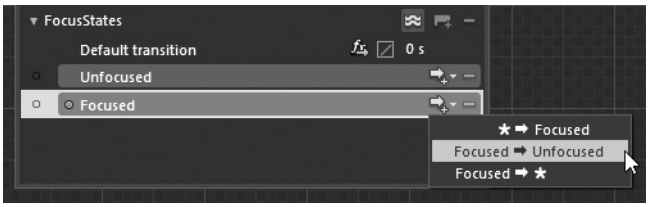


Рис. 5.52. С помощью кнопки Add Transition можно определить особые раскадровки для анимации переходов в отдельные состояния

Примечание. Пиктограмма с изображением звездочки в редакторе переходов обозначает *любое* состояние.

Как только определите особый переход из одного состояния в другое, появится редактор данного *конкретного* перехода, в котором доступны те же самые установки, что и для всей группы состояний. В качестве примера на рис. 5.53 показана настройка перехода из состояния `MouseOver` в состояние `Normal` с упругим эффектом в конце перехода (`Elastic Out`), когда курсор мыши, находящийся на элементе управления, отводится от него.

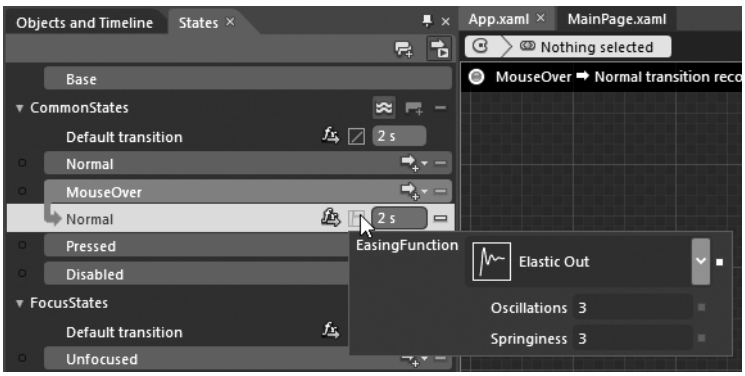


Рис. 5.53. Добавление эффекта упругости к переходу из одного состояния в другое

Краткий обзор специальных состояний

Помимо рассмотренных выше стандартных состояний, имеется также возможность определить особый ряд состояний, характерных для конкретного элемента управления. Потребность в этом может возникнуть в том случае, когда создаваемый шаблон должен вести себя по-разному при наступлении специально предусмотренных и реализованных в коде событий или же событий произвольного ввода, *не* определенных в группах стандартных состояний.

Допустим, шаблон должен обеспечивать особый внешний вид элемента управления, когда пользователь нажимает клавишу <Shift> и щелкает кнопкой мыши на этом элементе. Такого результата можно добиться с помощью групп специально настраиваемых состояний. Для того чтобы показать, каким образом подобная задача решается с помощью диспетчера VSM, перейдем к рассмотрению заключительного в этой главе вопроса, касающегося создания специальных объектов типа `UserControl`.

Исходный код. Исходный код примера проекта `SLControlTemplate` находится в папке `Ch 5 Code` загружаемого архива примеров проектов к данной книге.

Построение классов `UserControl` в Expression Blend

В заключение главы будет разъяснено, каким образом в среде Expression Blend IDE упрощается построение полнофункциональных классов `UserControl`. Вам, возможно, известно, что на платформах WPF и Silverlight понятие *элемента управления* реализуется в классе, инкапсулирующем коллекцию связанных вместе элементов пользовательского интерфейса. В отличие от шаблона элемента управления, применяемого к конкретному элементу управления и определяющего его внешний вид и поведение, специализированный класс `UserControl` представляет собой совершенно новый элемент

пользовательского интерфейса, создаваемый путем агрегирования существующих элементов управления.

Если бы эта книга была посвящена построению полнофункциональных специализированных элементов управления на платформе WPF или Silverlight, в ней непременно были бы рассмотрены такие вопросы, как свойства зависимостей и архитектура перенаправляемых событий. Но поскольку книга посвящена вопросам применения Expression Blend, ниже будет показано, каким образом в данной среде упрощается создание нового элемента управления класса `UserControl`, в котором поддерживается ряд особых визуальных состояний.

В меню Expression Blend имеется команда Make Into UserControl, аналогичная команде Make Into Control и доступная после выбора графического элемента на монтажном столе. Как правило, построение специализированного элемента управления класса `UserControl` начинается со специальной графики. Поэтому для рассмотрения последнего в этой главе примера создайте новый проект приложения WPF (или Silverlight, если угодно), присвоив ему имя `BlendUserControl`.

Примечание. Здесь и далее рассматривается проект на платформе WPF, чтобы показать на его примере особенности работы с диспетчером VSM в данном контексте.

Нарисуйте графический элемент на монтажном столе и щелкните на нем правой кнопкой мыши, чтобы выбрать команду Make Into UserControl из всплывающего контекстного меню (рис. 5.54).

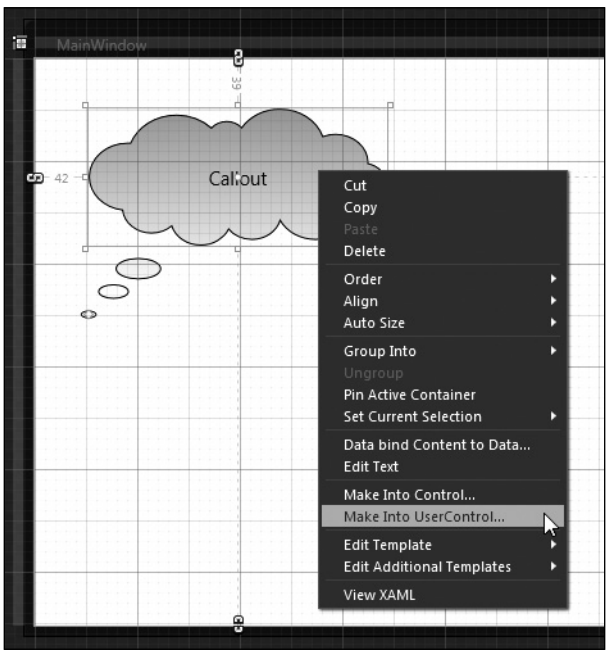


Рис. 5.54. Создание нового элемента управления класса `UserControl`

Присвойте подходящее имя, например `BubbleThoughtsControl` (элемент управления выноской), вновь созданному элементу управления в открывшемся диалоговом окне, как показано на рис. 5.55.

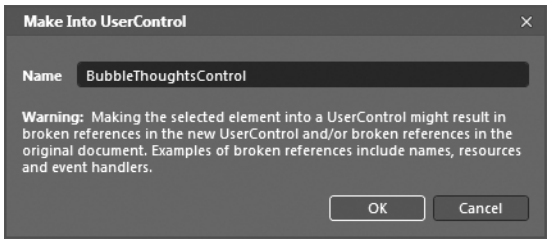


Рис. 5.55. Присвоение имени новому элементу управления класса `UserControl`

Щелкните на кнопке OK и перейдите к панели Projects. Как показано на рис. 5.56, в текущий проект автоматически введен новый XAML-файл и связанный с ним файл исходного кода для нового элемента управления класса `UserControl`.

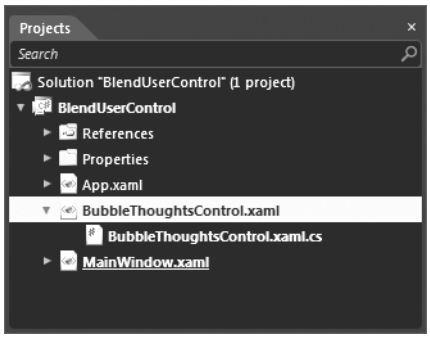


Рис. 5.56. XAML-файл и файл исходного кода, введенные в текущий проект для нового элемента управления класса `UserControl`

Примечание. В данном примере новый элемент управления класса `UserControl` автоматически построен из одного графического элемента. Но многие элементы управления данного класса обычно строятся из коллекции существующих объектов. Так, если выбрать сначала несколько графических элементов на монтажном столе, а затем команду меню Make Into UserControl, то все выбранные элементы будут сгруппированы в новый диспетчер компоновки и размещены в классе `UserControl`.

Если вы проанализируете разметку главного окна типа `Window`, то обнаружите, что исходная графика заменена экземпляром нового класса `UserControl`, как показано ниже[11].

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="clr-namespace:BlendUserControl"
  x:Class="BlendUserControl.MainWindow"
  x:Name="Window"
  Title="MainWindow"
```

[11] Если бы вы создали элементы управления класса `UserControl` вручную, то обратили бы также внимание на то, что все XAML-файлы специальной разметки автоматически выбраны в среде Expression Blend и перенесены в результате соответствующего преобразования в новый класс!

```
  Width="640" Height="480">

  <Grid x:Name="LayoutRoot">
    <local:BubbleThoughtsControl HorizontalAlignment="Left" Height="107"
          Margin="42,39,0,0" VerticalAlignment="Top" Width="233"/>
  </Grid>
</Window>
```

Примечание. Не обращайте особого внимания на знак предупреждения на полупрозрачном желтом фоне, которым помечена копия окна с элементом управления класса `UserControl`. Этим знаком вас предупреждают о необходимости перестроить свой проект в Expression Blend.

Ввод визуальных состояний

В данном случае речь идет о создании совершенно нового элемента управления, а это означает, что стандартные состояния для него по умолчанию не предоставляются. Так, если требуется воспользоваться диспетчером VSM, для этой цели придется определить специальные состояния, их группы и моменты перехода из одно такого состояния в другое.

Примечание. При создании специального элемента управления класса `UserControl` можно было бы, конечно, воспользоваться средой триггеров на платформе WPF, но в данном случае процесс построения такого элемента демонстрируется на примере применения диспетчера VSM.

Убедитесь в том, что активизирован монтажный стол с новым элементом управления класса `UserControl`, т.е. в визуальном конструкторе открыт файл разметки `BubbleThoughtsControl.xaml` или аналогичный файл, если вы назвали новый элемент управления как-то иначе, а затем перейдите к панели States, которая должна быть в настоящий момент пуста. Щелкните на кнопке Add state group (Добавить группу состояний), как показано на рис. 5.57.

Рис. 5.57. Ввод новой группы состояний в специальный элемент управления класса `UserControl`

Присвойте новой группе имя `MyMouseStates`, выбрав его среди стандартных имен или же введя особое имя. Аналогично рассмотренным выше группам `CommonStates` и `FocusedStates` стандартных состояний для группы специальных состояний могут быть назначены устанавливаемые по умолчанию моменты и эффекты перехода. Щелкните на кнопке Add state (Добавить состояние), как показано на рис. 5.58.

Присвойте новому состоянию имя `MouseOverState` (состояние “наведен курсор мыши”). Обратите внимание на автоматический переход в режим регистрации данного состояния. Итак, внесите ряд изменений в свойства рассматриваемого здесь элемента

управления на панели Properties и не забудьте воспользоваться панелью Objects and Timeline, чтобы быстро выбрать отдельные свойства данного элемента. Ради простоты рекомендуется вновь подобрать другой оттенок цвета заполнения геометрической формы данного элемента управления.

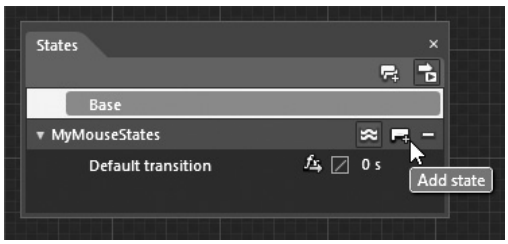


Рис. 5.58. Ввод состояния в группу

Введите второе (и последнее) состояние в группу `MyMouseStates`, присвоив ему имя `MouseDownState` (состояние "кнопка мыши нажата"). После этого внесите ряд изменений в рассматриваемый здесь элемент управления на панели Properties, выполнив, например, преобразование его геометрической формы. В качестве примера на рис. 5.59 приведены окончательно сформированные состояния, исходное время перехода в которые изменено до 1 секунды для всей группы специальных состояний в целом.

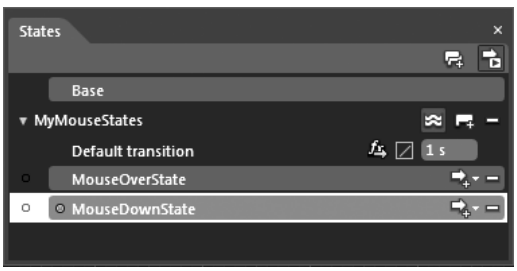


Рис. 5.59. Окончательно сформированная группа специальных состояний

Смена состояний в коде

Если бы вы запустили на выполнение свое приложение на данной стадии его разработки, то с удивлением обнаружили бы, что ни одно из сформированных вами состояний не активизируется, как бы вы ни пытались взаимодействовать мышью с элементом управления. Дело в том, что вы пока еще не указали, когда именно должна происходить смена состояний в этом элементе. И сделать это можно двумя способами. Первый из них заключается в том, чтобы произвести смену состояний программным путем, воспользовавшись методом `GoToState()` из класса `VisualStateManager`. А второй способ будет рассмотрен в следующем разделе.

Убедитесь в том, что элемент управления класса `UserControl` выбран на монтажном столе, перейдите к панели Properties и щелкните на кнопке Events (со знаком молнии на пиктограмме). Организуйте обработку события `MouseDown` как обычно, а затем введите приведенный ниже код в автоматически сформированный обработчик событий типа `MouseDown`.

```
private void callout_MouseDown(object sender,
  System.Windows.Input.MouseButtonEventArgs e)
{
```

```
    // Перейти в новое состояние!
    VisualStateManager.GoToState(this, "MouseDownState", true);
}
```

Первый аргумент метода `GoToState()` обозначает элемент управления, имеющий группы состояний. Как правило, это сам элемент управления класса `UserControl`. Второй аргумент данного метода обозначает имя состояния, в которое требуется перейти, а третий аргумент имеет логическое значение, указывающее, требуется ли использовать любые заранее определенные временные характеристики перехода в состояние. Еще раз запустите свое приложение на выполнение и щелкните на элементе управления. Его состояние должно измениться!

Смена состояний в разметке

Сменять состояния можно и без написания процедурного кода, используя объект поведения типа `GoToStateAction`. Такой способ может оказаться очень удобным при построении специальных шаблонов, которые предполагается применять во многих проектах и в то же время не загромождать ими элементы управления класса `UserControl`. Итак, найдите упомянутый выше объект поведения в библиотеке ресурсов, как показано на рис. 5.60.

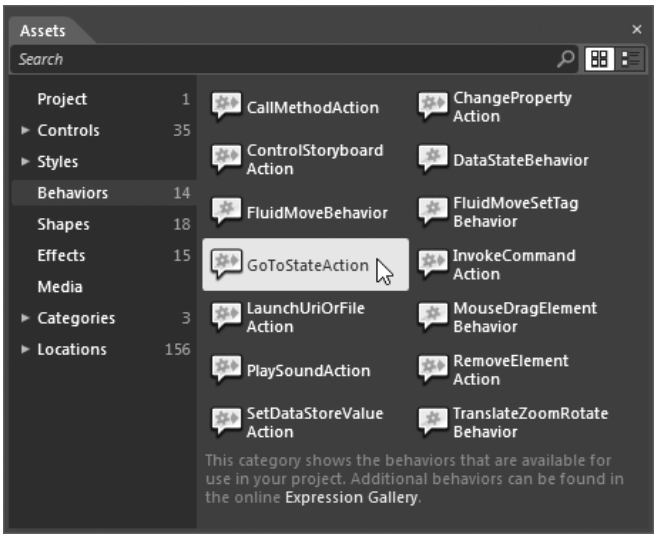


Рис. 5.60. С помощью объекта поведения `GoToStateAction` можно организовать смену состояний непосредственно в разметке

Перетащите объект поведения типа `GoToStateAction` на элемент управления класса `UserControl`. (Вы можете опустить его на этот элемент непосредственно на монтажном столе или же в узле иерархического представления на панели Objects and Timeline.) Как только вы сделаете это, появится ряд объектов, приведенных на рис. 5.61.

Выберите этот объект и перейдите к панели Properties, чтобы настроить соответствующее поведение. Так, в области Trigger можно выбрать контролируемое событие (в данном случае это событие `MouseEnter` (Курсор мыши вошел в пределы элемента управления)). А в области Common Properties можно выбрать специальное состояние, в которое требуется перейти (в данном случае это состояние `MouseOverState`). На рис. 5.62 приведен результат окончательной настройки объекта поведения типа `GoToStateAction`.

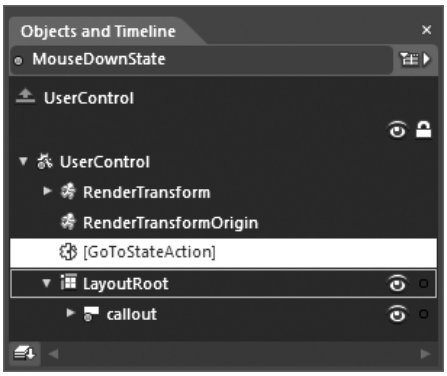


Рис. 5.61. Готовый к настройке объект поведения типа `GoToStateAction`

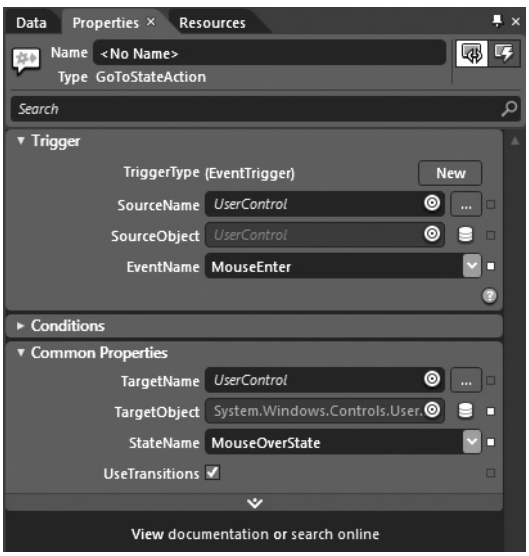


Рис. 5.62. Объект поведения типа `GoToStateAction`, настроенный на смену состояний

Вновь запустите свое приложение на выполнение и проверьте, происходит ли переход во второе состояние, когда курсор оказывается над элементом управления.

Дополнительные ресурсы по изучению диспетчера VSM

На этом рассмотрение способов и средств создания стилей и шаблонов в среде Expression Blend IDE завершается. Имеется ряд других связанных с ними вопросов, требующих дополнительного изучения, но и без этого вы должны теперь почувствовать себя более уверенно, выполняя основные операции со стилями и шаблонами в Expression Blend. Если же хотите узнать больше о диспетчере VSM, обратитесь за справкой к разделу “Определение различных визуальных состояний и временных характеристик перехода в них для элементов управления пользовательского интерфейса” (Define different visual states and transition times for a user control) справочного руководства Expression Blend (рис. 5.63).

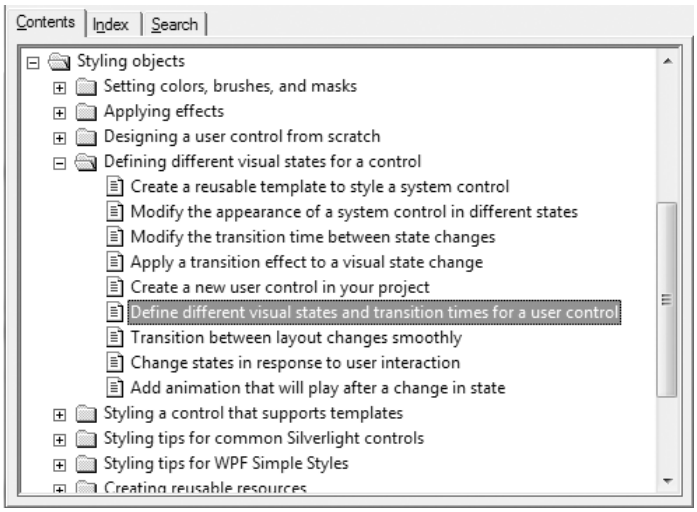


Рис. 5.63. Диспетчер VSM и процесс создания элементов управления класса `UserControl` более подробно рассматриваются в документации по Expression Blend

Резюме

В этой главе был рассмотрен целый ряд вопросов, но все они были посвящены процессу специальной настройки внешнего вида и поведения элементов управления на платформах WPF и Silverlight. Сначала в главе было пояснено назначение стилей. В частности, стили обеспечивают одинаковый внешний вид (т.е. одни и те же установки свойств) связанных вместе элементов пользовательского интерфейса, указываемых в атрибуте разметки `TargetType`. Затем был представлен целый ряд инструментальных средств, предоставляемых в среде Expression Blend IDE для упрощения процесса создания и применения стилей.

Далее в главе пояснялось назначение шаблонов, с помощью которых можно полностью изменить стиль оформления элементов управления, а следовательно, их внешний вид. Попутно было показано, каким образом в Expression Blend извлекается копия шаблона, применяемого к элементу управления по умолчанию. Но самом интересное, что в этой части главы был рассмотрен довольно простой способ построения шаблонов из существующей графики.

В создаваемые шаблоны обычно внедряются и так называемые визуальные подсказки, упрощающие взаимодействие пользователя с интерфейсом приложения. Как было показано в этой главе, в проектах на платформе WPF такое взаимодействие организуется с помощью оригинальной среды триггеров, тогда как на платформе Silverlight — с помощью диспетчера визуальных состояний (VSM), который, впрочем, был реализован и на платформе WPF после выпуска версии .NET 4.0.

И в завершение главы было пояснено назначение классов `UserControl`, с помощью которых создаются специальные элементы управления. Попутно было показано, каким образом новый элемент управления класса `UserControl` извлекается из существующей совокупности элементов пользовательского интерфейса, а затем определяются специальные состояния и переходы между ними с помощью диспетчера VSM.

ГЛАВА 6

Способы привязки данных в Expression Blend

В среде Expression Blend IDE имеется целый ряд инструментальных средств для связывания значений данных, получаемых из разных источников, с компонентами пользовательского интерфейса. Вообще говоря, такой процесс называется *привязкой данных* и поддерживается на обеих платформах, WPF и Silverlight, хотя и на разных уровнях прикладных интерфейсов API[1]. В этой главе будет рассмотрен целый ряд способов и средств привязки данных в Expression Blend, а также некоторые связанные с данным процессом приемы программирования.

В начале главы будет рассмотрен принцип выполнения операций *привязки данных одних элементов управления к другим*. Как подразумевает само название этого подхода к привязке данных, он позволяет связывать значение свойства *исходного* элемента управления со значением свойства *целевого* элемента управления в приложении. По ходу рассмотрения этого принципа будет разъяснено отличие между операциями односторонней и двухсторонней привязки данных, а также порядок преобразования значений данных при их передаче от источника к адресату с помощью специально создаваемых в коде *классов преобразования данных*.

Далее будет показано, каким образом свойства объектов, не относящихся к пользовательскому интерфейсу, например бизнес-объектов приложения или XML-документов, привязываются к свойствам объектов графического пользовательского интерфейса. Попутно будет представлена панель Data и целый ряд способов визуального связывания значений свойств класса с элементами пользовательского интерфейса.

В этой главе рассматриваются также *шаблоны данных*. Подобно шаблону элемента управления, представленному в главе 5, шаблон данных позволяет определить специальный набор инструкций визуализации, выполняемых во время операции привязки данных, вложенные диспетчеры компоновки, графические объекты и виды анимации.

И в заключение главы будет показано, каким образом в Expression Blend создаются новые специальные хранилища данных, а также разъяснено назначение выборочных данных, которые можно вводить в приложение на платформе WPF или Silverlight. Выборочные данные не только способствуют дальнейшему исследованию различных способов и средств привязки данных в Expression Blend, но и служат в качестве полезных заполняющих данных при создании прототипов[2].

---

[1] Как будет показано в главе 7, привязка данных поддерживается и в модели программирования на платформе Windows Phone 7.

[2] См. главу 8.

Примечание. Несмотря на то что в Expression Blend предоставляется довольно развитая инфраструктура для привязки данных, в целом ряде случаев требуется написание довольного большого объема процедурного кода, что выходит за рамки данной книги. Тем не менее в конце этой главы будут указаны ссылки на полезные ресурсы для дальнейшего изучения вопросов реализации привязки данных в коде.

Назначение привязки данных

Элементы графического пользовательского интерфейса нередко оказываются целью различных операций привязки данных. Проще говоря, *привязка данных* — это действие связывания свойства элемента управления со значениями данных. Совершая это действие, можно значительно упростить задачу написания кода, поскольку текущее состояние привязываемых свойств будет автоматически отображаться в пользовательском интерфейсе. Ниже приведены характерные примеры привязки данных.

- Привязка значения свойства логического класса к элементу управления типа `CheckBox`.
- Заполнение объекта типа `DataGrid` данными из специальной коллекции.
- Заполнение элемента управления типа `ListBox` значениями из XML-документа.

Для правильной реализации привязки данных необходимо ясно представлять себе отличие *источника* операции привязки от ее *адресата*, или *цели*. Как и следовало ожидать, в качестве источника операции привязки данных служит место, откуда поступают данные (свойство объекта, узел разметки XML и т.д.), а в качестве адресата (или цели) — свойство элемента пользовательского интерфейса (типа `CheckBox`, `TextBox` и т.д.), получающего из источника данные и использующего их. Для прояснения этого важного отличия приведенные выше примеры привязки данных можно уточнить следующим образом.

- Исходное логическое значение может быть привязано к целевому свойству элемента управления типа `CheckBox`.
- Целевой объект типа `DataGrid` может быть заполнен данными из исходной специальной коллекции.
- Целевой объект типа `ListBox` может быть заполнен данными из исходного XML-документа, опираясь на оператор `XPath`.

Откровенно говоря, пользоваться имеющейся инфраструктурой привязки данных совсем не обязательно. Если бы разработчик создал собственную логику привязки данных, связывание свойств двух объектов обычно включало бы в себя обработку различных событий и написание процедурного кода для установления связи между источником и адресатом. Так, если в главном окне типа `Window` имеется элемент управления типа `ScrollBar`, значение свойства которого требуется отображать в свойстве элемента управления типа `Label`, для этой цели может потребоваться обработка события `ValueChanged`, наступающего при изменении значения в свойстве элемента управления типа `ScrollBar`, а следовательно, и обновление соответствующего свойства элемента управления типа `Label`.

Как правило, привязка данных на платформах WPF и Silverlight позволяет устанавливать подобные связи исключительно средствами XAML. И хотя это справедливо в большинстве случаев, для реализации привязки данных иногда все же приходится писать дополнительный процедурный код, как будет показано далее в главе.

Привязка данных одних элементов управления к другим

Рассмотрение способов и средств, применяемых в Expression Blend для определения операций привязки данных, будет начато с примера, демонстрирующего способ привязки значения свойства одного элемента управления к значению свойства другого элемента управления (так называемой *привязки данных одних элементов управления к другим*). Этот способ привязки данных может оказаться очень полезным в тех случаях, когда требуется обновлять значение свойства элемента управления, исходя из выбора, сделанного пользователем.

Создание примера пользовательского интерфейса

Запустите на выполнение Expression Blend и создайте новый проект приложения WPF, присвоив ему имя `ControlToControlBinding`[3]. Для целей данного примера допустим, что в главном окне приложения, представленном объектом типа `Window`, находятся два объекта типа `StackPanel`, в свойстве `Orientation` обоих этих объектов установлено значение **`Horizontal`** и каждый из них содержит элементы управления типа `Slider` и `Label`.

Оба объекта типа `StackPanel` объединены в более крупный элемент управления типа `StackPanel`, образующий блок укладываемых одна в другую блочных панелей. Для построения пользовательского интерфейса, приведенного на рис. 6.1, воспользуйтесь панелью Assets и монтажным столом, введя соответствующие текстовые надписи в свойстве `Content` каждого элемента управления типа `Label`. А о настройке элементов управления типа `Slider` речь пойдет несколько позже.

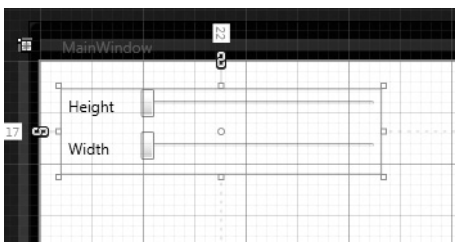


Рис. 6.1. Первоначальный вид пользовательского интерфейса

Примечание. Как пояснялось в главе 4, вы можете выделить ряд элементов управления на монтажном столе и щелкнуть на них правой кнопкой мыши, чтобы сгруппировать их в новый диспетчер компоновки по команде меню Group Into. Не забывайте также о возможности выполнять элементарные операции копирования и вставки на монтажном столе или на панели Objects and Timeline.

Ниже приведена разметка, автоматически формируемая в коде XAML при создании в среде Expression Blend IDE подобного пользовательского интерфейса. У вас она может немного отличаться в зависимости от характера настройки отдельных объектов, составляющих пользовательский интерфейс вашего приложения.

```
<StackPanel HorizontalAlignment="Left" Margin="28,40,0,0"
          Orientation="Vertical"
          VerticalAlignment="Top" Width="248">
```

[3] Способ, демонстрируемый в данном примере, более или менее подходит и для разработки проектов на платформе Silverlight, поэтому вы вольны создать проект приложения Silverlight, если это в большей степени отвечает вашим интересам.

```
  <StackPanel Height="33" Orientation="Horizontal"
           d:LayoutOverrides="Width">
    <Label Content="Height" Width="61"/>
    <Slider Width="187"/>
  </StackPanel>
  <StackPanel Height="33" Orientation="Horizontal"
           d:LayoutOverrides="Width">
    <Label Content="Width" Width="61"/>
    <Slider Width="187"/>
  </StackPanel>
</StackPanel>
```

Выберите элементы управления типа `Slider` на монтажном столе, нажав клавишу <Ctrl> и щелкнув на каждом из них по очереди, а затем перейдите к панели Properties и установите значение **200** в свойстве `Maximum`, доступном в области Common Properties этой панели (это и другие свойства вы сможете быстро найти, воспользовавшись полем поиска Search). В итоге каждый элемент управления типа `Slider` будет определен в разметке следующим образом:

```
<Slider Width="187" Maximum="200" Minimum="10"/>
```

Эти элементы управления типа `Slider` послужат для изменения размеров других элементов пользовательского интерфейса на монтажном столе с помощью операции привязки данных. И завершите создание пользовательского интерфейса, нарисовав на монтажном столе произвольный графический элемент как часть объекта `LayoutRoot` типа `Grid`. В качестве примера на рис. 6.2 показана геометрическая форма треугольника, выбранная из категории Shapes в библиотеке ресурсов.

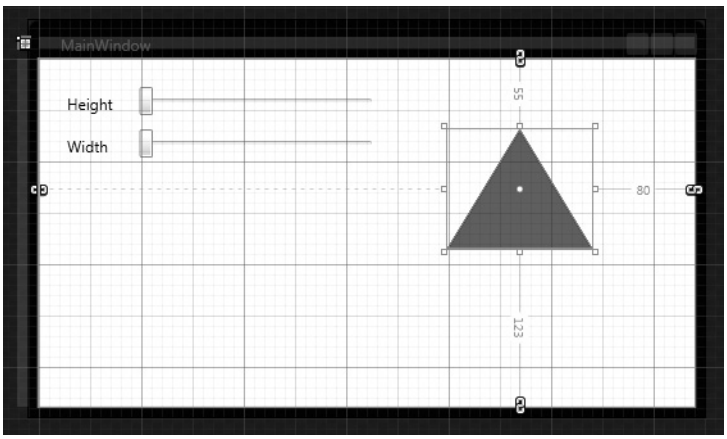


Рис. 6.2. Окончательный вид пользовательского интерфейса

Формирование новых привязок данных

Когда в среде Expression Blend требуется привязать значения свойства одного элемента управления к свойству другого элемента управления, то для этой цели необходимо сначала выбрать целевой элемент управления, т.е. *адресат* операции привязки данных (в данном случае это геометрическая форма треугольника). Итак, выберите этот элемент пользовательского интерфейса на монтажном столе и найдите его свойство `Height` в области Layout на панели Properties. После этого щелкните на кнопке Advanced options,

обозначенной пиктограммой мелкого квадратика справа от текстового поля свойства `Height`, как показано на рис. 6.3.

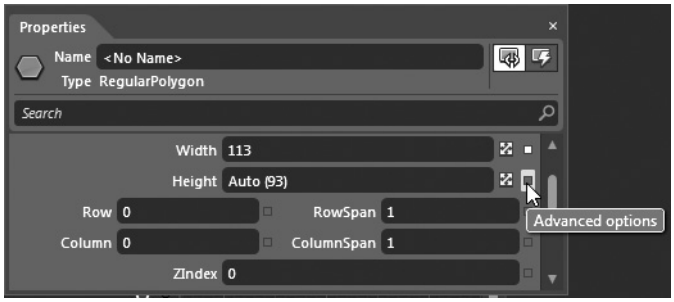


Рис. 6.3. Активизация дополнительных параметров настройки свойства `Height`

Выберите пункт Data Binding (Привязка данных) из всплывающего меню (рис. 6.4)[4].

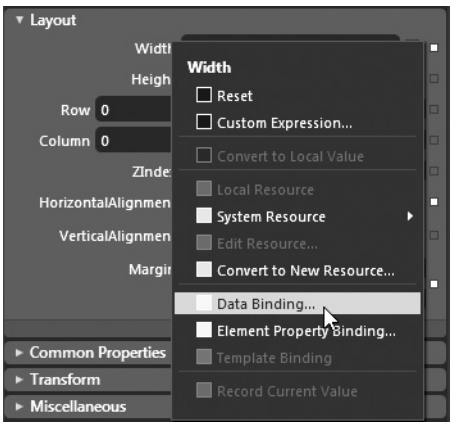


Рис. 6.4. Выбор пункта меню Data Binding

В верхней части открывшегося диалогового окна Create Data Binding (Формирование привязки данных) находятся три вкладки, на которых можно настраивать различные типы источников данных (рис. 6.5). Для этой цели, по существу, имеются следующие возможности.

- **Вкладка** Data Field **(Поле данных).** На этой вкладке в качестве источника данных можно выбрать объект .NET или XML-документ[5].
- **Вкладка** Element Property **(Свойство элемента).** На этой вкладке в качестве источника данных можно выбрать свойство элемента пользовательского интерфейса.
- **Вкладка** Data Context **(Контекст данных).** На этой вкладке можно выбрать *контекст данных*, уже определенный в приложении в качестве источника данных. Как поясняется далее в главе, контекст данных позволяет определить в

[4] Операции привязки данных можно также настроить, выбрав из меню пункт Element Property Binding (Привязка свойства элемента), как будет показано далее в главе.

[5] Как поясняется более подробно далее в этой главе, операции привязки данных слабо поддерживаются в прикладном интерфейсе Silverlight API. Поэтому кнопка +XML недоступна на вкладке Data Field при разработке проектов на платформе Silverlight.

родительском контейнере, например в диспетчере компоновки, источник данных, которым могут свободно пользоваться все порожденные элементы.

Каждый из перечисленных выше способов привязки данных будет рассмотрен более подробно на протяжении всей этой главы, а до тех пор выберите среднюю вкладку Element Property. Как пояснялось выше, на этой вкладке можно устанавливать связь между значением свойства исходного объекта и значением свойства целевого объекта, выбранного на монтажном столе. Итак, найдите и выберите первый объект типа `Slider` в расположенном *слева* иерархическом представлении, а затем выберите свойство `Value` в расположенном *справа* иерархическом представлении (см. рис. 6.5).

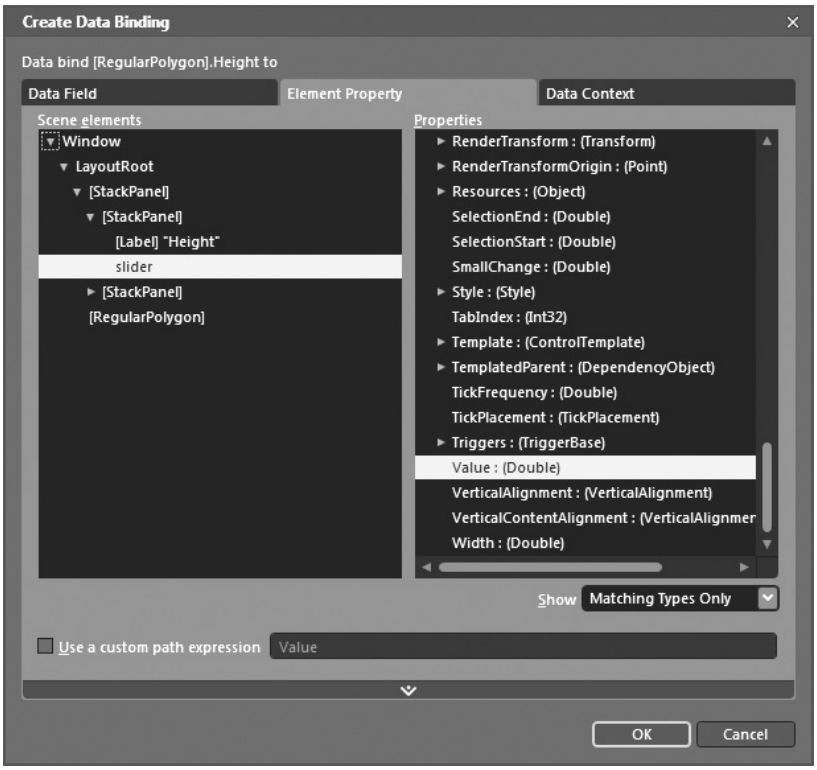


Рис. 6.5. Связывание свойства `Height` геометрической формы со свойством `Value` элемента управления типа `Slider`

Примечание. В данном примере элементам пользовательского интерфейса не присвоены подходящие имена, поэтому используются имена, заданные по умолчанию, например `slider`, `slider1` и т.д.

Щелкните на кнопке OK и повторите описанную выше процедуру, чтобы привязать свойство `Width` геометрической формы (в данном случае треугольника) к свойству `Value` второго элемента управления типа `Slider`. Если после этого вы запустите свое приложение на выполнение, то обнаружите, что размеры геометрической формы можно изменять перемещением ползунков. Не так уж и плохо всего лишь для нескольких действий мышью! Попробуйте также изменить значения свойств `Minimum` и `Maximum` каждого элемента управления типа `Slider` и понаблюдайте за их новым поведением.

Вернитесь к Expression Blend, выберите геометрическую форму на монтажном столе и перейдите к панели Properties. Как видите, свойства `Height` и `Width` этой формы обведены желтой ограничивающей рамкой (рис. 6.6, а также цветная вклейка). Эта ограничивающая рамка служит в Expression Blend визуальной подсказкой для установки указанных свойств с помощью операции привязки данных.

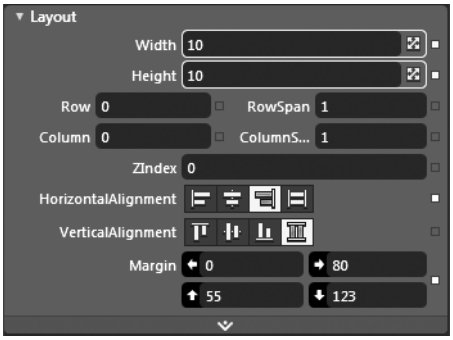


Рис. 6.6. Привязанные свойства обводятся желтой ограничивающей рамкой

Просмотр сформированной разметки

Если вы откроете редактор XAML, то обнаружите приведенную ниже разметку, автоматически сформированную в результате привязки данных. Обратите внимание на то, что в описании целевого объекта (в данном случае геометрической формы треугольника) используется расширение разметки XAML под названием `{Binding}`. У такого расширения разметки имеются два главных компонента. Во-первых, это имя исходного свойства (в данном случае свойства `Value` обоих элементов управления типа `Slider`). И во-вторых, это атрибут `ElementName`, обозначающий имя исходного объекта (в данном случае обоих элементов управления типа `Slider`).

```
<ed:RegularPolygon Fill="#FFAD0BAF" HorizontalAlignment="Right"
   InnerRadius="1" Margin="0,55,80,123" PointCount="3"
   Stretch="Fill" Stroke="Black"
   Width="{Binding Value, ElementName=slider1}"
   Height="{Binding Value, ElementName=slider}"/>
```

По существу, в этой разметке утверждается следующее: “значение свойства `Width` будет присвоено свойству `Value` объекта типа `slider1`, тогда как значение свойства `Height` — свойству `Value` объекта типа `slider`”.

Преобразование типов данных

В рассматриваемом здесь примере проекта привязка данных действует безошибочно, поскольку исходный тип данных свойства `Value` обоих элементов управления типа `Slider` совместим с исходным типом данных свойств `Height` и `Width` геометрической формы. Все эти свойства оперируют числовыми значениями (в данном случае типа `double`, т.е. с плавающей точкой двойной точности).

Но что, если требуется связать не совсем совместимые по типу данных свойства и преобразовать их значения, например, изменять цвет геометрической формы в зависимости от положения ползунка в третьем элементе управления типа `Slider`? По зрелом размышлении над этим вопросом становится очевидным следующее затруднение: значение свойства `Value` элемента управления типа `Slider` относится к числовому типу

данных `double`, тогда как в свойстве `Fill` геометрической формы требуется указать объект кисти! Следовательно, если требуется преобразовать значение свойства в новый тип данных, для этой цели можно написать в коде специальный класс, выполняющий подобное преобразование, а затем зарегистрировать его в разметке.

Итак, добавьте еще один элемент управления типа `Slider` на монтажном столе (его местоположение особого значения не имеет). Затем установите в свойстве `Content` элемента управления `Label` значение **`Shade`** (Оттенок) и настройте элемент управления типа `Slider` таким образом, чтобы его ползунок перемещался в пределах от максимального значения `255` до минимального значения **0**. Кроме того, убедитесь в том, что в свойстве `SmallChange` (Незначительное изменение) элемента управления типа `Slider` установлено значение **1**. По завершении разметка новых элементов управления должна выглядеть так, как показано ниже. (Следует иметь в виду, что и в этом случае новые элементы управления расположены по горизонтали на блочной панели типа `StackPanel`.)

```
<StackPanel Height="33" Orientation="Horizontal"
          d:LayoutOverrides="Width">
  <Label Content="Shade" Width="61"/>
  <Slider x:Name="slider2" Width="187"
        Maximum="255" SmallChange="1"/>
</StackPanel>
```

Создание специального класса преобразования данных

Прежде чем формировать новую привязку данных, нужно написать специальный класс, к которому придется обращаться во время операции привязки данных. С этой целью введите новый класс C# в свой проект по команде Project⇨Add New Item (Проект⇨Добавить новый элемент), выбираемой из главного меню в среде Expression Blend IDE. Добавьте новый класс `DoubleToSolidBrushConverter.cs` в открывшемся диалоговом окне New Item, как показано на рис. 6.7.

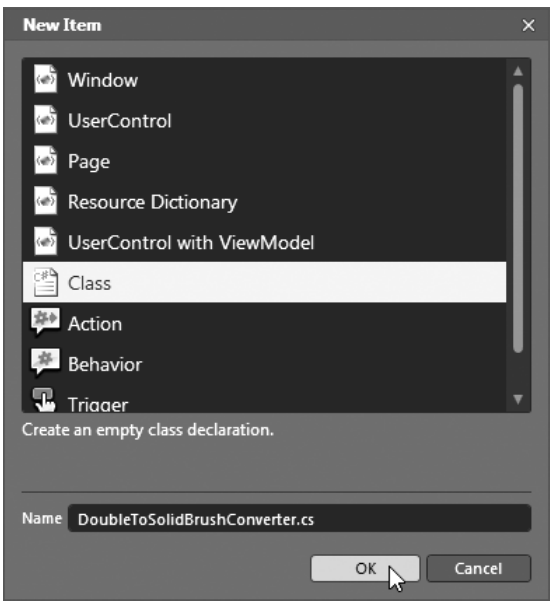


Рис. 6.7. Ввод нового класса C# в проект, разрабатываемый в среде Expression Blend IDE

Примечание. Здесь и далее предполагается, что вы умеете программировать на C#. В противном случае обращайтесь за справкой к исходному коду готового решения данного проекта из загружаемого архива примеров проектов к этой книге, читая этот раздел.

В специальном классе преобразования данных должен быть реализован конкретный интерфейс .NET под названием `IValueConverter`. В этом интерфейсе определяются два метода: `Convert()` и `ConvertBack()`. В частности, метод `Convert()` вызывается при выполнении операций односторонней привязки данных, тогда как метод `ConvertBack()` — при выполнении операций двухсторонней привязки данных (оба режима привязки данных более подробно рассматриваются далее в главе). Оба эти метода реализуются в классе преобразования данных, но они могут возвращать пустое значение `null`, если преобразование данных не требуется. В данном примере основной интерес вызывает операция односторонней привязки данных, поэтому введите в новый класс приведенный ниже код.

```
public class DoubleToSolidBrushConverter : IValueConverter
{
  public object Convert(object value, Type targetType, object parameter,
                    System.Globalization.CultureInfo culture)
  {
    double currValue = (double)value;
    byte valueAsByte = (byte)currValue;
    Color color = new Color();
    color.A = 255;
    color.G = valueAsByte;
    return new SolidColorBrush(color);
  }

  public object ConvertBack(object value, Type targetType,
                        object parameter,
                        System.Globalization.CultureInfo culture)
  { return null; }
}
```

Напомним, что метод `Convert()` служит для преобразования исходного значения типа `double`, передаваемого от элемента управления типа `Slider` к объекту специальной кисти. Как видите, это исходное значение типа `double` передается данному методу в качестве аргумента `value` типа `object`, а затем приводится к типу данных `byte`, который ожидает получить новый объект типа `Color`. Именно это новое значение и послужит для настройки объекта типа `SolidColorBrush` на заданный оттенок зеленого в зависимости от настройки свойства `G` объекта типа `SolidColorBrush`.

Примечание. Прежде чем продолжить работу над данным примером проекта, перестройте исходный код по команде меню Project⇨Build Project (Проект⇨Построить проект). Если вы не сделаете этого, то не сможете найти специальный класс преобразования в рассматриваемой далее разметке.

Выбор класса преобразования данных в Expression Blend

А теперь, когда класс преобразования данных внедрен на свое место в текущем проекте, к нему можно обращаться при определении операции привязки данных в разметке XAML. С этой целью откройте файл разметки главного окна приложения `MainWindow.xaml` в визуальном конструкторе Expression Blend и вновь выберите объект геометрической формы (в данном случае треугольника) на монтажном столе, но на этот раз

щелкните на кнопке Advanced options, обозначенной пиктограммой мелкого квадратика справа от свойства `Fill`, доступного в области Brushes на панели Properties. После этого выберите из всплывающего меню пункт Data Binding таким же образом, как делали это ранее в данном примере при настройке свойств `Height` и `Width` той же самой геометрической формы. И наконец, выберите вкладку Element Property в открывшемся диалоговом окне Create Data Binding.

Выберите третий элемент управления типа `Slider` в иерархическом представлении *слева* в диалоговом окне Create Data Binding, а затем попытайтесь найти свойство `Value` *справа* в том же окне. Как ни странно, оно там отсутствует! Дело в том, что по умолчанию на вкладке Element Property в диалоговом окне Create Data Binding отображаются только *совместимые* типы данных. Поэтому раскройте список Show (Показ) в нижней части данного окна и выберите из него режим All Properties (Все свойства). И только после этого вы сможете выбрать свойство `Value`, как показано на рис. 6.8.

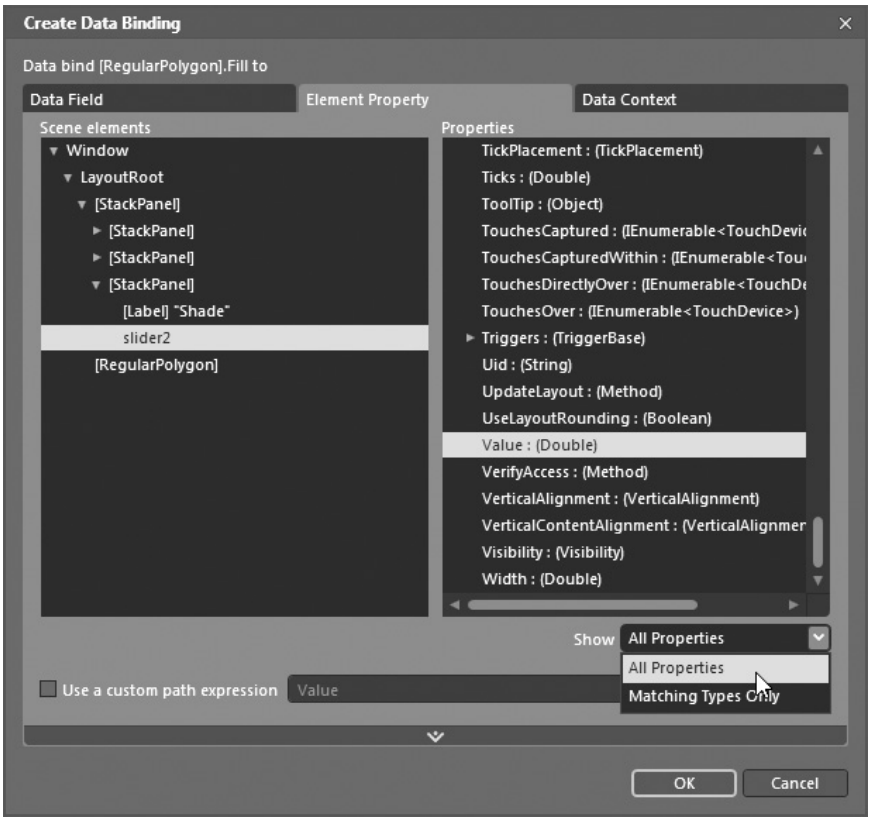


Рис. 6.8. Для просмотра несовместимых свойств следует выбрать режим All Properties

В том же самом диалоговом окне можно указать и специальный класс преобразования данных. С этой целью разверните область дополнительных параметров настройки в нижней части диалогового окна Create Data Binding и щелкните на кнопке Add new value converter (Добавить новый преобразователь данных) как показано на рис. 6.9, где приведена лишь нужная в данном случае (т.е. нижняя) часть этого диалогового окна.

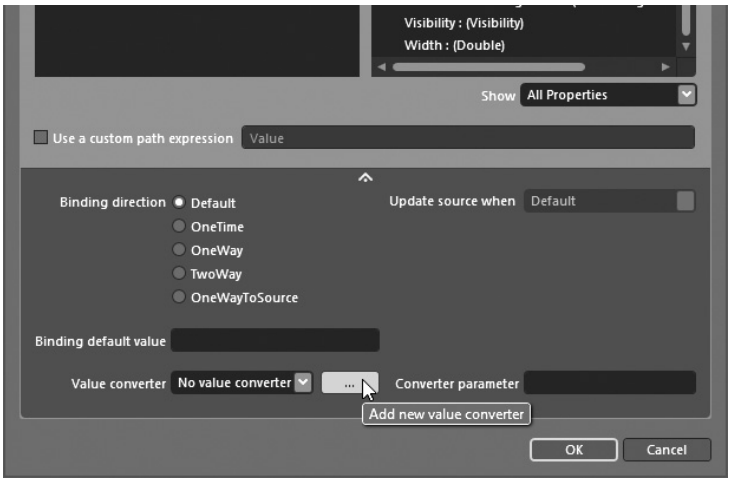


Рис. 6.9. В области дополнительных параметров настройки диалогового окна Create Data Binding можно указать специальный класс преобразования данных

Щелкнув на этой кнопке, вы должны обнаружить специальный класс преобразования данных `DoubleToSolidBrushConverter` среди прочих классов в открывшемся диалоговом окне (рис. 6.10). Если вам не удастся обнаружить специальный класс преобразования данных в этом окне, это, скорее всего, означает, что вы забыли перестроить свой проект, как рекомендовалось сделать в предыдущем примечании.

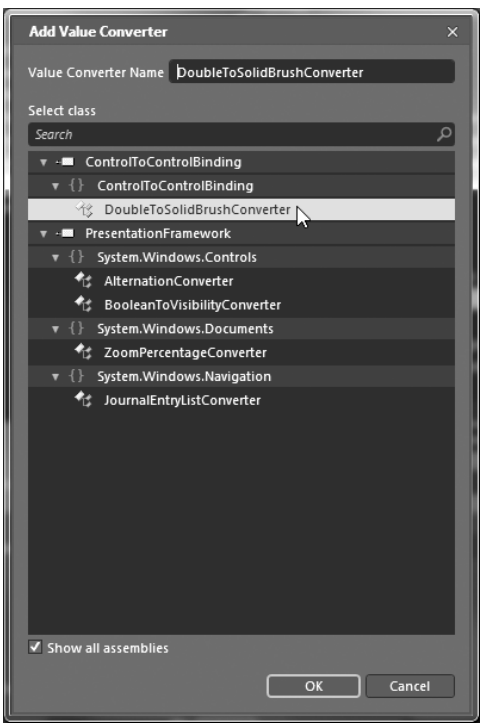


Рис. 6.10. Выбор специального класса преобразования данных

Запустите свое приложение на выполнение. Цвет заполнения геометрической формы должен изменяться при перемещении ползунка `Shade`, как показано на рис. 6.11, а также на цветной вклейке.

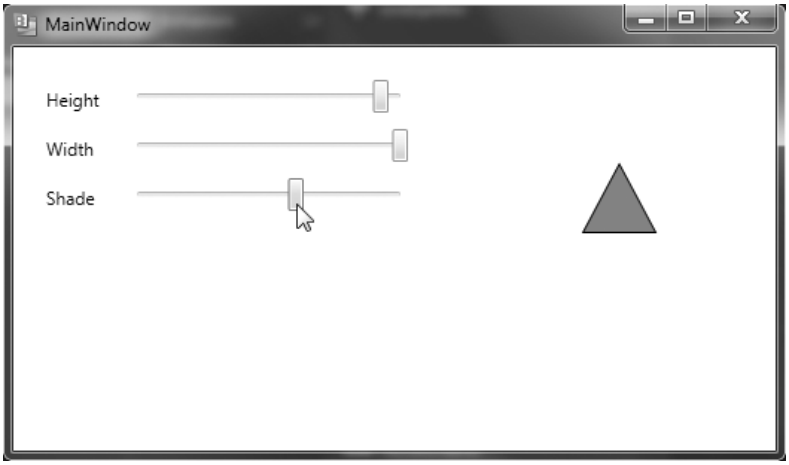


Рис. 6.11. Результат преобразования устанавливаемого ползунком числового значения типа `double` в раскрашиваемый кистью оттенок сплошного цвета заполнения геометрической формы

Просмотр сформированной разметки

Если вы проанализируете сформированную разметку, то обнаружите в ней специальный класс преобразования данных, объявленный в качестве ресурса объекта, применяемого на уровне главного окна приложения, представленного объектом типа `Window`. Ниже приведен соответствующий фрагмент этой разметки.

```
<Window.Resources>
  <local:DoubleToSolidBrushConverter
        x:Key="DoubleToSolidBrushConverter"/>
</Window.Resources>
```

Префикс `local:` в приведенном выше фрагменте разметки был сформирован в Expression Blend автоматически, когда вы выбрали специальный класс преобразования данных. Как известно, для описания специального класса в XAML необходимо определить пространство имен XML, увязываемое с пространством имен .NET, где и определяется специальный класс. Как показано в приведенном ниже фрагменте разметки, в открывающем дескрипторе `<Window>` префикс `local:` увязывает пространство имен XML с корневым пространством имен .NET для текущего проекта.

```
<Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  ...
  xmlns:local="clr-namespace:ControlToControlBinding"
  ...
>
```

Так или иначе, в атрибуте `Converter` расширения разметки `{Binding}` геометрической формы делается ссылка на этот ресурс объекта. (Для большей удобочитаемости приведенный ниже фрагмент разметки был разбит на несколько строк, хотя это и не следует делать в самом проекте.)

```
<ed:RegularPolygon Fill="{Binding Value,
  Converter={StaticResource DoubleToSolidBrushConverter},
```

```
  ElementName=slider2}"
  ...
/>
```

А теперь перейдем к рассмотрению назначения режимов привязки данных.

Представление о режимах привязки данных

В операциях привязки данных поддерживается атрибут `Mode`, с помощью которого определяется порядок синхронизации источника и адреса этих операций во время работы приложения. Подобным способом можно, в частности, обеспечить своевременное обновление адресата при изменении источника или, наоборот, автоматическое обновление источника при изменении адресата. Для этих целей в атрибуте `Mode` может быть установлено одно из следующих значений.

- **`OneWay`**. Адресат обновляется, когда изменяется источник.
- **`TwoWay`**. Адресат обновляется, когда изменяется источник, а *если* изменяется адресат, то обновляется источник.
- **`OneWayToSource`**. Это значение противоположно по своему смыслу значению `OneWay`: источник обновляется, когда изменяется адресат[6].
- **`OneTime`**. Адресат обновляется значением из источника при запуске приложения на выполнение, а затем вся логика привязки данных предается забвению.

Установка режимов привязки данных в Expression Blend

Для того чтобы ознакомиться поближе с наиболее распространенными режимами привязки данных, добавьте еще несколько элементов управления на монтажном столе. В частности, введите элемент управления типа `Label`, указав в его свойстве `Content` текстовую надпись **`Change Number`** (Изменить число), еще один элемент управления типа `Slider`, а также элемент управления `myTextBox` типа `TextBox`, удовлетворившись исходными настройками свойств двух последних.

А теперь привяжите свойство `Value` четвертого элемента управления типа `Slider` к свойству `Text` элемента управления типа `TextBox`, но на этот раз поступите иначе, чем прежде. Сначала выберите элемент управления типа `TextBox` на монтажном столе и найдите свойство `Text` на панели Properties. Затем щелкните на кнопке Advanced options и выберите пункт Element Property Binding (Привязка свойства элемента) из всплывающего меню, как показано на рис. 6.12.

Как только вы выберете этот пункт меню, то сразу же заметите, что курсор принял вид знака, похожего на бычий глаз (рис. 6.13). Этот знак указывает на то, что вы можете теперь щелкнуть на элементе управления, представляющем на монтажном столе *источник* для операции привязки данных. Итак, щелкните на вновь созданном, четвертом элементе управления типа `Slider`.

После этого можете выбрать свойство `Value` из раскрывающегося списка в открывшемся диалоговом окне. А если вы развернете область дополнительных параметров настройки в нижней части этого окна, щелкнув на пиктограмме стрелки, направленной вниз, то обнаружите целый ряд доступных для выбора режимов привязки данных. Выберите пока что режим `OneWay` (Односторонняя привязка данных), как показано на рис. 6.14.

[6] Значение `OneWayToSource` режима привязки данных не поддерживается на платформе Silverlight.

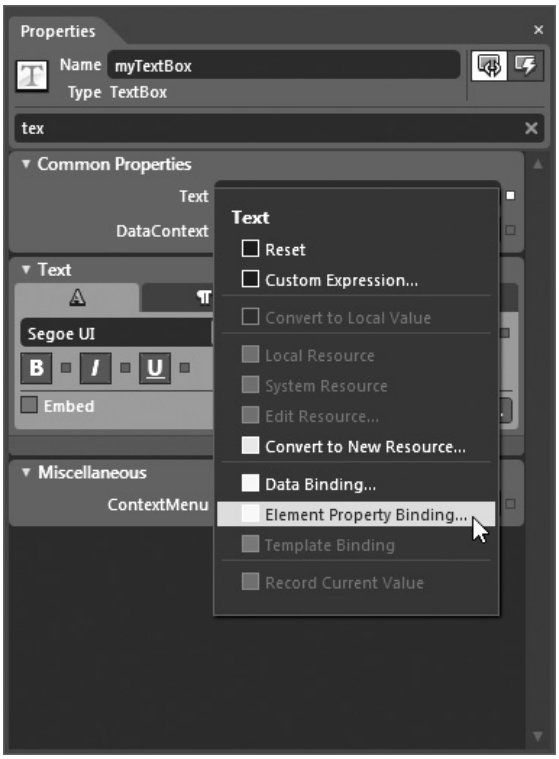


Рис. 6.12. Выбор пункта меню Element Property Binding — альтернативный способ определения привязки данных

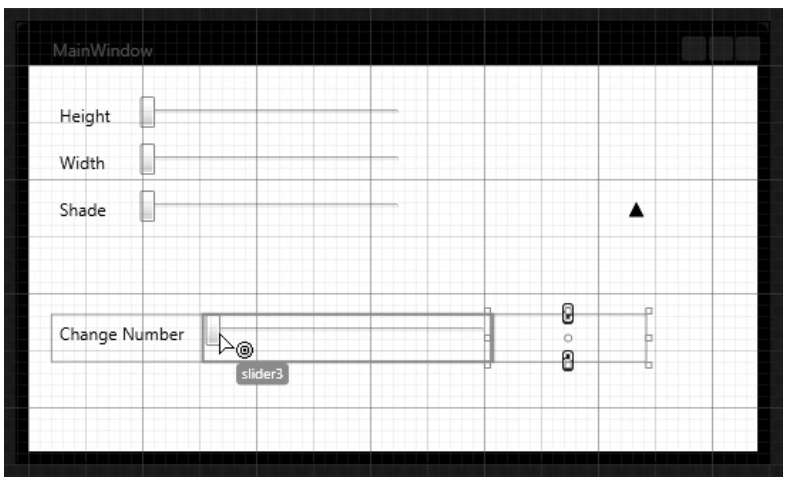


Рис. 6.13. Выбор элемента управления типа `Slider` в качестве источника для операции привязки данных (в данном случае привязки свойства элемента)

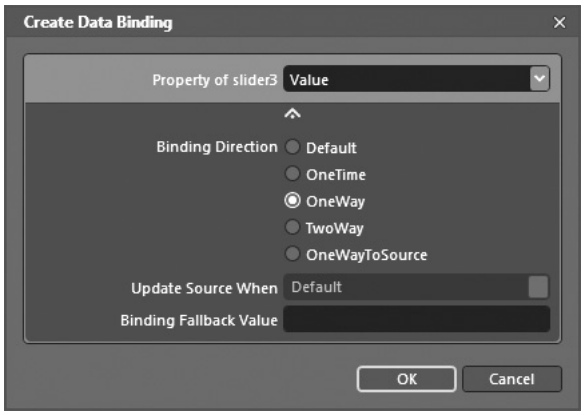


Рис. 6.14. Установка режима односторонней привязки данных

Установка режима двухсторонней привязки данных

Если вы теперь запустите свое приложение на выполнение, то сможете убедиться, что текстовое поле обновляется при перемещении ползунка **Change Number**. Но если вы попытаетесь ввести в этом текстовом поле значение, находящееся в пределах области действия четвертого элемента управления типа `Slider`, например число **100**, то его ползунок не изменит своего положения, поскольку в режиме односторонней привязки данных обновляется только адресат, когда изменяется источник данной операции, но не наоборот. Если же вы перенастроите эту операцию на режим двухсторонней привязки данных, то сможете изменить положение ползунка, введя новые данные в текстовом поле.

Для коррекции любой привязки данных найдите свойство, настроенное на нее в настоящий момент (в данном случае — свойство `Text`), щелкните на кнопке Advanced options и выберите, как обычно, пункт Data Binding из всплывающего меню. Потом можете выбрать режим двухсторонней привязки данных (`TwoWay`) и определить порядок обновления исходного свойства. В качестве примера на рис. 6.15 показан результат обновления элемента управления типа `Slider`, когда элемент управления типа `TextBox` теряет логический фокус.

Теперь можете убедиться сами, что после ввода значения в текстовом поле и потере им логического фокуса, например при нажатии клавиши табуляции, положение ползунка изменяется. Если же вы проанализируете разметку, автоматически сформированную в коде XAML, то обнаружите, что свойство `Text` элемента управления типа `TextBox` настроено так, как показано ниже. Обратите, в частности, внимание на атрибут `Mode`.

```
<TextBox ...
  Text="{Binding Value, ElementName=slider3, Mode=TwoWay,
        UpdateSourceTrigger=LostFocus}"/>
```

Примечание. Запустив данное приложение на выполнение, обратите внимание на то, что в текстовом поле отображается числовое значение исходного типа с плавающей точкой из свойства `Value`. Если вам требуется отображать в этом поле только целые числовые значения, то придется создать (и зарегистрировать) еще один класс преобразования данных. Но если вы не умеете программировать на C#, то воспользуйтесь готовым решением данного проекта из архива примеров проектов к этой книге. В этом решении предоставляется весь необходимый исходный код и разметка.

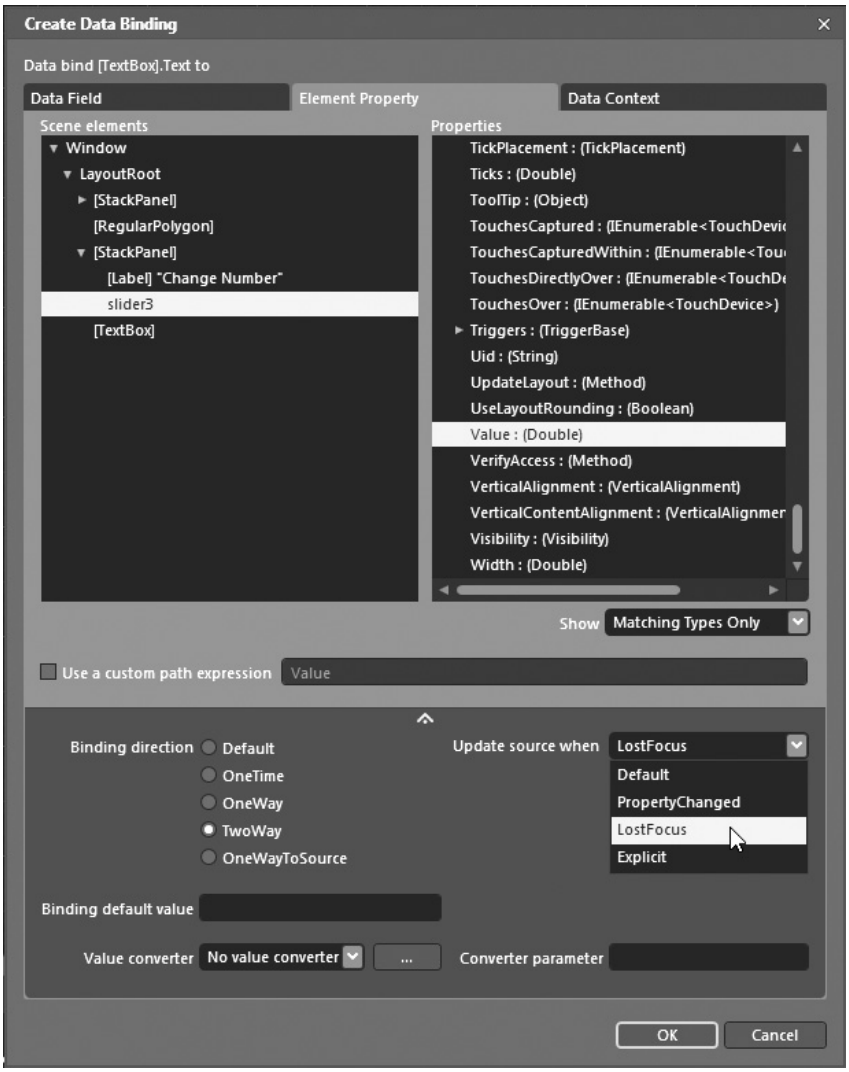


Рис. 6.15. Установка режима двухсторонней привязки данных

К сожалению, здесь недостаточно места для подробного рассмотрения двух других режимов привязки данных: `OneTime` и `OneWayToSource`. Поэтому обращайтесь за дополнительными сведениями к документации на .NET Framework. На этом первый пример, демонстрирующий привязку данных, завершается. Надеюсь, у вас сложилось достаточно полное представление об установлении связей между взаимосвязанными свойствами элементов пользовательского интерфейса. А далее речь пойдет о том, как связываются свойства элементов управления и объектов, не относящихся к пользовательскому интерфейсу.

Исходный код. Исходный код примера проекта `ControlToControlBinding` находится в папке `Ch 6 Code` загружаемого архива примеров проектов к данной книге.

Привязка к свойствам объектов, не относящихся к пользовательскому интерфейсу

В предыдущем примере проекта был продемонстрирован ряд способов связывания значений свойств различных элементов пользовательского интерфейса посредством привязки данных одних элементов управления к другим. Несмотря на всю эффективность подобных способов, во многих случаях требуется привязывать значения свойств объектов, *не* относящихся к пользовательскому интерфейсу, к элементам графического пользовательского интерфейса.

Например, коллекцию бизнес-объектов требуется представить в табличном виде с помощью элемента управления типа `DataGrid`, а данные из XML-документа — в списочном виде с помощью ряда элементов управления `ListBox`. Для того чтобы стало понятнее, каким образом в Expression Blend осуществляется привязка к свойствам объектов, не принадлежащих к пользовательскому интерфейсу, в следующем примере проекта будет продемонстрировано создание коллекции специальных объектов, значения свойств которых требуется отображать в различных элементах пользовательского интерфейса.

Создание отдельной коллекции специальных объектов

Создайте новый проект приложения WPF (или Silverlight), присвоив ему имя `CollectionDataContext`. Введите новый класс `PurchaseOrder` (заказ на поставку) по команде меню Project⇨Add New Item. Обновите этот класс кодом, поддерживающим ряд свойств, выбранных на ваше усмотрение, а также специальным конструктором, в котором устанавливаются значения этих свойств. Ниже приведен один из возможных вариантов написания класса `PurchaseOrder` в исходном коде на C#.

```
public class PurchaseOrder
{
  public PurchaseOrder(){}
  public PurchaseOrder(int amt, double cost, string desc)
  {
    Amount = amt;
    TotalCost = cost;
    Description = desc;
  }

  public int Amount { get; set; }
  public double TotalCost{ get; set; }
  public string Description{ get; set; }
}
```

Конечно, значения трех указанных выше свойств можно связать с элементом пользовательского интерфейса в Expression Blend, опираясь на единственный экземпляр объекта класса `PurchaseOrder`. Но на практике чаще всего требуется привязывать *коллекцию* подобных объектов к элементам пользовательского интерфейса.

Поэтому определите второй класс `PurchaseOrders` (заказы на поставку); обратите внимание на имя этого класса во множественном числе. Этот класс можно добавить в тот же самый файл исходного кода на C# или же ввести его в текущий проект как еще один новый класс по команде меню Project⇨Add New Item.

Обновите этот класс как расширение класса `ObservableCollection`, который удобен для работы с операциями привязки данных, поскольку он отвечает за автоматическое обновление привязок при изменении содержимого (добавлении, удалении или видоиз-

менении отдельных элементов).[7] И наконец, напишите код, обеспечивающий добавление нескольких объектов в коллекцию при ее активизации после запуска приложения. В приведенном ниже примере исходного кода класса `PurchaseOrders` я воздал должное своим четвероногим домашним питомцам, которые были рядом со мной, когда я писал эти строки.

```
public class PurchaseOrders :
  System.Collections.ObjectModel.ObservableCollection<PurchaseOrder>
{
  public PurchaseOrders()
  {
    // Добавить в коллекцию несколько элементов после запуска.
    this.Add(new PurchaseOrder(5, 50.00, "Mikko's Cat Nip Treat"));
    this.Add(new PurchaseOrder(5, 50.00, "Saku's Best Dog Bone"));
    this.Add(new PurchaseOrder(1, 2.50, "Extra Bland Tofu"));
  }
}
```

А теперь, когда создана специальная коллекция бизнес-объектов, можно построить пользовательский интерфейс, в котором будут отображаться данные каждого члена коллекции. (Напомним, что в рассматриваемом здесь примере проекта нужно увязать свойства отдельных объектов типа `PurchaseOrder` из этой коллекции со свойствами элементов пользовательского интерфейса.) В частности, свойство каждого объекта типа `PurchaseOrder` можно привязать к ряду раскрывающихся списков или же всю коллекцию к табличному виду типа `DataGrid`. Так или иначе, решение задачи такой привязки данных в среде Expression Blend IDE существенно упрощается с помощью панели Data.

Определение источника данных объекта на панели Data

Если в Expression Blend требуется привязать специальный объект к пользовательскому интерфейсу, например, коллекцию типа `PurchaseOrders` из рассматриваемого здесь примера проекта, то для этой цели следует сначала перейти к панели Data, чтобы определить новый *источник данных объекта*. Итак, перейдите к панели Data, располагаемой по умолчанию с правой стороны рабочего пространства Expression Blend, щелкните на кнопке раскрывающегося списка Create data source и выберите вариант Create Object Data Source (Создать источник данных объекта) из раскрывающегося списка (рис. 6.16).

В итоге откроется новое диалоговое окно, в котором можно выбрать самые разные типы объектов .NET, в том числе и любые специальные классы, включенные в состав текущего проекта.

Примечание. Если вы не увидите в этом окне специальный класс, введенный вами в проект, значит, вы просто забыли создать свой проект!

Как показано на рис. 6.17, все узлы стандартных библиотек свернуты для большей удобочитаемости, а специальные классы представлены сборками того же наименования. Итак, выберите коллекцию типа `PurchaseOrders` и обратите внимание на то, что в верхней части текущего диалогового окна автоматически указывается предполагаемое имя `PurchaseOrdersDataSource` нового источника данных. Это очень удобно, хотя по желанию можете переименовать его.

---

[7] Класс `ObservableCollection` относится к пространству имен `System.Collections.ObjectModel`.

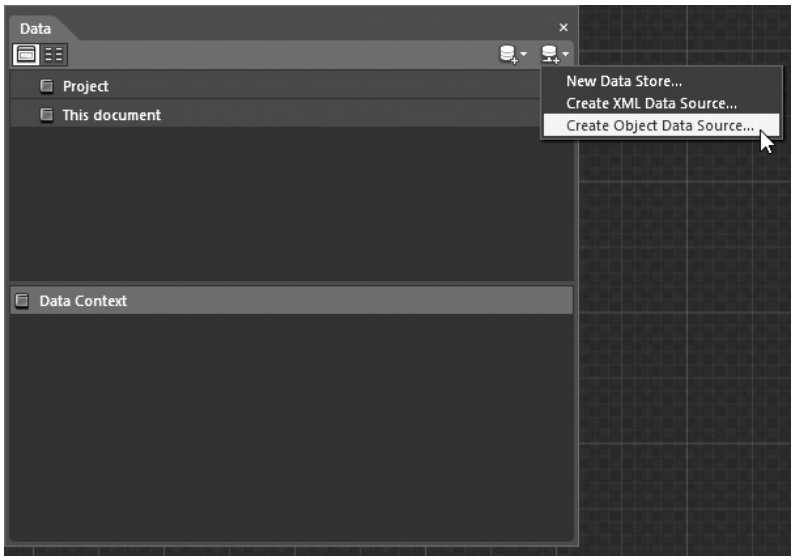


Рис. 6.16. На панели Data можно определить источники данных для текущего проекта

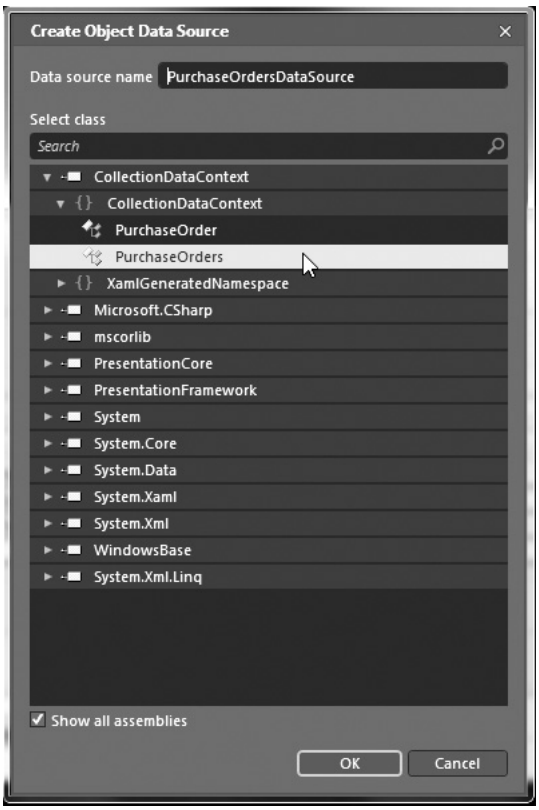


Рис. 6.17. Выбор коллекции типа `PurchaseOrders` в качестве нового источника данных

Щелкнув на кнопке OK, проанализируйте содержимое панели Data, приведенной на рис. 6.18. На этой панели вы обнаружите новый источник данных объекта со всеми свойствами объектов, составляющих выбранную вами коллекцию.

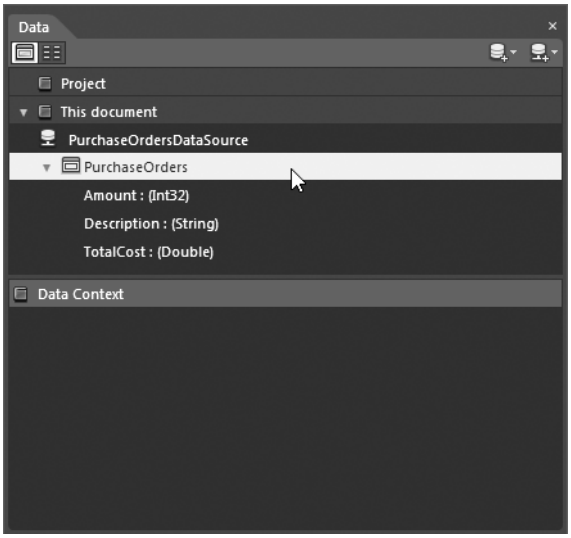


Рис. 6.18. На панели Data отображаются все источники данных, применяемые в текущем проекте

Если вы проанализируете теперь разметку исходного окна типа `Window` (или элемента управления типа `UserControl` на начальной веб-странице, при условии, что работаете над данным проектом на платформе Silverlight), то непременно заметите, что в среде Expression Blend IDE автоматически определен новый ресурс объекта (подробнее о таких ресурсах см. в главе 2), увязанный с объектом специальной коллекции, как показано ниже.

```
<Window.Resources>
  <local:PurchaseOrders x:Key="PurchaseOrdersDataSource"
                        d:IsDataSource="True"/>
</Window.Resources>
```

Именно этот ресурс и будет применяться в операции привязки данных.

Привязка всей коллекции к списку

На данном этапе работы над проектом можете определить пользовательский интерфейс, в котором отображаются значения свойств, получаемые из источника данных объекта. Проще говоря, после определения источника данных на панели Data любой элемент отображаемого иерархического представления может быть перенесен методом перетаскивания на монтажный стол для формирования нового элемента пользовательского интерфейса, который автоматически привязывается к этому источнику данных.

Если вы посмотрите на левый верхний угол панели Data, то обнаружите там две кнопки. Левая кнопка, выбираемая по умолчанию, активизирует режим отображения списком (List Mode) выбранного в настоящий момент источника данных. Когда источник данных установлен в режим отображения списком, можете перетащить отдельное свойство или объект из панели Data, чтобы отобразить его в новом списке, представленном элементом управления типа `ListBox`. А правая кнопка настраивает источник данных на работу в режиме отображения подробностей (Details Mode), рассматриваемом далее в главе.

Допустим, вы находитесь в устанавливаемом по умолчанию режиме отображения списком. Если вы выберете узел `PurchaseOrders` на панели Data и перетащите его на монтажный стол, то обнаружите на нем один добавленный элемент управления типа `ListBox`, в котором отображаются все свойства каждого объекта из выбранной коллекции (рис. 6.19).

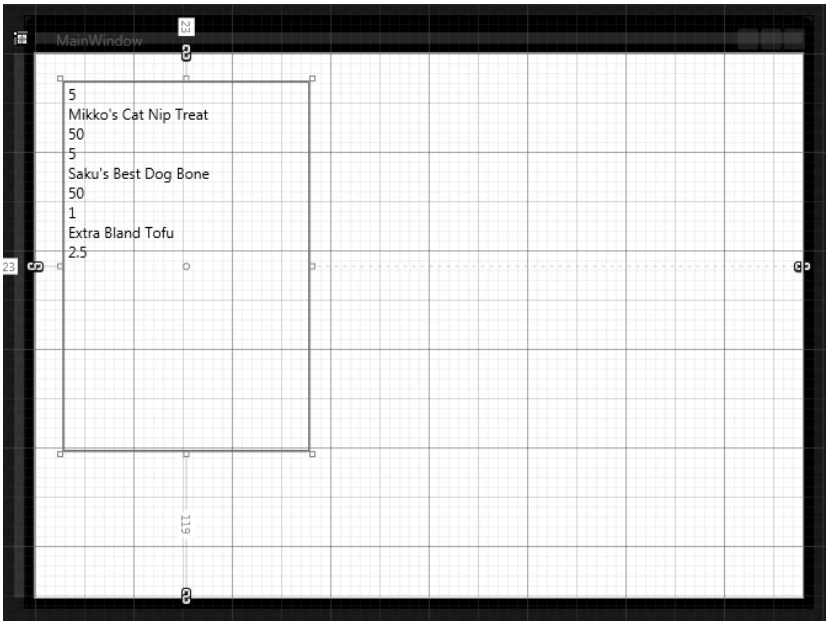


Рис. 6.19. В одном элементе управления типа `ListBox` содержатся данные всех объектов из коллекции

На первый взгляд, привязывать всю коллекцию объектов к единственному элементу управления типа `ListBox` нецелесообразно, поскольку в каждом элементе списка значения свойств каждого объекта отображаются подряд. В самом деле, такой пользовательский интерфейс не очень удобен для отображения данных привязываемого к нему источника. Впрочем, на обеих платформах, WPF и Silverlight, предоставляется возможность создавать *шаблоны данных*, которые определяют порядок отображения данных из источника, привязываемого к целевому элементу пользовательского интерфейса.

В частности, можно создать такой шаблон данных, чтобы вместо следующих подряд строк данных отображались дополнительные фрагменты описательного текста, графики, анимации и прочих элементов оформления. По существу, все, что делается при построении шаблона элемента управления (см. главу 5), можно сделать и в операции привязки данных. Более подробно шаблоны данных рассматриваются далее в главе, а до тех пор нужно пояснить назначение *контекста данных*.

Назначение контекста данных

Если вы откроете редактор XAML, то заметите, что в элементе разметки `<Grid>`, описывающем диспетчер компоновки `LayoutRoot`, появился новый атрибут `DataContext`, которому присваивается выражение, привязывающее этот диспетчер к ресурсу объекта `PurchaseOrdersDataSource`, применяемому на уровне главного окна типа `Window`. Когда источник данных устанавливается в атрибуте `DataContext` диспетчера компоновки, любой порожденный элемент может использовать этот источник данных (в данном случае коллекцию типа `PurchaseOrders`) при выполнении операции привязки данных.

```
<Grid x:Name="LayoutRoot" DataContext="{Binding Source=
     {StaticResource PurchaseOrdersDataSource}}">
     ...
</Grid>
```

Обратите также внимание на то, что в элементе разметки `<Grid>` устанавливаются два представляющих особый интерес свойства элемента управления типа `ListBox`: `ItemsSource` (Источник для элементов) и `ItemTemplate` (Шаблон элемента). В частности, свойство `ItemsSource` может выглядеть в разметке не совсем обычно, поскольку оно описывается только пустым расширением разметки `{Binding}`. Но это просто означает, что источник данных привязывается к элементу управления типа `ListBox` в результате операции, которая на самом деле определяется связанным с ним шаблоном данных.

```
<ListBox ItemTemplate="{DynamicResource PurchaseOrderTemplate}"
       ItemsSource="{Binding}"
  ...
/>
```

Разметка шаблона данных с первого взгляда

Ресурс, указываемый в свойстве `ItemTemplate`, представляет собой новый ресурс объекта, помещаемый на свое место в результате перетаскивания узла `PurchaseOrders` из панели Data на монтажный стол. Не стоит пока что анализировать подробно приведенную ниже разметку, поскольку речь о шаблонах данных пойдет несколько позже.

```
<DataTemplate x:Key="PurchaseOrderTemplate">
  <StackPanel>
    <TextBlock Text="{Binding Amount}"/>
    <TextBlock Text="{Binding Description}"/>
    <TextBlock Text="{Binding TotalCost}"/>
  </StackPanel>
</DataTemplate>
```

Просмотр привязываемого контекста данных

И последнее, но не менее важное замечание: если еще раз проанализировать содержимое панели Data, то в нижней ее части можно обнаружить новый узел контекста данных (рис. 6.20). По существу, в списках, находящихся в верхней части панели Data, отображаются все источники данных, которые *могут* быть использованы в текущем проекте, тогда как в нижней части этой панели — источники данных, которые *фактически* используются как контексты данных.

Быстрый возврат проекта в исходное состояние

А теперь, когда стало понятнее, что именно формируется в Expression Blend при переносе всей коллекции на монтажный стол, нажмите комбинацию клавиш <Ctrl+Z>, чтобы вернуться к исходному внешнему виду главного окна приложения. После этого убедитесь в том, что в разметке отсутствует описание элемента управления типа `ListBox`, но сохранилось описание источника данных объекта, как показано ниже.

```
<Window
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="CollectionDataContext.MainWindow"
  x:Name="Window"
  Title=" MainWindow"
  Width="640" Height="480">

  <Window.Resources>
    <local:PurchaseOrders x:Key="PurchaseOrdersDataSource"
```

```
                    d:IsDataSource="True"/>
  </Window.Resources>

  <Grid x:Name="LayoutRoot"/>
</Window>
```

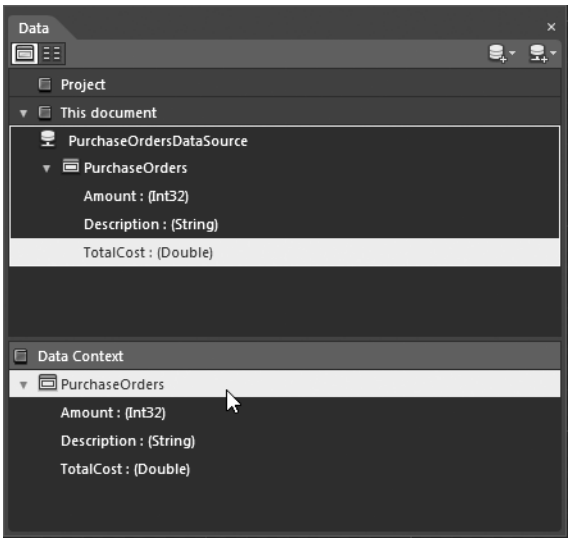


Рис. 6.20. На панели Data отображаются действующие контексты данных

Привязка отдельных свойств к элементам управления типа `ListBox`

Если перетащить отдельное свойство из панели Data на монтажный стол, на нем появится новый элемент управления типа `ListBox`, при условии, что это делается в режиме отображения списком, устанавливаемом на панели Data по умолчанию. Допустим, требуется определить три элемента управления типа `ListBox`, которые должны содержать значения свойств каждого объекта типа `PurchaseOrder` из коллекции `PurchaseOrders`. Для этого выберите свойство `Amount` на панели Data и перетащите его на монтажный стол текущего окна типа `Window`. В итоге на монтажном столе появится один автоматически сформированный элемент управления типа `ListBox`. В качестве примера на рис. 6.21 показан вид монтажного стола после переноса на него свойств `Amount` (Количество), `Description` (Описание) и `TotalCost` (Общая сумма). (Для большей наглядности данного примера размеры текущих объектов типа `Window` и `Grid` были немного изменены.)

Рис. 6.21. Три элемента управления типа `ListBox`, привязанных к свойствам специальных объектов

Привязка коллекции объектов к элементу управления типа DataGrid

Ранее было наглядно продемонстрировано, что, если перетащить отдельное свойство или всю коллекцию объектов из панели Data на монтажный стол, соответствующие данные появятся в привязываемых к ним по умолчанию элементах управления типа `ListBox`. Разумеется, эти данные можно привязать на панели Data и к другим элементам пользовательского интерфейса, при условии, что они предварительно введены на монтажном столе.

В качестве примера найдите элемент управления типа `DataGrid` в библиотеке ресурсов и перетащите его на монтажный стол текущего окна типа `Window`, расположив его чуть ниже уже находящихся там элементов управления типа `ListBox`. Затем выберите узел `PurchaseOrders` на панели Data вместо отдельных свойств, находящихся ниже этого узла, и перетащите его на новый элемент управления типа `DataGrid`. В итоге содержимое всей коллекции объектов автоматически отобразится в табличном виде, как показано на рис. 6.22.

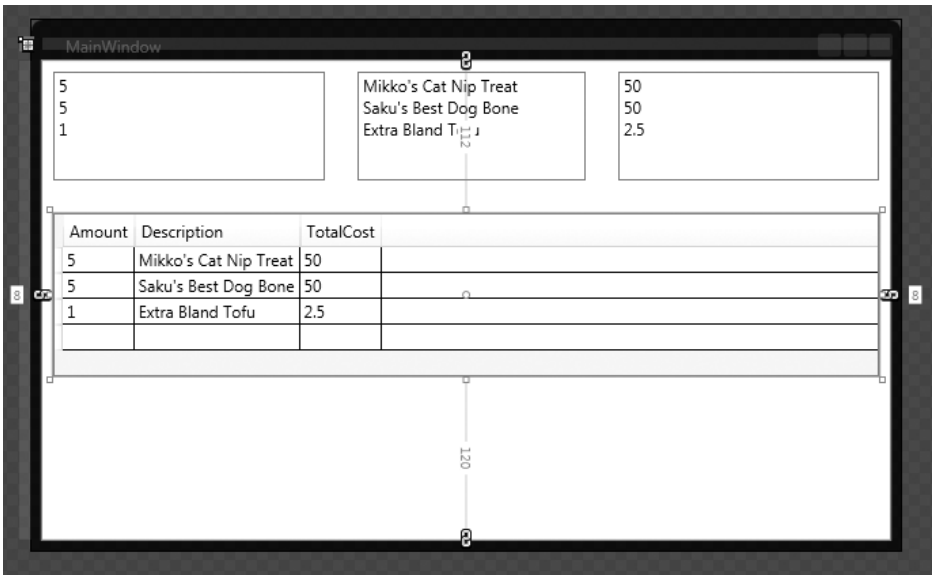


Рис. 6.22. Привязка всей коллекции к элементу пользовательского интерфейса типа `DataGrid`

Если вы проанализируете сформированную в итоге разметку, то обнаружите, что описание элемента управления типа `DataGrid` по-прежнему опирается на контекст данных, определяемый диспетчером компоновки `LayoutRoot`. Но на этот раз свойства объектов из коллекции увязаны с отдельными элементами разметки `<DataGridTextColumn>`, описывающими столбцы таблицы, как показано ниже.

```
<Grid x:Name="LayoutRoot" DataContext="{Binding
     Source={StaticResource PurchaseOrdersDataSource}}">
...
  <DataGrid Margin="8,112,8,120" AutoGenerateColumns="False"
          ItemsSource="{Binding}">
    <DataGrid.Columns>
      <DataGridTextColumn Binding="{Binding Amount}" Header="Amount"/>
```

```
      <DataGridTextColumn Binding="{Binding Description}"
                          Header="Description"/>
      <DataGridTextColumn Binding="{Binding TotalCost}"
                          Header="TotalCost"/>
    </DataGrid.Columns>
  </DataGrid>
</Grid>
```

Манипулирование коллекцией объектов во время выполнения

Напомним, что рассматриваемая здесь специальная коллекция объектов типа `PurchaseOrders` является расширением класса `ObservableCollection`. Поэтому она обладает особой способностью обновлять привязки данных к пользовательскому интерфейсу при изменении содержимого коллекции (добавлении, удалении или обновлении ее элементов). В завершение рассматриваемого здесь примера проекта добавьте элемент управления типа `Button` на монтажном столе текущего окна типа `Window` и организуйте обработку события `Click`, связанного с этим элементом управления, на панели Properties.

Напомним, что когда был определен источник данных объекта, в главное окно типа `Window` был автоматически добавлен ресурс объекта `PurchaseOrdersDataSource`, увязываемый с экземпляром коллекции типа `PurchaseOrders`. При запуске приложения на выполнение объект этой коллекции создается с помощью используемого по умолчанию конструктора[8]. Поэтому, если требуется получить доступ к экземпляру объекта того же самого класса, придется искать нужный элемент по ключевому имени в контейнере ресурсов. В качестве примера ниже приведен исходный код обработчика событий типа `Click`, в котором новый объект типа `PurchaseOrder` добавляется в коллекцию.

```
private void btnAddNewObject_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
  // Сначала получить нужный ресурс объекта.
  PurchaseOrders myOrders =
    (PurchaseOrders)this.Resources["PurchaseOrdersDataSource"];

  // Затем сформировать случайные значения для числовых свойств.
  Random r = new Random();
  int amount = r.Next(50);
  double cost = r.NextDouble();

  // И наконец, добавить новый элемент с проверкой на случайные числа.
  myOrders.Add(new PurchaseOrder(amount, cost, "TEST ITEM!"));
}
```

В этом обработчике событий генерируется ряд случайных данных для проверки количества и стоимости заказанного товара, но вы вольны расширить рассматриваемый здесь проект рядом новых элементов управления вводом текста для сбора особых данных. Так или иначе, если вы запустите свое приложение на выполнение и щелкнете несколько раз на кнопке, представленной элементом управления типа `Button`, то заметите, как автоматически обновляются таблица и списки, представленные элементами управления типа `DataGrid` и `ListBox` соответственно (рис. 6.23).

[8] Для создания любого объекта из разметки можно вызывать только конструктор, используемый по умолчанию.

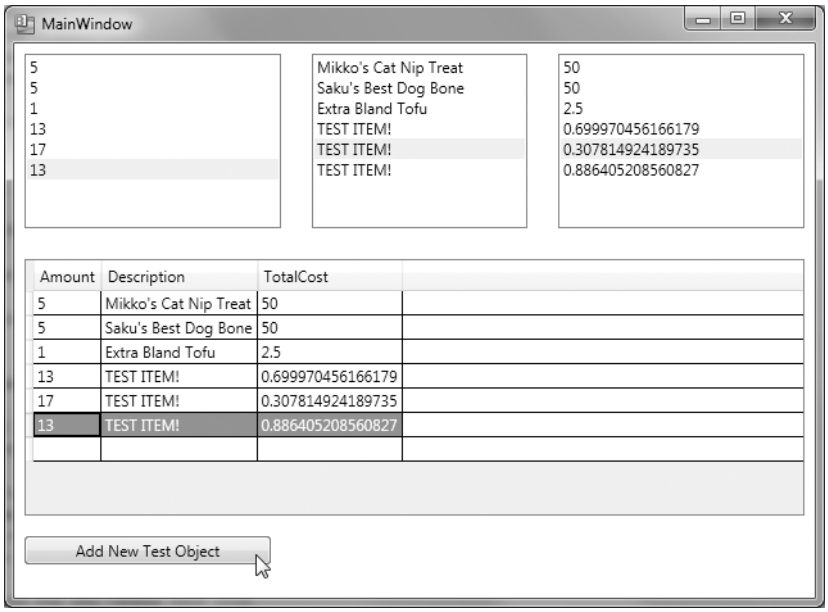


Рис. 6.23. Ввод, повторная привязка и отображение данных заказа

На этом рассмотрение примера привязки коллекции специальных объектов к элементам пользовательского интерфейса завершается. Далее будет пояснено назначение шаблонов данных.

Исходный код. Исходный код примера проекта `CollectionDataContext` находится в папке `Ch 6 Code` загружаемого архива примеров проектов к данной книге.

Работа с шаблонами данных

В следующем примере созданные ранее классы `PurchaseOrder` и `PurchaseOrders` будут вновь использованы для того, чтобы продемонстрировать, насколько упрощается создание специальных шаблонов данных в Expression Blend. С этой целью создайте новый проект приложения WPF, присвоив ему имя `FunWithDataTemplates`. Выберите из главного меню команду Project⇨Add Existing Item (Проект⇨Добавить существующий элемент), чтобы скопировать из предыдущего проекта файлы исходного кода C#, в которых определяются бизнес-объекты и их специальная коллекция.

После создания нового проекта перетащите узел `PurchaseOrders` из панели Data на монтажный стол главного окна типа `Window`, чтобы установить связь с этим источником данных объекта, как это делалось в предыдущем примере проекта. В итоге главное окно вашего приложения должно выглядеть таким же образом, как и на рис. 6.19. Напомним, что в среде Expression Blend IDE автоматически формируется используемый по умолчанию шаблон данных в качестве ресурса, применяемого на уровне главного окна приложения и описываемого в файле разметки `MainWindow.xaml`. Именно этот шаблон используется в элементе управления типа `ListBox` для отображения значений отдельных свойств из коллекции объектов в простых, нестилизованных элементах списка типа `TextBlock`:

```
<DataTemplate x:Key="PurchaseOrderTemplate">
  <StackPanel>
    <TextBlock Text="{Binding Amount}"/>
```

```
    <TextBlock Text="{Binding Description}"/>
    <TextBlock Text="{Binding TotalCost}"/>
  </StackPanel>
</DataTemplate>
```

Правка шаблона данных

Если требуется видоизменить используемый по умолчанию шаблон данных, перейдите сначала к панели Resources и найдите нужный шаблон по имени (рис. 6.24). Затем откройте его для правки, щелкнув на кнопке Edit resource.

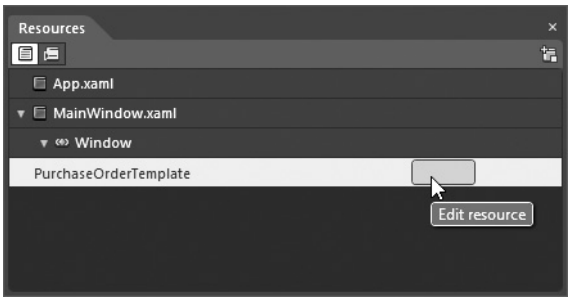


Рис. 6.24. Поиск шаблона данных на панели Resources

Перейдите к панели Objects and Timeline и выберите первый элемент списка типа `TextBlock`, как показано на рис. 6.25.

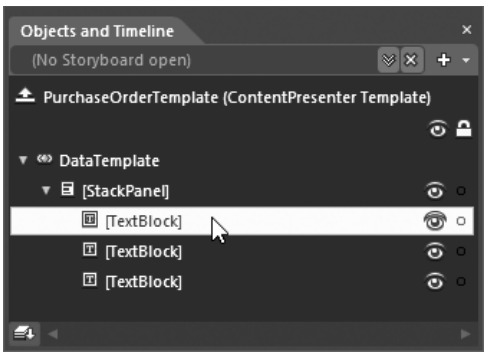


Рис. 6.25. Выбор первого элемента списка типа `TextBlock` в шаблоне данных для последующей правки

Стилевое оформление элементов списка в шаблоне данных

На данном этапе работы над рассматриваемым здесь примером проекта можете воспользоваться панелью Properties, чтобы внести изменения в любые свойства объекта типа `TextBlock`, чтобы определить стиль его оформления для отображения привязываемого к нему значения свойства `Amount`. В частности, можете выбрать другой цвет переднего плана в свойстве `Foreground` или внести изменения в шрифтовое оформление в области Text данной панели. В качестве примера ниже приведена разметка, в которой описывается стилевое оформление первого элемента списка типа `TextBlock`.

```
<TextBlock Text="{Binding Amount}" Foreground="#FFE91616"
        FontWeight="Bold" FontSize="18.667"/>
```

Если вы запустите свое приложение на выполнение, то обнаружите, что отображаемые в списке значения свойства `Amount` выглядят теперь более привлекательно, чем прежде (рис. 6.26).

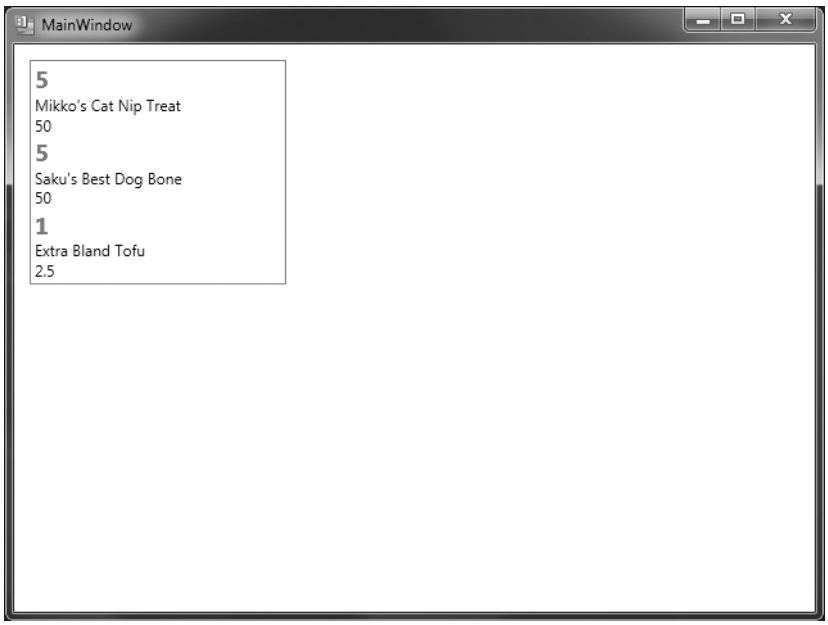


Рис. 6.26. Первая попытка стилевого оформления в шаблоне данных

Определение составных элементов пользовательского интерфейса для шаблона данных

Конечно, выделение цветом улучшает внешний вид отображаемых данных, но он все равно остается довольно неопределенным. Что, например, означает цифра 5, выделенная красным цветом? Поэтому вернитесь к рассматриваемому здесь шаблону на панели Objects and Timeline и еще раз выберите первый элемент списка типа `TextBlock`. Щелкните правой кнопкой мыши на узле этого объекта и сгруппируйте его в новую вложенную блочную панель типа `StackPanel` по команде Group Into⇨StackPanel, выбираемой из всплывающего контекстного меню.

После этого установите значение **`Horizontal`** в свойстве `Orientation` вновь созданного элемента управления типа `StackPanel`. И наконец, расположите новый элемент типа `Label` выше уже имеющегося элемента списка типа `TextBlock`. (Напомним: можно перетаскивать узлы иерархического представления на новые места.) По завершении иерархическое представление на панели Objects and Timeline должно выглядеть так, как показано на рис. 6.27.

Прежде чем переходить к следующему этапу работы над данным проектом, введите в свойстве `Content` элемента управления типа `Label` текстовую надпись **`Amount`**. Если вы внимательно исследуете редактор текущего шаблона данных, то заметите, что все его элементы отображаются в мелком масштабе, поэтому воспользуйтесь элементами управления в левом нижнем углу монтажного стола (или колесиком мыши), чтобы увеличить масштаб. Благодаря этому вы обнаружите, что новый вложенный диспетчер компоновки и его элемент управления расположены неверно. Для исправления этого недостатка установите значение **70** в свойстве элемента управления типа `Label`.

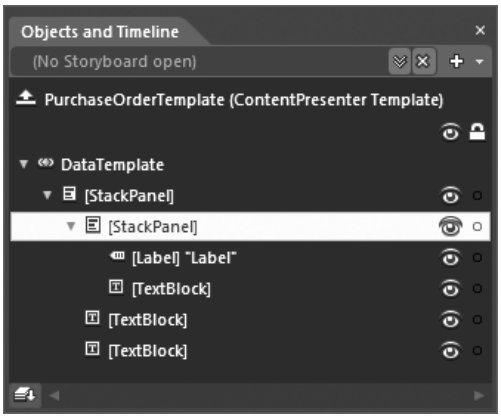


Рис. 6.27. Первый элемент списка теперь оформлен в шаблоне данных как вложенная блочная панель типа `StackPanel`

Оформите в текущем шаблоне данных два других элемента списка типа `TextBlock` по описанной выше общей процедуре. И в этом случае вам, скорее всего, придется сгруппировать каждый такой элемент во вложенный диспетчер компоновки по команде Group Into, выбираемой из контекстного меню на панели Objects and Timeline, а затем добавить к нему несколько других элементов по своему усмотрению.

В качестве примера на рис. 6.28 приведен окончательный вид панели Objects and Timeline (и части увеличенного монтажного стола) для рассматриваемого здесь шаблона данных. Обратите внимание на то, что вложенные объекты типа `StackPanel` переупорядочены таким образом, чтобы количество и стоимость товаров перечислялись до их описания. А для специального указания на описание товара в данный шаблон введен графический элемент, изображающий маленькую стрелку.

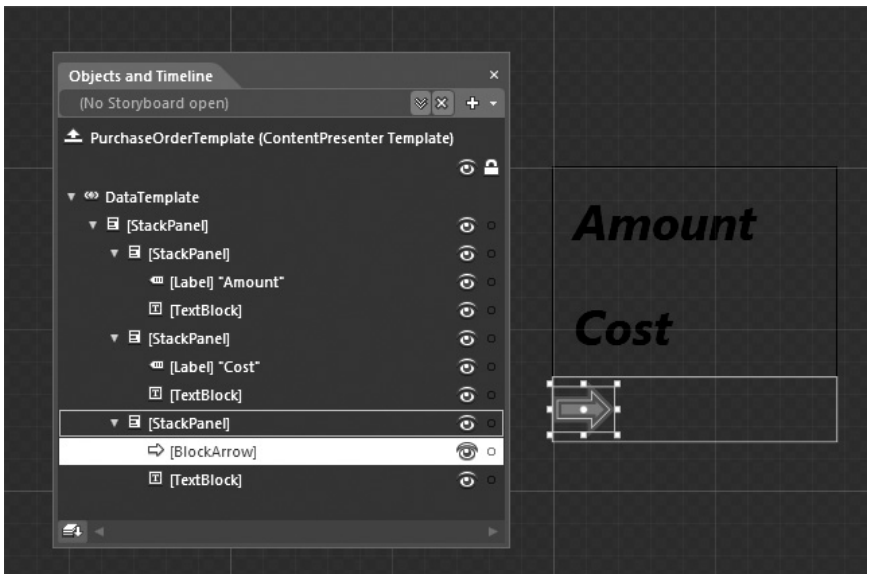


Рис. 6.28. Окончательный вид компоновки по шаблону данных

Ниже приведена исходная разметка, описывающая рассматриваемый здесь шаблон данных в коде XAML. И хотя она может оказаться у вас несколько иной, общая идея должна быть вам ясна.

```
<DataTemplate x:Key="PurchaseOrderTemplate">
  <StackPanel>
    <StackPanel Orientation="Horizontal" >
      <Label Width="70" Content="Amount" BorderBrush="#FF309914"
           FontWeight="Bold" FontStyle="Italic"/>
      <TextBlock Text="{Binding Amount}" Foreground="#FFE91616"
               FontWeight="Bold" FontSize="18.667"
               d:LayoutOverrides="Width"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal" >
       <Label Width="70" Content="Cost" FontWeight="Bold"
            FontStyle="Italic"/>
      <TextBlock Text="{Binding TotalCost}" Foreground="#FF1031E7"
               FontWeight="Bold" FontSize="18.667"
               d:LayoutOverrides="Width"/>
    </StackPanel>
    <StackPanel Orientation="Horizontal">
       <ed:BlockArrow Fill="#FF5A5AC2" Height="11" Orientation="Right"
                  Stroke="#FF1030E4" Width="15"
                  StrokeThickness="2"/>
      <TextBlock Text="{Binding Description}"
               d:LayoutOverrides="Width"
               Foreground="#FFAD11DE"/>
    </StackPanel>
  </StackPanel>
</DataTemplate>
```

Помните, что во время правки любого шаблона данных вы всегда можете вернуться к визуальному конструктору главного окна (объекта типа `Window`) или элемента управления на начальной веб-странице (объекта типа `UserControl`), щелкнув на кнопке Return scope to Window/UserControl (Вернуться в пределы окна или элемента пользовательского интерфейса), как показано на рис. 6.29.

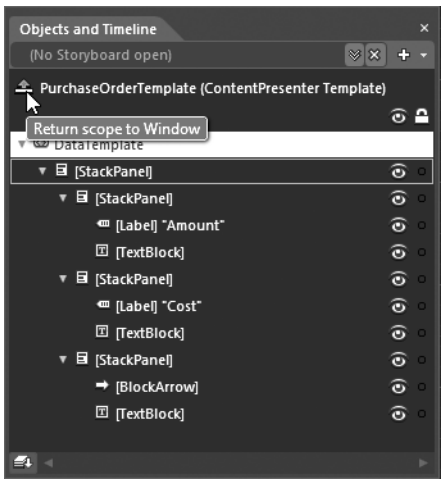


Рис. 6.29. Возврат к визуальному конструктору окна из визуального конструктора шаблона

Итак, вернитесь к монтажному столу, чтобы посмотреть результаты правки шаблона данных (рис. 6.30; цветная вклейка). Если вы запустите теперь свое приложение на выполнение, то сможете выбрать нужный элемент из списка, оформленного по рассматриваемому здесь шаблону данных.

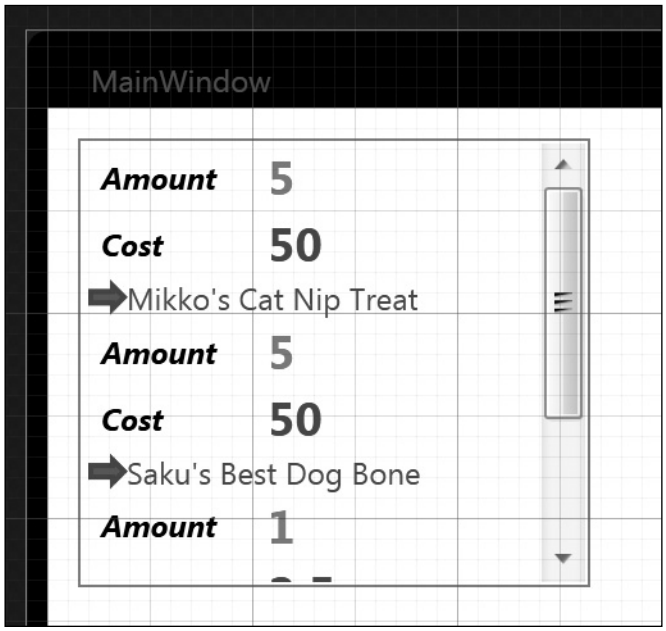


Рис. 6.30. Окончательный вид списка, оформленного по шаблону данных

Создание шаблонов элементов управления, содержащих шаблоны данных

Завершая рассмотрение специальных шаблонов данных, следует особо отметить возможность свободно создавать шаблоны элементов управления (см. главу 5), содержащие *встраиваемые* шаблоны данных. А для чего это вообще нужно? Допустим, что, когда пользователь выбирает один из элементов из списка типа `ListBox`, требуется, чтобы выбранный элемент данных типа `StackPanel` каким-то образом изменил свой внешний вид, например, с помощью анимации отскока или как-то иначе. А таких списков в окне может быть несколько.

Примечание. Далее в этом разделе будет использована панель States, рассматривавшаяся в главе 5. Поэтому обращайтесь за справкой к разделу "Работа с диспетчером VSM на панели States" этой главы, если возникнет такая необходимость.

Если требуется ввести интерактивные средства в шаблон данных, то можно, в частности, щелкнуть правой кнопкой мыши на том элементе управления, к которому в настоящий момент применяется шаблон данных (в данном случае — `ListBox`), а затем выбрать команду Edit Additional Templates (Править дополнительные шаблоны) из всплывающего контекстного меню и далее — команду редактирования копии сформированного контейнера элементов (рис. 6.31).

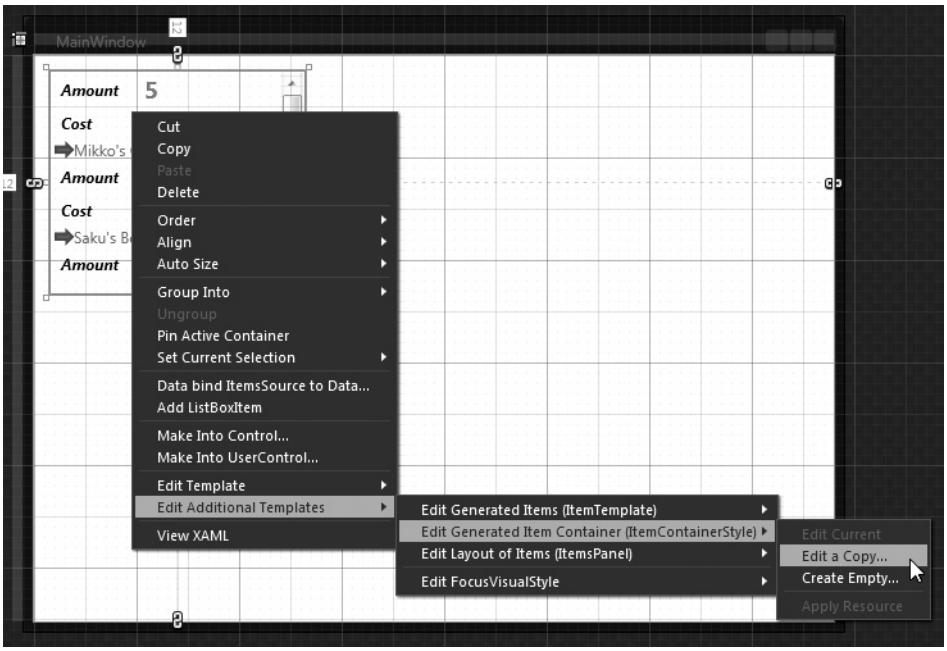


Рис. 6.31. Правка шаблона контейнера элементов из списка типа `ListBox`

При выборе такой возможности в среде Expression Blend IDE формируется новый стиль оформления элемента управления типа `ListBox`. А это, в свою очередь, дает возможность определить в элементе разметки `<ContentPresenter>` порядок отображения данных при разных условиях, например, когда элемент выбран из списка, когда элемент не выбран, когда он получает логический фокус и многие другие уведомления. Итак, выберите указанную выше команду, в открывшемся диалоговом окне Create Style Resource присвойте редактируемому стилю имя `myListStyle` и сохраните его в качестве ресурса, применяемого на уровне главного окна типа `Window` (рис. 6.32).

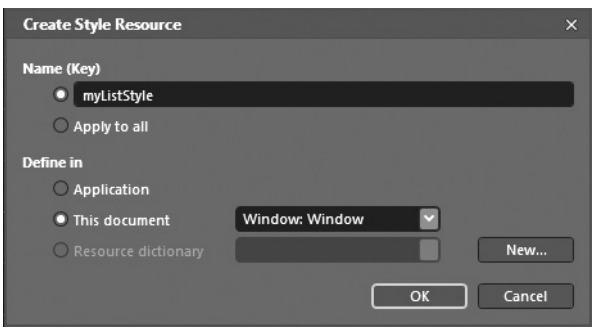


Рис. 6.32. Присвоение имени и сохранение нового ресурса стиля

Ввод интерактивных средств в шаблон

Перейдя к панели Objects and Timeline, вы обнаружите на ней внутренний узел `<ContentPresenter>`. Выберите этот узел в иерархическом представлении. Напомним, что ваша задача — ввести в шаблон элемента управления немного интерактивных средств,

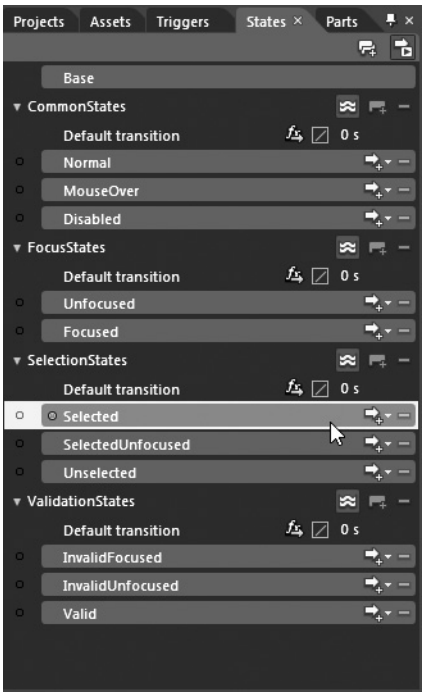


Рис. 6.33. Визуальные состояния, доступные по умолчанию для шаблона элемента управления типа `ListBox`

и для ее решения вам придется обратиться к триггерам или диспетчеру визуальных состояний (VSM), как пояснялось в главе 5. В рассматриваемом здесь примере проекта будет использован диспетчер VSM.

Откройте панель States, на которой вы обнаружите целый список доступных по умолчанию визуальных состояний. Выберите из этого списка состояние `Selected` (рис. 6.33), чтобы открыть редактор состояний.

Выбрав состояние `Selected`, можете внести любые изменения в узле `<ContentPresenter>`. В качестве примера на рис. 6.34 показано уменьшение масштаба по осям X и Y. Для этого в полях X и Y на вкладке Scale из области Transform установлены значения **0,85**.

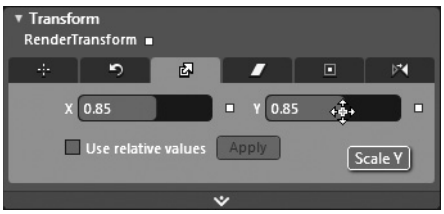


Рис. 6.34. Сокращение размеров выбранного элемента списка

Затем можете применить эффект отскакивания. Для этого введите сначала новый переход в состояние `Selected` на панели States, как показано на рис. 6.35.

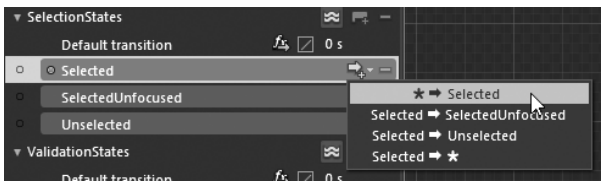


Рис. 6.35. Ввод перехода в новое состояние

Далее воспользуйтесь соответствующим редактором анимации эффектов инерционности движения, чтобы ввести эффект отскакивания и отредактировать его по своему усмотрению (рис. 6.36).

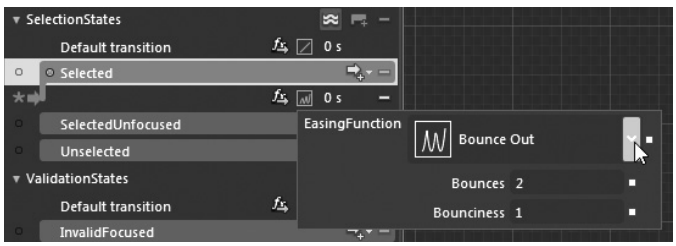


Рис. 6.36. Применение эффекта отскакивания при переходе в состояние `Selected`

И последнее, но не менее важное действие: установите подходящую продолжительность анимации данного эффекта, поскольку нулевое значение означает, что анимация вообще не будет видна! В качестве примера на рис. 6.37 показано, что продолжительность анимации эффекта отскакивания выбрана равной 1 секунде.

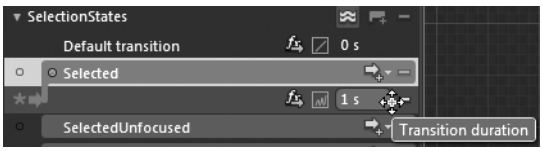


Рис. 6.37. Установка подходящей продолжительности анимации эффекта отскакивания

Запустите свое приложение на выполнение и попробуйте выбрать отдельные элементы из списка. При этом вы должны наблюдать изящный эффект отскакивания всех элементов, составляющих список и определенных по шаблону данных. Насколько полезным оказывается подобный эффект при выборе элементов из списка, судите сами.

В рассматриваемом здесь примере проекта были продемонстрированы довольно интересные способы специальной настройки операций привязки данных. На этом примере вы смогли сами убедиться в том, что стилевое оформление шаблонов данных можно осуществить самыми разными способами. Надеюсь, что этот пример поможет вам лучше понять особенности специальной настройки операций привязки данных и стилевого оформления шаблонов данных в среде Expression Blend IDE.

Исходный код. Исходный код примера проекта `FunWithDataTemplates` находится в папке `Ch 6 Code` загружаемого архива примеров проектов к данной книге.

Определение источника данных XML на платформе WPF

Если вы занимаетесь разработкой прикладных программ на платформе .NET, то хорошо знаете, что на ней поддерживаются различные программные средства для манипулирования данными в формате XML, в том числе исходное пространство имен `System.Xml` и прикладной интерфейс LINQ to XML API. Так, если данные XML требуется видоизменить во время выполнения приложений на платформе WPF или Silverlight, например, добавить, обновить или видоизменить новые элементы, для этого придется прибегнуть к написанию процедурного кода. Но в среде Expression Blend IDE предоставляется также возможность привязывать элементы пользовательского интерфейса к данным, содержащимся в XML-документе, используя источник данных XML, выбираемый на панели Data.

Примечание. К сожалению, возможность привязывать данные XML к элементам пользовательского интерфейса с помощью источника данных XML имеется только для приложений на платформе WPF. Разумеется, данными XML можно манипулировать и в приложениях на платформе Silverlight, но для этого придется написать процедурный код. За дополнительными сведениями и примерами манипулирования данными XML в приложениях на платформе Silverlight обращайтесь к документации по Silverlight.

Учитывая, что привязка данных из XML-документа доступна только на платформе WPF, создайте новый проект приложения WPF, присвоив ему имя `WpfXmlDataBinding`. Если у вас имеется XML-документ, данными из которого вы хотели бы манипулировать, вы, конечно, вольны воспользоваться именно им, указав его файл с расширением `*.xml` в своем проекте. Но для целей рассматриваемого здесь примера выбран файл `Inventory.xml`, который находится в подпапке проекта `Ch 6 Code\WpfXmlDataBinding\WpfXmlDataBinding` загружаемого архива примеров проектов к данной книге. Ниже приведено содержимое этого файла.

```
<?xml version="1.0" encoding="utf-8"?>
<Inventory>
  <Product ProductID ="0">
    <Cost>5.00</Cost>
    <Description>
      Eight Times the sugar and twice the caffeine
    </Description>
    <Name>Super Spazz Soda Pop</Name>
    <HotItem>true</HotItem>
  </Product>
  <Product ProductID ="1">
    <Cost>10.00</Cost>
    <Description>A soothing night time cookie</Description>
    <Name>Sleepy Time Cookies</Name>
    <HotItem>true</HotItem>
  </Product>
  <Product ProductID ="2">
    <Cost>15.00</Cost>
    <Description>It's Tofu, what can you say?</Description>
    <Name>Joe's Tofu</Name>
    <HotItem>false</HotItem>
  </Product>
</Inventory>
```

Введите этот файл (или свой собственный XML-файл) в текущий проект по команде меню Project⇨Add Existing Item. В итоге этот файл появится на панели Projects.

Примечания. Для привязки данных из XML-файла на панели Data его совсем не обязательно включать в проект, но благодаря этому упрощается открытие данного файла для последующей правки в среде Expression Blend IDE.

Ввод источника данных XML

Сначала откройте файл разметки главного окна `MainWindow.xaml` в окне монтажного стола, а затем перейдите к панели Data и введите новый источник данных XML, как показано на рис. 6.38.

Найдите XML-документ, который хотели бы использовать в своем проекте (в данном случае это файл `Inventory.xml`), в открывшемся окне Create XML Data Source (Создание источника данных XML), как показано на рис. 6.39. Присвойте новому источнику данных XML имя `InventoryXmlDataStore` и сохраните его в качестве ресурса объекта в текущем проекте.

Щелкните на кнопке OK. В данный момент панель Data должна выглядеть так, как показано на рис. 6.40.

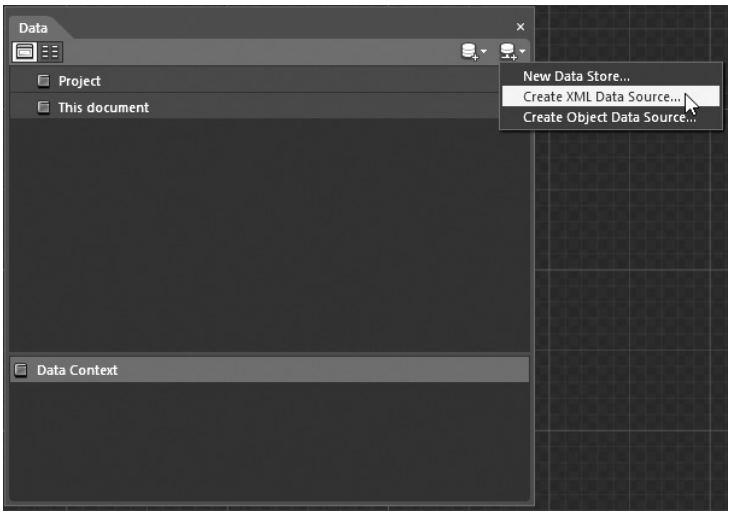


Рис. 6.38. Ввод нового источника данных XML

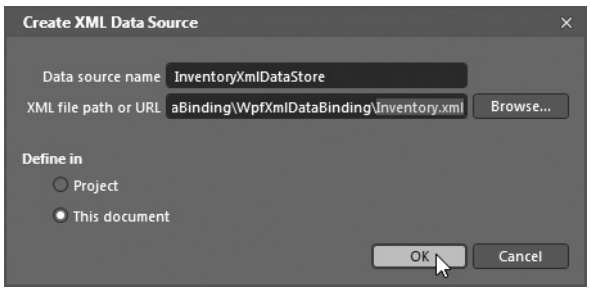


Рис. 6.39. Создание источника данных из выбранного XML-документа

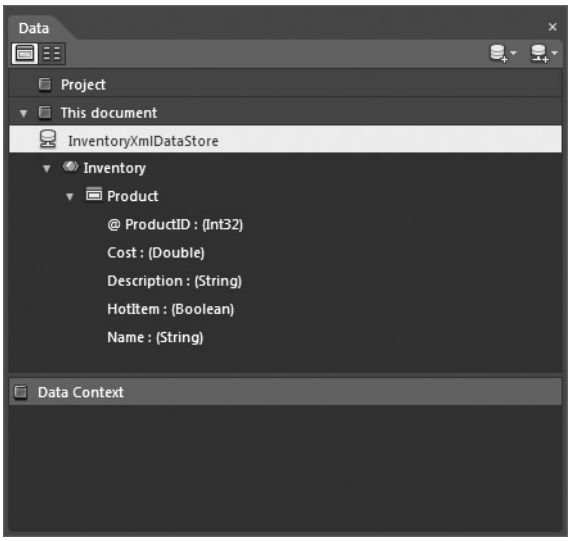


Рис. 6.40. Новый источник данных XML на панели Data

Если вы проанализируете данные XAML в файле разметки главного окна `MainWindow.xaml`, то обнаружите вновь определенный новый ресурс типа `XmlDataProvider`:

```
<Window.Resources>
  <XmlDataProvider x:Key="InventoryDataSource"
                   Source="\Inventory.xml"
                   d:IsDataSource="True"/>
</Window.Resources>
```

Привязка данных XML к элементам пользовательского интерфейса с помощью оператора `XPath`

Теперь, когда определен источник данных XML, можете перетащить узлы из панели Data на монтажный стол, подобно тому, как делали это ранее с источником данных объекта. Так, если перетащить весь узел `Product` на монтажный стол, в среде Expression Blend IDE автоматически сформируется элемент управления типа `ListBox`, в котором отображаются все данные каждого элемента разметки XML. Как показано в приведенном ниже фрагменте разметки, свойство `ItemsSource` привязано к узлу `<Products>` с помощью оператора `XPath`.

```
<ListBox HorizontalAlignment="Left"
       ItemTemplate="{DynamicResource ProductTemplate}"
       ItemsSource="{Binding XPath=/Inventory/Product}"
       Margin="89,65,0,77" Width="200"/>
```

Кроме того, в автоматически сформированном шаблоне, указанном в свойстве `ItemTemplate`, используются дополнительные операторы `XPath` для привязки различных атрибутов и узлов, подчиненных узлу `<Product>`, как показано в приведенном ниже фрагменте разметки.

```
<DataTemplate x:Key="ProductTemplate">
  <StackPanel>
    <TextBlock Text="{Binding XPath=@ProductID}"/>
    <TextBlock Text="{Binding XPath=Cost}"/>
    <TextBlock Text="{Binding XPath=Description}"/>
    <CheckBox IsChecked="{Binding XPath=HotItem}"/>
    <TextBlock Text="{Binding XPath=Name}"/>
  </StackPanel>
</DataTemplate>
```

Опять же, как и при использовании источника данных объекта, отдельные атрибуты (например, `ProductID`) или подчиненные узлы (`Cost`, `Description` и т.д.) можно перетащить непосредственно на монтажный стол, чтобы сформировать элементы управления типа `ListBox`, в которых должно отображаться только определенное подмножество данных. Так, если перетащить узел `HotItem` на монтажный стол, в среде Expression Blend IDE автоматически сформируется новый элемент управления типа `ListBox`, в котором будут отображаться новые привязанные к нему данные, как показано в приведенном ниже фрагменте разметки.

```
<!-- Списковое окно -->
<ListBox HorizontalAlignment="Right"
       ItemTemplate="{DynamicResource ProductTemplate1}"
       ItemsSource="{Binding XPath=/Inventory/Product}"
       Margin="0,78,54,64" Width="200"/>

<!-- Шаблон данных -->
  <DataTemplate x:Key="ProductTemplate1">
    <StackPanel>
```

```
      <CheckBox IsChecked="{Binding XPath=HotItem}"/>
    </StackPanel>
</DataTemplate>
```

Подобные шаблоны данных вам, скорее всего, придется видоизменить способами, описанными ранее в этой главе. Но нетрудно заметить, что общий принцип работы с источником данных XML очень похож на работу с источником данных объекта. Поэтому у вас в этом отношении не должно возникнуть особых затруднений.

Примечание. В руководстве пользователя Expression Blend приводится очень интересный пример, демонстрирующий применение источника данных XML для привязки к веб-каналу. Если вас интересует этот вопрос, обращайтесь за справкой к разделу "Пример создания программы для чтения новостей из веб-канала" (Try it: Create an RSS news reader).

Прежде чем переходить к завершающей теме этой главы (созданию новых хранилищ данных и выборочных данных), рассмотрим еще один пример, в котором демонстрируется второй способ применения панели Data. На этом примере будет показано, насколько быстро можно построить представление данных с перечислением подробностей, активизировав режим Details на панели Data.

Исходный код. Исходный код примера проекта `WpfXmlDataBinding` находится в папке `Ch 6 Code` загружаемого архива примеров проектов к данной книге.

Привязка данных к перечисляемым подробностям

В рассмотренных ранее примерах привязки данных было показано, каким образом элементы данных увязываются с новыми элементами управления типа `ListBox` на панели Data. Но справа от кнопки выбора режима отображения списком (List Mode) находится кнопка Details Mode (рис. 6.41), после щелчка на которой источник данных переводится в режим отображения подробностей.

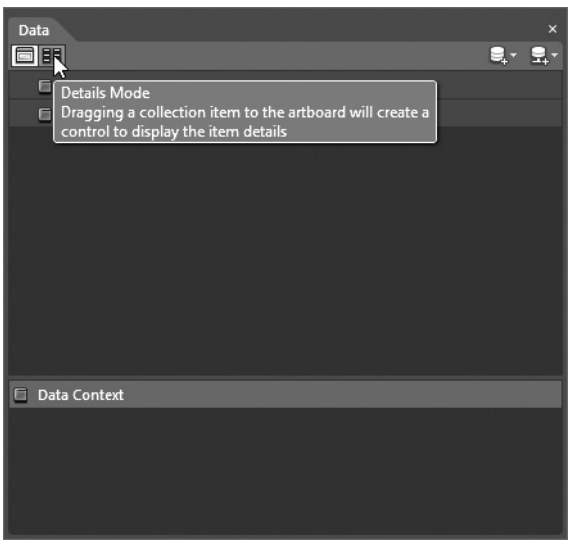


Рис. 6.41. На панели Data можно переходить из режима отображения списком в режим отображения подробностей и обратно

Если после выбора режима Details перетащить отдельные свойства из панели Data на монтажный стол, то соответствующие значения данных фиксируются в элементе управления типа `TextBlock`, а не `ListBox`. А если перетащить родительский узел на монтажный стол, то в среде Expression Blend IDE будет автоматически сформирована разметка, в которой описывается циклическое обращение к подробностям выбранного элемента (в рассматриваемом здесь примере будет показано, как это делается).

Итак, для изучения на практике данного режима работы панели Data создайте новый проект приложения WPF (или Silverlight), присвоив ему имя `ListDetailsDataBinding`. Введите в новый проект файл исходного кода C#, содержащий классы `PurchaseOrder` и `PurchaseOrders`, создайте этот проект и воспользуйтесь панелью Data, чтобы определить новый источник данных объекта, привязываемый к специальному классу `PurchaseOrders`.

Создание пользовательского интерфейса

Убедитесь в том, что вы по-прежнему находитесь в режиме отображения списком на панели Data, и перетащите узел `Description` на монтажный стол, чтобы сформировать элемент управления типа `ListBox`, в котором отображается текстовое описание каждого элемента списка. А для описания содержимого этого элемента управления добавьте также элемент управления типа `Label`. Щелкните на кнопке Details Mode в левом верхнем углу панели Data и выберите все три свойства (`Amount`, `Description` и `TotalCost`), нажав клавишу <Shift> и щелкнув сначала на первом свойстве, а затем на последнем (рис. 6.42).

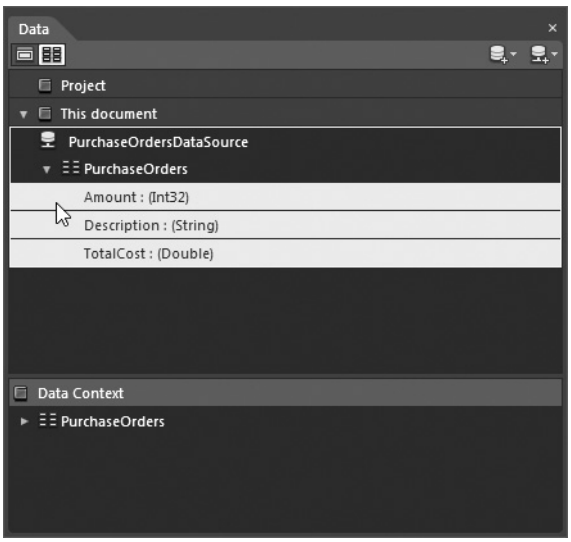


Рис. 6.42. Выбор нескольких элементов в режиме отображения подробностей на панели Data

Выбрав указанные выше элементы, перетащите их на монтажный стол. На данном этапе монтажный стол должен выглядеть так, как показано на рис. 6.43.

Из панели Data было перенесено несколько элементов, и поэтому в среде Expression Blend IDE был сформирован объект типа `Grid`, содержащий ряд связанных друг с другом элементов управления типа `TextBlock`. Если вы проанализируете теперь содержимое панели Objects and Timeline, то обнаружите на ней иерархическое представление элементов пользовательского интерфейса (рис. 6.44).

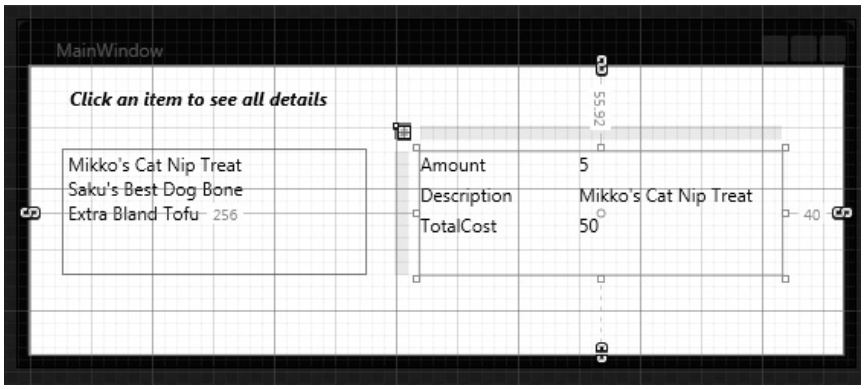


Рис. 6.43. Вид пользовательского интерфейса с перечислением подробностей

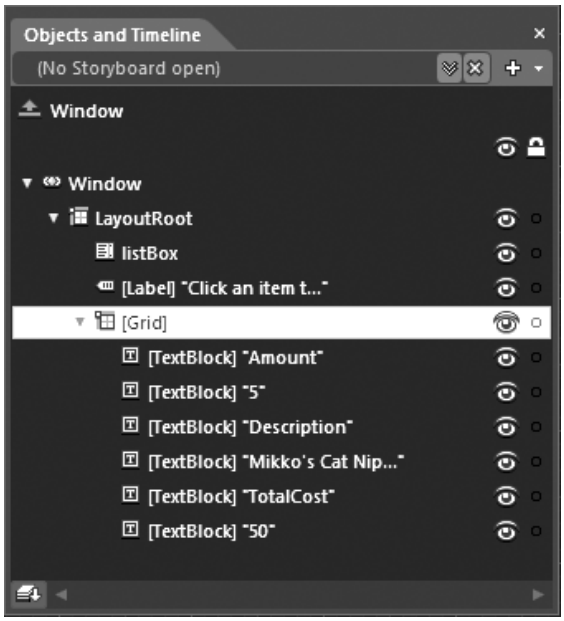


Рис. 6.44. В результате перетаскивания нескольких элементов из панели Data на монтажный стол образуется сетка, состоящая из элементов управления

А теперь запустите свое приложение на выполнение. Если вы щелкнете на любом элементе списка, то в области отображения подробностей (сетке из элементов управления типа `TextBlock`) появится описание выбранного элемента. Не так уж и плохо, не так ли? Конечно, вы вольны изменить компоновку элементов на подчиненной сетке, видоизменить соответствующие шаблоны данных и внести другие коррективы во внешний вид пользовательского интерфейса, но не будем на этом больше останавливаться и перейдем непосредственно к анализу разметки, описывающей привязку данных.

Анализ сформированной разметки

В результате перетаскивания исходного узла `Description` из панели Data на монтажный стол в среде Expression Blend IDE автоматически формируется основная разметка, в которой определяется контекст данных для корневого объекта типа `Grid`, увязываемого со специальной коллекцией. Кроме того, к элементу управления типа `ListBox` применяется простой шаблон данных для отображения значений свойств `Description`. Ниже приведена первоначальная разметка, сформированная *перед* добавлением дополнительных элементов в режиме Details.

```
<Window
... >
  <Window.Resources>
    <CollectionDataContext:PurchaseOrders
      x:Key="PurchaseOrdersDataSource" d:IsDataSource="True"/>
    <DataTemplate x:Key="PurchaseOrderTemplate">
      <StackPanel>
        <TextBlock Text="{Binding Description}"/>
      </StackPanel>
    </DataTemplate>
  </Window.Resources>

  <Grid x:Name="LayoutRoot" DataContext="{Binding Source=
    {StaticResource PurchaseOrdersDataSource}}">
    <ListBox HorizontalAlignment="Left"
           ItemTemplate="{DynamicResource
  </Grid>
</Window>
```

После добавления оставшихся элементов представления подробностей в подчиненной сетке будет определен собственный контекст данных, привязанный к свойству `SelectedItem` элемента управления типа `ListBox`. В каждом элементе управления типа `TextBlock`, находящемся на этой сетке, используется вполне предполагаемое расширение разметки `{Binding}` для связывания с соответствующим свойством объекта из специальной коллекции. Следует иметь в виду, что каждому элементу в Expression Blend автоматически присваивается имя, поскольку оно не было задано явным образом в свойстве `Name`. Ниже приведен фрагмент разметки, сформированной после добавления оставшихся элементов представления подробностей.

```
<Grid DataContext="{Binding SelectedItem, ElementName=listBox}"
     Margin="257,53.92,36,0"
     d:DataContext="{Binding [0]}" Height="97" VerticalAlignment="Top">
  <TextBlock HorizontalAlignment="Left" VerticalAlignment="Top"
           Width="100" Text="Amount"/>
  <TextBlock Text="{Binding Amount}" HorizontalAlignment="Left"
           VerticalAlignment="Top"
           Width="150" Margin="104,0,0,0"/>
...
</Grid>
```

Примечание. Второе свойство `DataContext`, обозначенное префиксом дескриптора `d`, используется на поверхности визуального конструктора Expression Blend и не требуется для нормального выполнения операции привязки данных.

На рис. 6.45 показан один из возможных вариантов окончательного оформления пользовательского интерфейса в рассматриваемом здесь примере проекта.

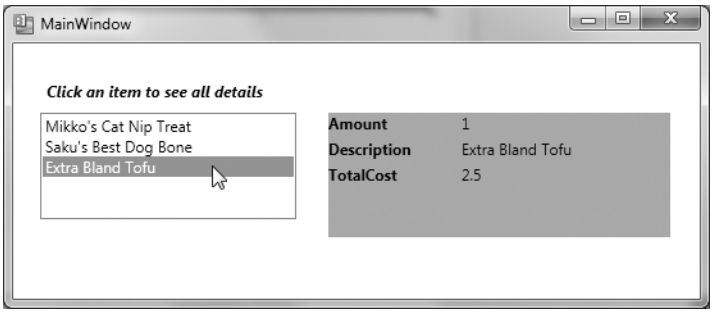


Рис. 6.45. Результат выполнения операции привязки данных к перечисляемым подробностям

Исходный код. Исходный код примера проекта `ListDetailsDataBinding` находится в папке `Ch 6 Code` загружаемого архива примеров проектов к данной книге.

Назначение выборочных данных

Во многих примерах, рассмотренных до сих пор в этой главе, использовалась специальная коллекция объектов типа `PurchaseOrder`, которая была создана вручную в редакторе исходного кода C#. Оказывается, что при построении пользовательского интерфейса разработчикам приложений довольно часто требуется оперировать некоторым количеством временных данных. Для этой цели *реальные* данные могут быть извлечены из реляционной базы данных, параллельно создаваемой коллегой-администратором базы данных, или же из отдельных бизнес-объектов, используемых в приложении и представляющих собой нечто более реальное, чем совокупность диаграмм классов, составленных на унифицированном языке моделирования (UML).

При построении приложения на платформе WPF или Silverlight в среде Expression Blend IDE имеется возможность ввести так называемые *выборочные данные*. Как подразумевает само название, выборочные данные позволяют визуализировать пользовательский интерфейс при отсутствии реальных данных. Выборочные данные могут быть вообще использованы при изучении способов и средств привязки данных в Expression Blend.

Пример. Выборочные данные оказываются полезными и при построении прототипов приложений средствами SketchFlow. Подробнее об этом речь пойдет в главе 8.

Ввод выборочных данных в проект

Создайте новый проект приложения WPF или Silverlight, присвоив ему имя `FunWithSampleData`[9]. Затем перейдите к панели Data и выберите из меню команду для ввода выборочных данных в новый проект, как показано на рис. 6.46. Оставьте без изменения все исходные установки параметров в открывшемся диалоговом окне.

В данный момент на панели Data доступен целый ряд тестовых элементов, в том числе и специальная коллекция, являющаяся расширением класса `ObservableCollection`, а также два устанавливаемых по умолчанию свойства — `Property1` и `Property2`. Дважды щелкните на текущем узле `Collection` и переименуйте его в `PersonCollection`

[9] В отличие от большинства проектов, рассматриваемых в этой книге, решение для данного проекта в загружаемом архиве не предоставляется, поскольку оно выходит за рамки обычного учебного примера.

(Личная коллекция). Аналогичным образом переименуйте оба упомянутых свойства в `FirstName` (Имя) и `LastName` (Фамилия) соответственно. На рис. 6.47 показана панель Data после внесенных корректив.

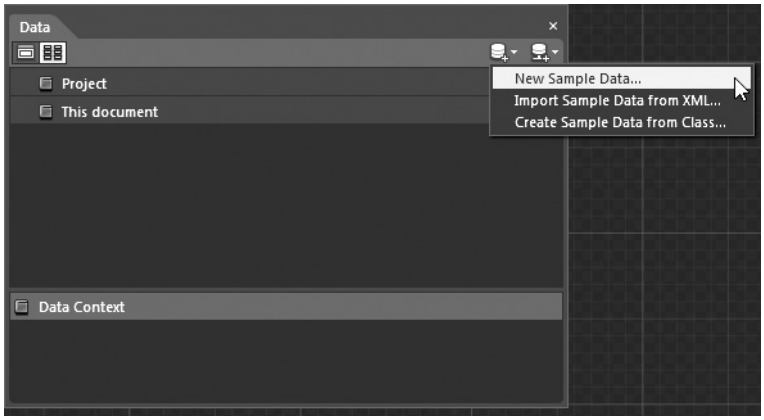


Рис. 6.46. Ввод выборочных данных в новый проект

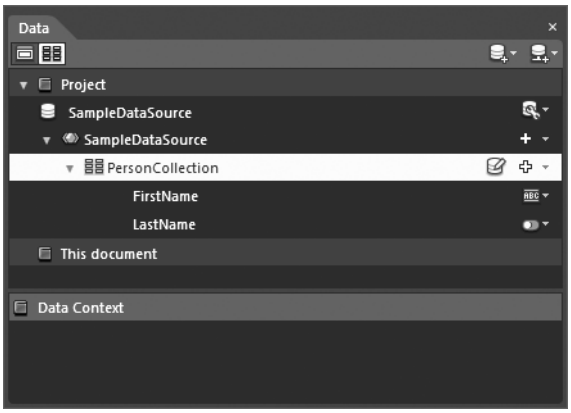


Рис. 6.47. Переименование исходных элементов

Добавление дополнительных свойств

В создаваемое здесь хранилище данных можете ввести дополнительные свойства, воспользовавшись кнопкой New Property (Новое свойство). Если же вы щелкнете на стрелке, направленной вниз, то обнаружите четыре варианта выбора данной операции (рис. 6.48).

Ниже дается краткое описание первых трех вариантов выбора, а четвертый рассматривается далее.

- Add Simple Property **(Добавить простое свойство).** Этот вариант выбирается для ввода свойства `String` (Строка), `Number` (Число) или `Boolean` (Логическое значение). По умолчанию вводится свойство `String`, но это положение можно изменить после создания хранилища данных.
- Add Complex Property **(Добавить сложное свойство).** Этот вариант выбирается для создания свойства, которое может содержать порожденные свойства (иными словами, новый класс со специальными свойствами).

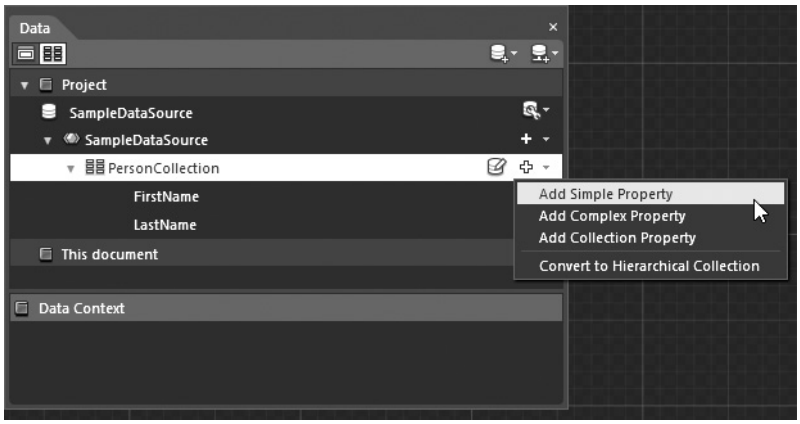


Рис. 6.48. Добавление дополнительных свойств

- Add Collection Property **(Добавить свойство коллекции).** Этот вариант выбирается для создания нового класса в качестве расширения класса `ObservableCollection`.

Добавьте новое простое свойство `Picture` (Фотография), которое по умолчанию становится свойством данных типа `String`.

Видоизменение типов данных и значений

Добавив дополнительные свойства, можете настроить типы их данных. Для этого прежде всего щелкните на раскрывающемся списке, выполняющем функции встроенного редактора свойств. В качестве примера на рис. 6.49 показано, каким образом тип данных свойства `Picture` изменяется на `Image`. Следует, однако, иметь в виду, что свойство `Picture` может быть настроено на выбор конкретного файла изображения, например фотографий разных людей. Впрочем, это в данном случае не так уж и важно, поскольку в Expression Blend по умолчанию используются некоторые образцы изображений.

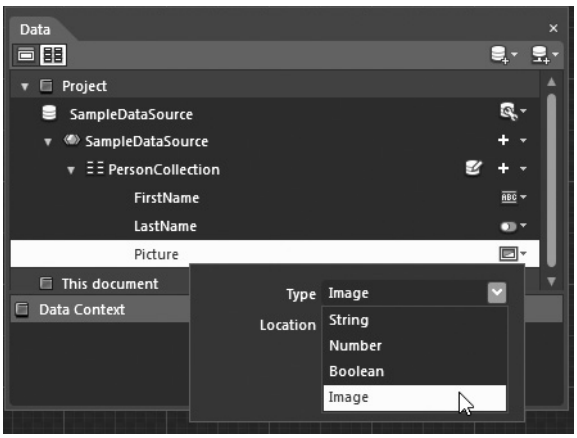


Рис. 6.49. Видоизменение типов данных отдельных свойств

Кроме того, можете настроить типы данных всех свойств, щелкнув на кнопке Edit sample values (Править выборочные значения), как показано на рис. 6.50.

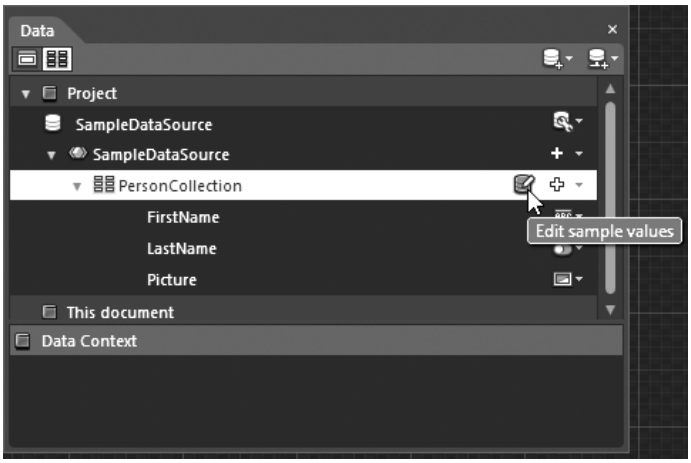


Рис. 6.50. Правка значений всех свойств

В открывшемся диалоговом окне Edit Sample Values можете, например, изменить логическое значение типа `Boolean` в свойстве `LastName` на строковое значение типа `String`, как показано на рис. 6.51. Как видите, каждое свойство настраивается на используемый по умолчанию набор строковых данных и файлов изображений (ведь это же *выборочные* данные). А в нижней части данного окна вы можете также задать количество тестовых записей, которые требуется сформировать.

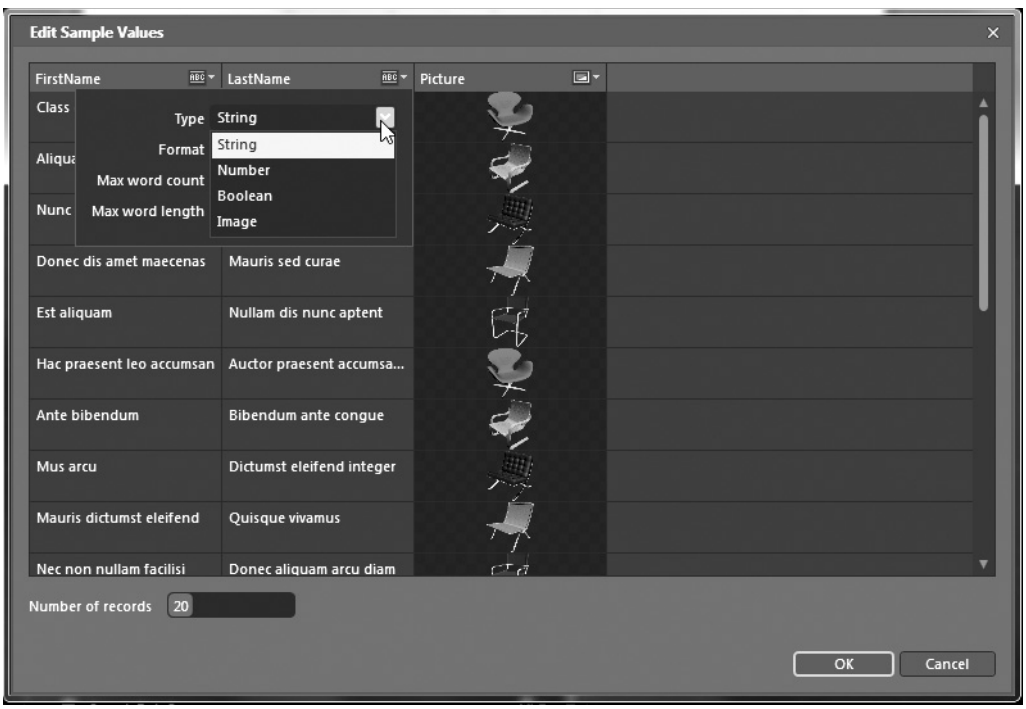


Рис. 6.51. Диалоговое окно Edit Sample Values

Привязка выборочных данных к пользовательскому интерфейсу

Надеюсь, что дойдя до этого раздела, вы уже знаете, что делать дальше. В частности, можете перетащить выбранные элементы из панели Data на монтажный стол. Не забудьте только выбрать режим List на панели Data, поскольку по умолчанию выборочные данные вводятся в режиме Details. На рис. 6.52 приведен конечный результат перетаскивания узла `PersonCollection` на элемент управления типа `DataGrid` в режиме List.

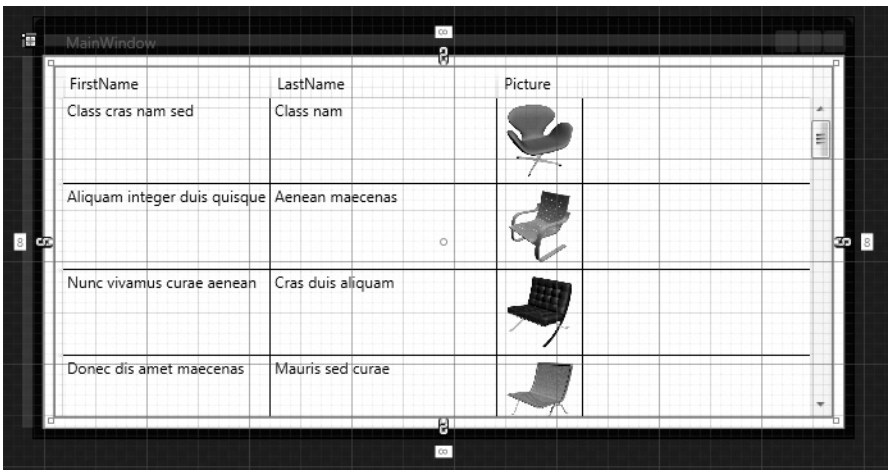


Рис. 6.52. Визуализация выборочных данных в табличном представлении элемента управления типа `DataGrid`

Следует также иметь в виду, что выборочные данные можно преобразовать из обычного списка объектов в формат иерархического представления, как показано на рис. 6.53.

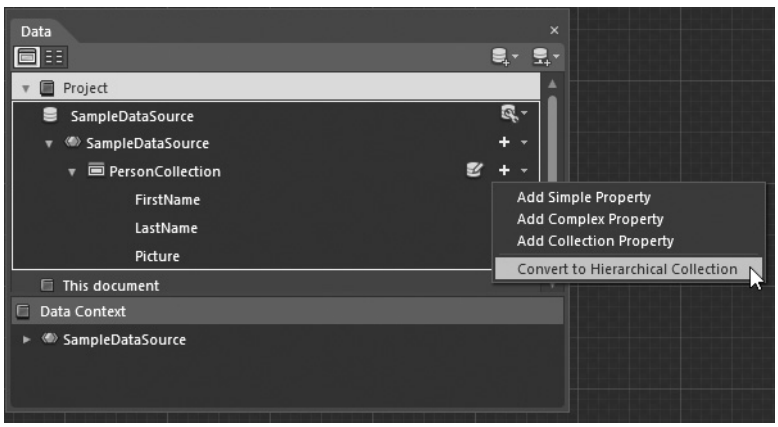


Рис. 6.53. Преобразование выборочных данных в формат иерархического представления

На рис. 6.54 показаны те же самые данные, привязанные к новому элементу управления типа `TreeView`, в формате иерархического представления.

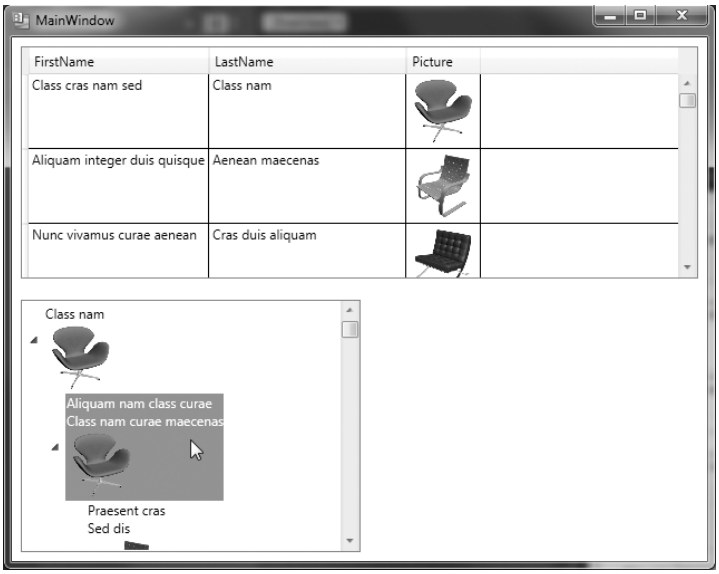


Рис. 6.54. Отображение выборочных данных в иерархическом представлении элемента управления типа `TreeView`

Дополнительный учебный материал по выборочным данным

Если вас заинтересуют вопросы применения выборочных данных, а это, скорее всего, произойдет на стадии создания прототипов новых приложений WPF или Silverlight[10], обратитесь за дополнительными сведениями к руководству пользователя Expression Blend. В частности, в разделе "Создание выборочных данных" (Creating sample data) этого руководства вы найдете немало полезной информации на данную тему (рис. 6.55).

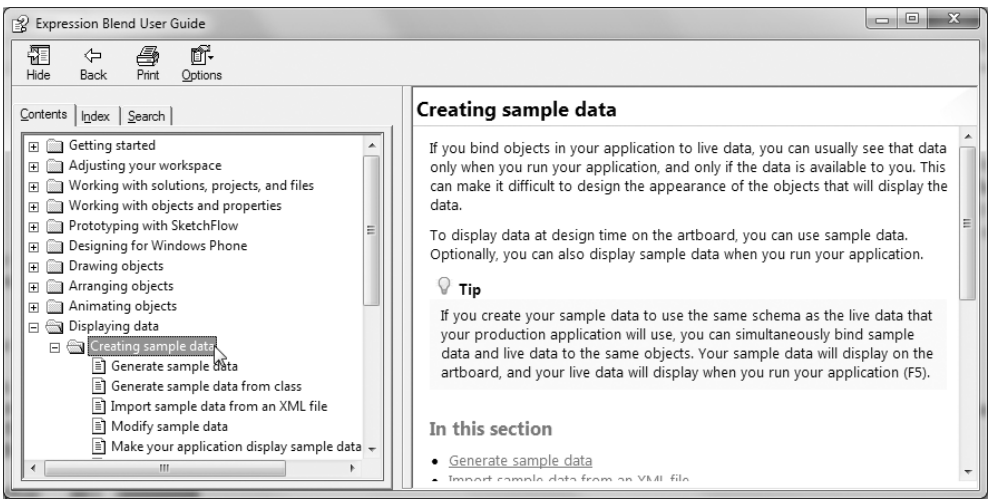


Рис. 6.55. В этом разделе руководства пользователя Expression Blend можно найти дополнительные сведения по выборочным данным

[10] Подробнее о создании прототипов приложений средствами SketchFlow речь пойдет в главе 8.

Заключительные краткие замечания на тему привязки данных

Из предыдущего материала этой главы вы усвоили целый ряд способов и средств привязки данных в Expression Blend. В частности, научились привязывать свойства одних элементов управления к свойствам других, специальные коллекции — к элементам пользовательского интерфейса, создавать шаблоны данных, формировать представления данных с перечислением подробностей и применять выборочные данные. Но этим тема привязки данных не ограничивается. Поэтому в заключение вкратце рассмотрим ряд других вопросов, связанных с привязкой данных, а более подробно можете изучить их самостоятельно.

Привязка данных из реляционной базы данных

Данные приложения чаще всего хранятся в реляционной базе данных. Как известно, на платформе .NET предоставляется довольно полный прикладной интерфейс ADO.NET для организации доступа к подобного рода данным. Используя эту модель программирования, вы сможете манипулировать таблицами реляционной базы данных самыми разными способами: на подключенном и отключенном уровнях, на уровне архитектуры Entity Framework и т.д.

Как правило, логика доступа к данным организована в специальные библиотеки .NET, а обращение к ней осуществляется из внешних приложений: главного окна типа `Window` на платформе WPF, специальной службы WCF и пр. Обращаясь к библиотеке функций доступа к данных из проекта приложения, разрабатываемого в Expression Blend, можете создать новый источник данных объекта, из которого будут вызываться конкретные методы выборки данных для соответствующих объектов. А получаемый в итоге набор данных привязывается далее к элементу пользовательского интерфейса с помощью панели Data.

Мы не будем рассматривать здесь процесс построения в Expression Blend пользовательского интерфейса, в котором применяются библиотеки функций доступа к данным, поскольку для этого пришлось быть создать базу данных и написать немало процедурного кода на C#. Но если вы желаете ознакомиться с вопросами применения ADO.NET для построения библиотеки данных и ее применения в проектах приложений, разрабатываемых в Expression Blend, обращайтесь за справкой к разделу “Пример отображения данных из базы данных SQL” (Try it: Display data from a sample SQL database) руководства пользователя Expression Blend (рис. 6.56). В этом разделе вы найдете учебный материал, в котором подробно излагается весь процесс.

Назначение шаблонов проектов с привязкой данных (шаблон проектирования MVVM)

Следует также вкратце упомянуть, что если в Expression Blend установлены необходимые шаблоны, как поясняется в главе 7, то в проектах, разрабатываемых на платформе WPF, Silverlight или Windows Phone 7, может быть использован весьма распространенный шаблон проектирования Model-View-ViewModel (MVVM — Модель–Представление–Модель представления). Этот шаблон проектирования помогает разрабатывать приложения, в которых уровни пользовательского интерфейса и бизнес-логики связаны как можно более слабо. Если выбрать шаблон проекта с привязкой данных (`Databound`), то первоначально его исходный код будет сформирован по шаблону проектирования MVVM. В качестве примера на рис. 6.57 показано, каким образом шаблон проекта `Silverlight Databound Application` выбирается в диалоговом окне New Project среды Expression Blend IDE.

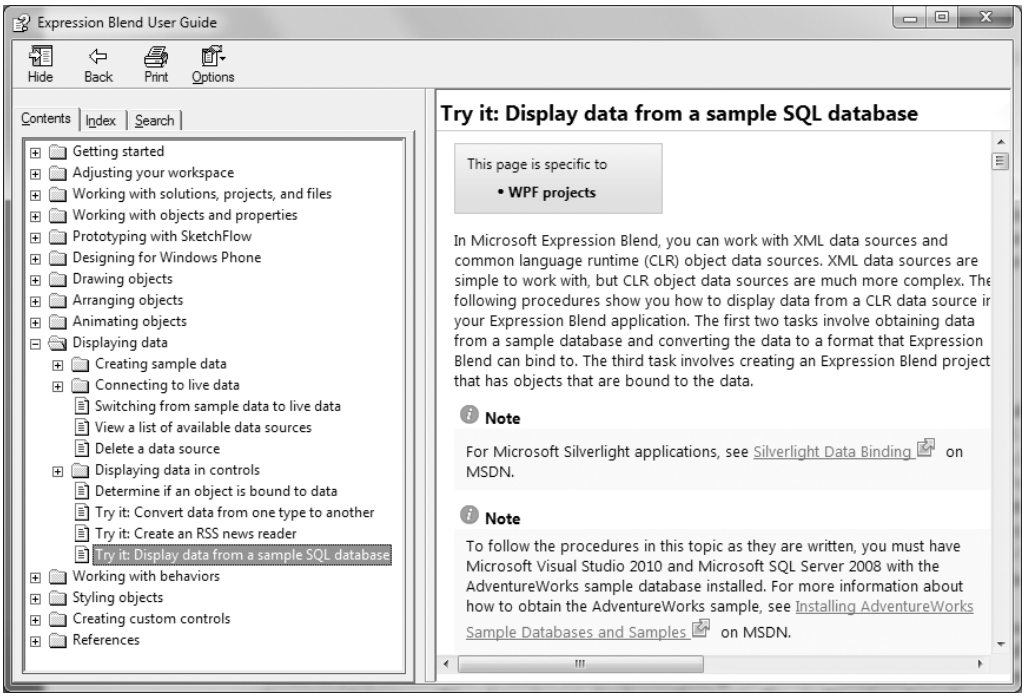


Рис. 6.56. В этом разделе можно найти подробные сведения об организации доступа к базе данных

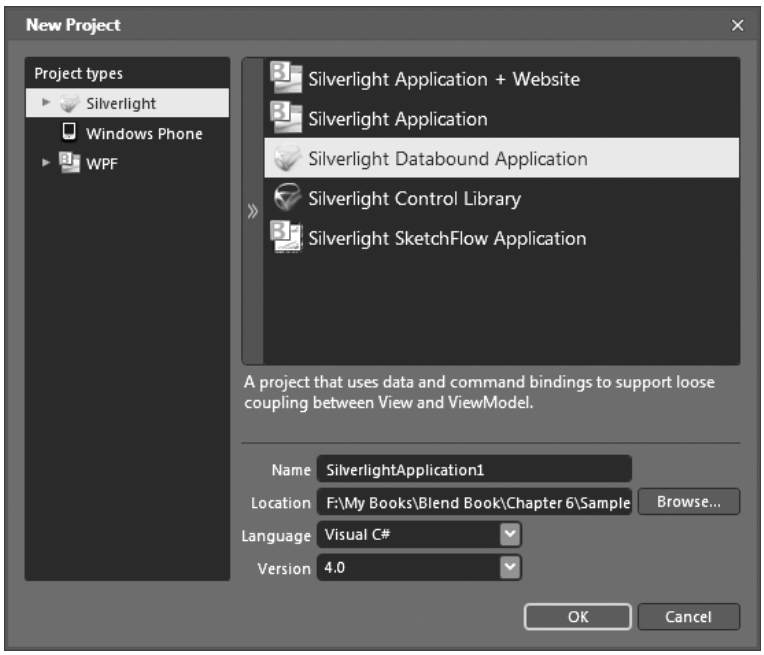


Рис. 6.57. Разработка проектов с привязкой данных в Expression Blend выполняется по шаблону проектирования MVVM

Безусловно, освоение шаблона проектирования MVVM — дело полезное для всякого разработчика приложений на платформе WPF или Silverlight, но, к сожалению, этот вопрос выходит за рамки рассмотрения данной книги. Как известно, обращение к шаблону проектирования MVVM может повлечь за собой необходимость применить целый ряд более передовых методик программирования. Если же вы хотите ознакомиться поближе с шаблоном проектирования MVVM, рекомендуется, прежде всего, создать в Expression Blend новый проект типа `Databound` и внимательно проанализировать первоначально сформированный исходный код этого проекта.

Рекомендуется также прочитать очень полезную статью "Приложения WPF с шаблоном проектирования "Модель–Представление–Модель представления" в *MSDN Magazine* (аналогичные приемы подходят и для разработки приложений на платформе Silverlight)[11]. В этой статье подробно анализируются причины широкой распространенности шаблона проектирования MVVM и разъясняется его общая архитектура. Однако эта статья рассчитана на опытных разработчиков приложений на платформе .NET.

Резюме

В этой главе был рассмотрен целый ряд способов и средств Expression Blend, упрощающих связывание отдельных значений из источника данных с элементами пользовательского интерфейса. В начале этой главы было рассмотрено назначение привязки данных одних элементов управления к другим и операций одно- и двухсторонней привязки данных, а также показано, как пользоваться диалоговым окном Create Data Binding, создавать и регистрировать специальные классы преобразования данных.

Затем речь шла об особенностях применения панели Data для создания источников данных объектов на примерах составления специальной коллекции объектов, привязываемых к различным элементам пользовательского интерфейса, и настройках операций привязки данных по специальным шаблонам данных для отображения в пользовательском интерфейсе. Эти примеры наглядно показывают, что специальные шаблоны данных могут быть зачастую созданы теми же самыми способами, которые применяются при построении специальных шаблонов элементов управления.

И в заключение были вкратце рассмотрены средства Expression Blend, позволяющие применять выборочные данные в разрабатываемых приложениях. С помощью этих средств можно определить и использовать имитационные данные, заменяющие реальные данные, которые могут оказаться недоступными при построении пользовательского интерфейса приложения. А в самом конце главы были в самых общих чертах обозначены такие важные, но выходящие за рамки данной книги вопросы, как разработка приложений по широко распространенному шаблону проектирования MVVM.

[11] См. `http://msdn.microsoft.com/ru-ru/magazine/dd419663.aspx`.

ГЛАВА 7

Разработка приложений на платформе Windows Phone 7

На момент написания этой книги Windows Phone 7 считалась самой последней версией операционной системы для мобильных устройств, разработанной корпорацией Microsoft в качестве непосредственного конкурента платформы для мобильных устройств iPhone, выпускаемых компанией Apple и другими производителями. С точки зрения разработчика приложений на платформе .NET создание аналогичных приложений на платформе Windows Phone 7 — весьма заманчивое предложение, поскольку ее собственный набор инструментальных средств, по существу, опирается на платформу Silverlight![1] А это означает, что приложение на платформе Windows Phone 7 можно разрабатывать средствами C#, VB, XAML, Visual Studio 2010, Expression Blend, используя ту же самую модель программирования, что и при создании приложений на платформе WPF или Silverlight.

В начале этой главы будет показано, каким образом устанавливается набор инструментальных средств Windows Phone 7 SDK, необходимый для разработки приложений на данной платформе и дополняющий Visual Studio 2010 и Expression Blend 4 необходимыми библиотеками, шаблонами и расширениями интегрированной среды разработки. Как только на вашем компьютере будут правильно установлены все необходимые для разработки инструментальные средства, вы ознакомитесь далее в этой главе с особенностями создания новых типов проектов и новыми компонентами Expression Blend, с помощью которых разрабатываются приложения на платформе Windows Phone 7.

Принимая во внимание то обстоятельство, что подавляющее большинство вопросов разработки приложений, рассматривавшихся в главах 1–6, имеют непосредственное отношение к платформе Windows Phone 7, в остальной части этой главы будут рассмотрены примеры создания и видоизменения основных типов проектов Windows Phone 7 (простого, панорамного и сводного), наглядно демонстрирующие различные аспекты разработки приложений на платформе Windows Phone 7.

Примечание. Несмотря на то что на платформе Windows Phone 7, по существу, применяется та же самая модель программирования, что и на платформе Silverlight, вам придется освоить целый ряд специальных приемов программирования мобильных устройств, в том числе средства навигации, взаимодействия с мобильным устройством и т.д. Но, принимая во внимание основное

[1] Официально эта платформа называется Silverlight for Windows Phone.

назначение этой книги — научить вас пользоваться Expression Blend для разработки пользовательских интерфейсов, а не реализовывать законченные программные решения, рассмотрение способов программирования в этой главе сведено к минимуму. Тем не менее в конце главы делаются ссылки на ряд полезных ресурсов в Интернете для дальнейшего самостоятельного и углубленного изучения вопросов разработки приложений на платформе Windows Phone 7.

Установка Windows Phone 7 SDK

Платформа для мобильных устройств была выпущена корпорацией Microsoft после выхода в свет интегрированных сред разработки Expression Blend 4 и Visual Studio 2010. Поэтому, если хотите разрабатывать проекты на этой платформе, вам придется загрузить и установить свободно доступный комплект инструментальных средств разработки (SDK), официально называемый Windows Phone 7 Developer Tools. Несмотря на то что этот комплект может быть загружен из самых разных мест в Интернете, сделать это проще всего, посетив начальную страницу веб-сайта, посвященного Expression Blend. С этой целью откройте избранный браузер и перейдите по следующему адресу:

`http://www.microsoft.com/expression/windowsphone`

На рис. 7.1 приведена страница, доступная по указанному выше адресу на момент написания этой книги, хотя в будущем он может измениться. Но в любом случае найдите ссылку `Download the Developer Tools` (Загрузить инструментальные средства разработчика) и щелкните на ней.

Рис. 7.1. Windows Phone 7 SDK можно свободно загрузить с веб-сайта, посвященного Expression Blend

Далее вам будет предложено загрузить небольшую исполняемую программу, начинающую процесс установки, как показано на рис. 7.2. Поэтому щелкните на кнопке Run (Выполнить), чтобы продолжить данный процесс.

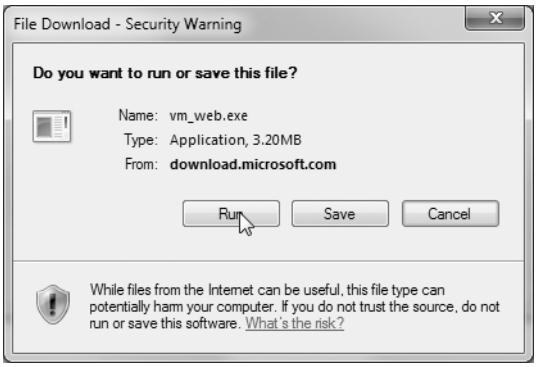


Рис. 7.2. Выполнение внутренней программы установки

Примечание. Если получите сообщение об ошибке, уведомляющее о том, что программа установки не может быть выполнена в режиме совместимости, загрузите ее еще раз, но на этот раз сохраните на рабочем столе операционной системы своего компьютера. Затем щелкните правой кнопкой мыши на ярлыке этой исполняемой программы и выберите команду Run as Administrator (Запуск от имени администратора) из всплывающего контекстного меню.

После загрузки программы установки еще раз щелкните на кнопке Run в открывшемся диалоговом окне, чтобы фактически начать процесс установки. Приняв условия лицензионного соглашения, щелкните на кнопке Install Now (Установить сейчас), как показано на рис. 7.3, чтобы дать согласие на установку всего набора инструментальных средств разработки на платформе Windows Phone 7.

Рис. 7.3. Установка Windows Phone 7 SDK

Весь процесс установки может занять некоторое время (где-то около получаса, что зависит от скорости соединения с Интернетом), поэтому сделайте перерыв на чашку

чая с бутербродом, а возможно, и на что-нибудь более существенное. По завершении установки вам, скорее всего, будет предложено перезагрузить свой компьютер. Сделайте это непременно.

Примечание. После установки Windows Phone 7 SDK рекомендуется сделать обычное обновление Windows на своем компьютере. Это даст возможность загрузить ряд сервисных пакетов.

Исследование нового комплекта инструментальных средств разработки

По завершении процесса установки вы получите в свое распоряжение целый набор инструментальных средств и шаблонов для разработки. Но конкретный его состав зависит от текущей конфигурации вашего компьютера. Программа установки Windows Phone 7 SDK обладает достаточно развитой логикой, чтобы установить только те компоненты, которые в настоящий момент отсутствуют на вашем компьютере. Так, если вы работаете на совершенно новом компьютере, где нет и в помине Visual Studio, Expression Blend или платформы .NET или Silverlight, после установки в вашем распоряжении окажутся следующие компоненты.

- Упрощенная версия Visual Studio 2010 Express.
- Версия Microsoft .NET Framework 4.0.
- Последняя версия Silverlight.
- Эмулятор Windows Phone 7.
- Свободно доступная версия Expression Blend для разработки приложений на платформе Windows Phone 7[2].
- Версия Microsoft XNA Game Studio 4.0.

Если же на вашем компьютере установлена среда Visual Studio 2010, программа установки обновит ее новыми шаблонами проектов на платформе Windows Phone 7, вместо того чтобы устанавливать упрощенную версию Visual Studio 2010 Express. А если на вашем компьютере уже установлена среда Expression Blend, она будет также обновлена новыми шаблонами проектов на платформе Windows Phone 7 вместо установки свободно доступной версии для разработки приложений только на данной конкретной платформе.

Кроме того, эмулятор Windows Phone 7 устанавливается таким образом, чтобы его можно было активизировать в Windows с помощью кнопки Пуск, хотя, как правило, это удобнее делать непосредственно. В процессе создания проекта на платформе Windows Phone 7 непосредственно в среде Expression Blend или Visual Studio 2010 этот эмулятор запускается автоматически для размещения разрабатываемого приложения.

Назначение Microsoft XNA Game Studio 4.0

Если вы щелкнете на кнопке Пуск в Windows, то обнаружите новый компонент Microsoft XNA Game Studio 4.0, который, по существу, является средой программирования, благодаря которой разработку игр для мобильных устройств Windows Phone, консоли Xbox 360 и компьютеров, работающих под Windows, можно выполнять в

[2] Корпорация Microsoft великодушно предоставляет специально созданную и свободно доступную версию Expression Blend, предназначенную только для разработки проектов на платформе Windows Phone 7 и соответственно называемую Expression Blend for Windows Phone. Следует, однако, иметь в виду, что в этой версии нельзя разрабатывать приложения на платформе WPF или Silverlight.

интегрированной среде Visual Studio, а не в Expression Blend! В состав XNA Game Studio входит набор библиотек XNA Framework, а по существу, сборок .NET, применяемых в разработке игр на платформе Microsoft .NET Framework, а также целый ряд шаблонов проектов, предназначенных для разработки в среде Visual Studio 2010.

Следует, однако, иметь в виду, что видеоигры на платформе Windows Phone 7 (а также Silverlight и WPF) можно разрабатывать и без компонента XNA Game Studio. Для этого, в частности, достаточно функциональных возможностей, доступных в прикладных интерфейсах .NET 4.0/Silverlight API, а также инструментальных средств и способов, с которыми вы ознакомились в предыдущих главах. Тем не менее в состав XNA Game Studio входят дополнительные и более эффективные библиотеки, предназначенные для создания видеоигр с более развитой логикой.

В этой книге компонент XNA Game Studio не рассматривается, поэтому заинтересовавшиеся им отсылаем на очень полезный в информативном плане веб-сайт, адрес которого приводится ниже. На этом веб-сайте можно найти немало учебного материала по разработке игр, в том числе статьи по алгоритмам искусственного интеллекта, внедрению физических свойств в игры (обнаружение столкновений, установка углов съемки камерой слежения и т.д.) и многим другим темам.

```
http://create.msdn.com/en-us/education/roadmap
```

И еще одно, последнее замечание по поводу компонента XNA Game Studio: после установки Windows Phone 7 SDK вы получите в свое распоряжение новый набор шаблонов проектов, предназначенных для разработки в среде Visual Studio 2010. Как показано на рис. 7.4, эти шаблоны проектов можно применять при создании приложений на самых разных платформах: Xbox 360, Windows Phone 7 или собственно в ОС Windows.

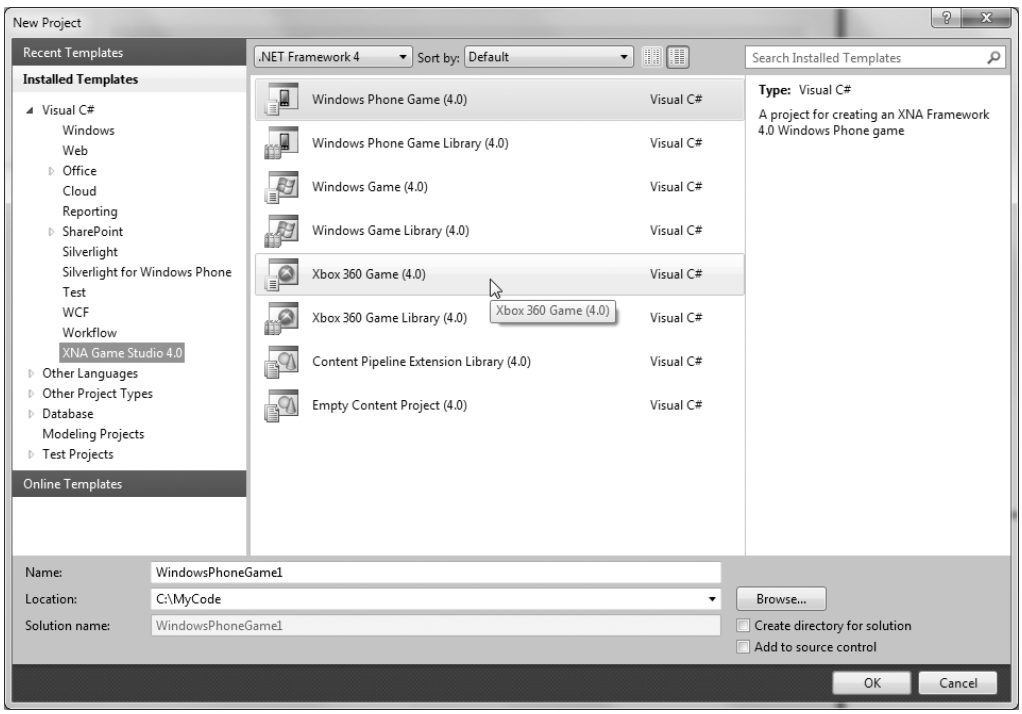


Рис. 7.4. В состав XNA Game Studio входит целый ряд новых шаблонов проектов, предназначенных для разработки в среде Visual Studio 2010

Установка документации на Windows Phone 7 SDK

Если вы намерены углубленно изучить модель программирования на платформе Windows Phone 7, рекомендуется обновить локальную справочную систему .NET Framework 4.0, чтобы выбирать в ней документацию на Windows Phone 7 SDK. Если на вашем компьютере уже установлена среда Visual Studio 2010[3], выберите сначала команду Пуск⇨Все программы⇨Microsoft Visual Studio 2010⇨Visual Studio Tools в Windows, а затем инструмент Manage Help Settings (Управление настройками справки). После этого можете выбрать установку документации в локальной справочной системе, как показано на рис. 7.5[4].

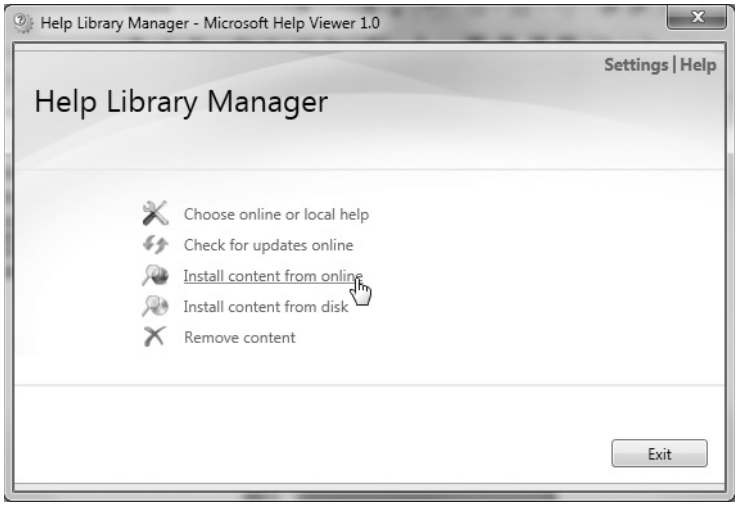


Рис. 7.5. Подготовка к установке документации на Windows Phone 7 SDK

И последнее, но не менее важное замечание: теперь можно установить любую часть справочной системы .NET, в том числе справочную документацию на Windows Phone 7 и Silverlight (рис. 7.6).

Примечание. Инструмент Manage Help Settings рекомендуется запускать на выполнение через каждые несколько недель, чтобы обнаруживать с его помощью полезные обновления документации и загружать самые последние сведения.

После установки документации на Windows Phone 7 SDK можете активизировать локальную справочную систему .NET Framework 4.0 и осуществить в ней поиск по теме “Silverlight for Windows Phone”, как показано на рис. 7.7.

Краткий обзор основных пространств имен Windows Phone 7 SDK

В этой главе не рассматриваются программные вопросы разработки приложений на платформе Windows Phone 7. Тем не менее нужно упомянуть хотя бы вкратце о новых прикладных интерфейсах, которыми вам, возможно, придется пользоваться в дальнейшем. Напомним, что при разработке приложения на платформе Windows Phone 7 на

---

[3] Если же вы не хотите устанавливать эту документу локально, можете обращаться к ней глобально через Интернет по адресу `http://msdn.microsoft.com`.

[4] Если вы не установили документацию на платформу .NET Framework 4.0 локально, воспользуйтесь для этой цели инструментом Manage Help Settings.

самом деле создается прикладная программа на платформе Silverlight. Поэтому если у вас имеется некоторый опыт прикладного программирования на платформе Silverlight или WPF, вы почувствуете себя в среде Windows Phone 7 как дома.

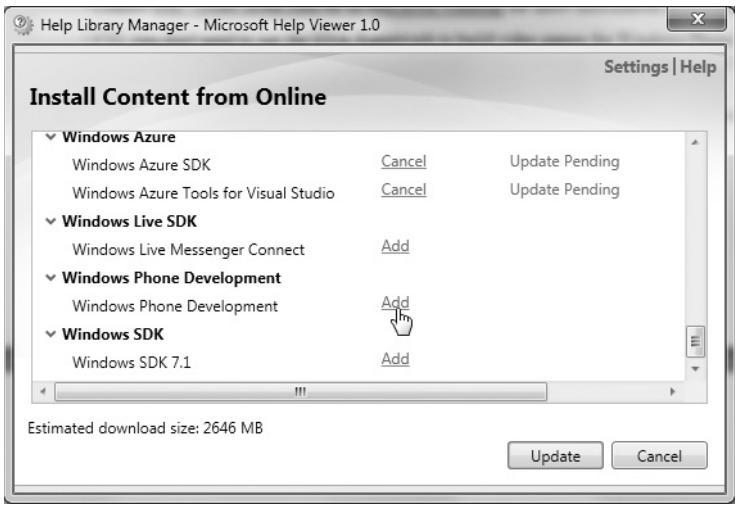


Рис. 7.6. Установка документации на Windows Phone 7 SDK

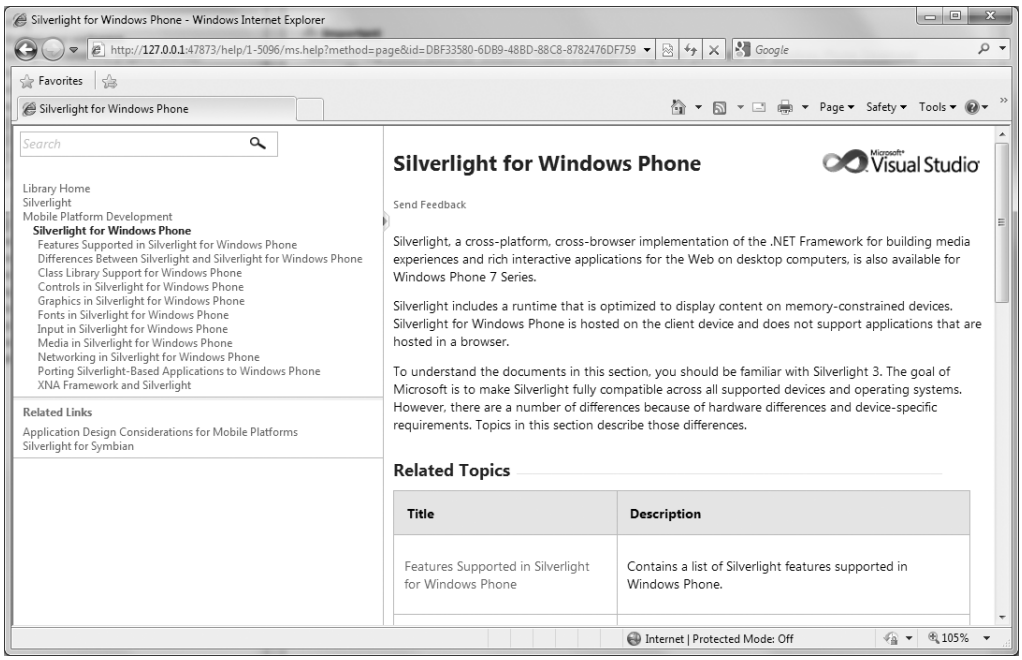


Рис. 7.7. Портал Silverlight for Windows Phone

Итак, в состав Windows Phone 7 SDK входит несколько новых сборок .NET, в том числе `Microsoft.Phone.dll` и `Microsoft.Phone.Interop.dll`, в которых определен ряд новых пространств имен .NET. В табл. 7.1 дается краткое описание некоторых, хотя и не всех

пространств имен для разработки приложений на платформе Windows Phone 7. Подобнее ознакомиться с ними вы можете, обратившись за справкой к документации на .NET Framework 4.0.[5]

Таблица 7.1. Основные пространства имен Windows Phone 7 SDK

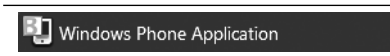
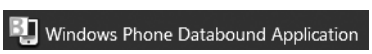
Пространство имен	Назначение
`Microsoft.Devices`	Определяет небольшое количество типов данных для непосредственного программирования мобильных устройств на платформе Windows Phone 7. К их числу относятся типы данных для программирования таких функций мобильных устройств, как, например, вибрирование, сбор информации о версии платформы, установленной в отдельном устройстве, и пр.
`Microsoft.Devices.Radio`	Предоставляет программный доступ к аппаратным средствам коротковолнового радиоприемника, встроенного в мобильное устройство
`MicrosoftDevices.Sensors`	Предоставляет доступ к функциям прикладного интерфейса, управляющим акселерометром, встроенным в мобильное устройство
`Microsoft.Phone.Controls`	Определяет ряд элементов управления мобильным телефоном, в том числе и тех, которые предназначены для построения панорамных и сводных видов пользовательских интерфейсов. Подробнее о подобных навигационных моделях речь пойдет далее в главе
`Microsoft.Phone.Notification`	Позволяет организовать прием данных от службы извещающих уведомлений Microsoft Push Notification Service в приложении на платформе Windows Phone 7
`Microsoft.Phone.Tasks`	Определяет ряд типов данных для взаимодействия с главными службами мобильного устройства, в том числе телефонной, электронной почты и встроенной фотокамеры

Просмотр новых шаблонов проектов в Expression Blend

Запустите Expression Blend на выполнение и выберите команду File⇨New Project из главного меню. В открывшемся диалоговом окне New Project вы должны обнаружить новый узел `Windows Phone`, в котором определен целый ряд новых шаблонов проектов, как показано на рис. 7.8.

Новые шаблоны проектов Windows Phone 7, доступные в Expression Blend, вкратце описываются в табл. 7.2. Как видите, они аналогичны, но не тождественны своим аналогам на платформах WPF и Silverlight.

Таблица 7.2. Шаблоны проектов Windows Phone 7, доступные в Expression Blend

Шаблон проекта	Назначение
	Этот шаблон простого проекта предназначен для разработки приложений на платформе Silverlight или Windows Phone 7
	Также предназначен для разработки приложений на платформе Windows Phone 7, но в нем применяются объекты типа `View` и `ViewModel` для слабого связывания логики представления и данных (см. главу 6)

[5] В конце главы будут приведены ссылки на некоторые ресурсы в Интернете, откуда можно загрузить примеры проектов на платформе Windows Phone 7, в которых применяются эти новые сборки и пространства имен.

Окончание табл. 7.2

Шаблон проекта	Назначение
Windows Phone Panorama Application	В этом шаблоне проекта применяется элемент управления типа `Panorama` для создания приложений панорамного типа (подробнее об этом речь пойдет далее)
Windows Phone Pivot Application	В этом шаблоне проекта применяется элемент управления типа `Pivot` для создания приложений сводного, снабженного вкладками типа (подробнее об этом речь пойдет далее)
Windows Phone Control Library	Предназначен для создания специальных элементов управления, неоднократно используемых в разных приложениях, разрабатываемых на платформе Windows Phone 7

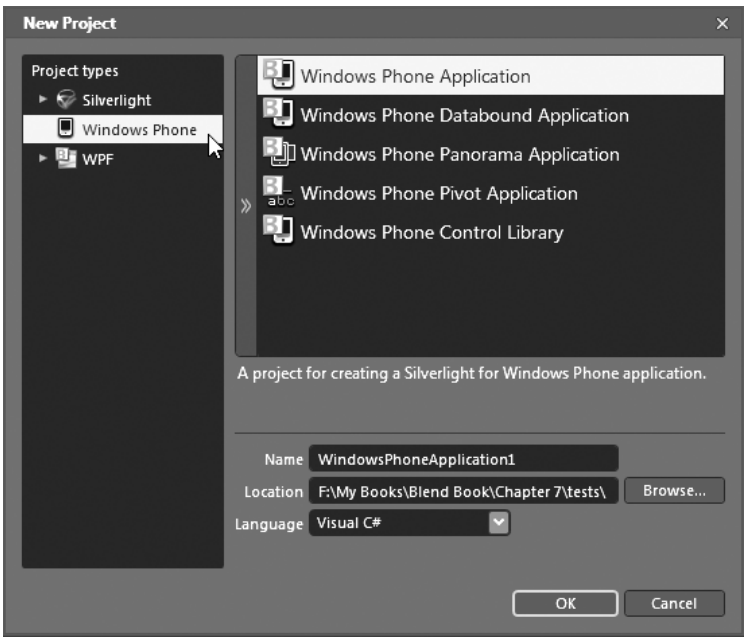


Рис. 7.8. Новые шаблоны проектов Windows Phone 7, доступные в Expression Blend

Перечисленные выше типы шаблонов проектов будут более подробно рассмотрены далее в главе. Но по сути в приложениях панорамного типа имеется возможность конструировать прокручиваемые по горизонтали длинные виртуальные полотна с элементами пользовательского интерфейса, которые плавно панорамируются по экрану мобильного устройства, тогда как в приложениях сводного типа пользователь может перелистывать связанные вместе страницы пользовательского интерфейса, подобно страницам книги, хотя и зачастую с применением анимации.

Обновление руководства пользователя Expression Blend

Следует также иметь в виду, что после установки Windows Phone 7 SDK локальная версия руководства пользователя Expression Blend регулярно обновляется новым учебным материалом и пошаговыми разборами на соответствующие темы, как показано на рис. 7.9.

Как ни странно, новая часть руководства пользователя Expression Blend, посвященная разработке приложений на платформе Windows Phone 7, не очень велика по объему, что само по себе неплохо! Как упоминалось ранее в этой главе, большую часть

тех способов и средств Expression Blend, с которыми вы уже ознакомились в предыдущих главах, можно с успехом применять при разработке приложений на платформе Windows Phone 7. Например, в процессе разработки приложения Windows Phone 7 вам по-прежнему придется пользоваться средствами Expression Blend в следующих целях.

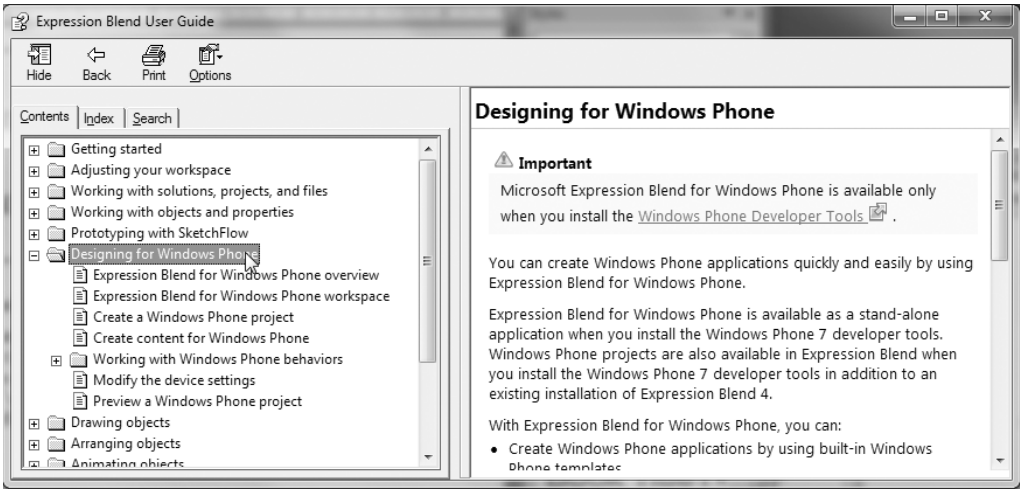


Рис. 7.9. Обновленное руководство пользователя Expression Blend

- Создание графики и организация взаимодействия с ней (см. главу 2).
- Оформление и применение ресурсов объектов (см. главу 2).
- Создание анимации и применение эффектов инерционности движения (см. главу 3).
- Построение пользовательских интерфейсов из элементов управления, диспетчеров компоновки и объектов поведения (см. главу 4).
- Создание стилей, шаблонов и специальных классов `UserControl` (см. главу 5).
- Применение панели Data для привязки данных и создания специальных шаблонов данных (см. главу 6).

Напомним, что одно из самых главных преимуществ разработки прикладных программ на платформе Windows Phone 7 заключается в том, что базовый прикладной интерфейс API технологически опирается на существующую платформу Silverlight, а та, в свою очередь, на платформу WPF. В действительности в документации на Windows Phone 7 SDK рассматриваются вопросы переноса уже имеющихся приложений с платформы Silverlight на платформу Windows Phone 7. В частности, об этом речь идет в разделе “Перенос приложений с платформы Silverlight на платформу Windows Phone 7” (Porting Silverlight-Based Applications to Windows Phone)[6].

Просмотр новых шаблонов проектов в Visual Studio 2010

Несмотря на то что в примерах, приведенных в этой главе, интегрированная среда разработки Visual Studio 2010 не применяется, следует все же заметить, что если

[6] Обратитесь также за справкой к разделу документации “Отличия между платформами Silverlight и Windows Phone 7” (Differences Between Silverlight and Silverlight for Windows Phone). В нем вы найдете немало существенных отличий, о которых полезно знать и которые свидетельствуют о том, что обе платформы не являются совершенно одинаковыми.

выбрать из главного меню этой среды команду File⇨New Project, то в иерархическом представлении языка программирования, выбранного в открывшемся диалоговом окне, появится новый узел `Silverlight for Windows Phone 7` (рис. 7.10).

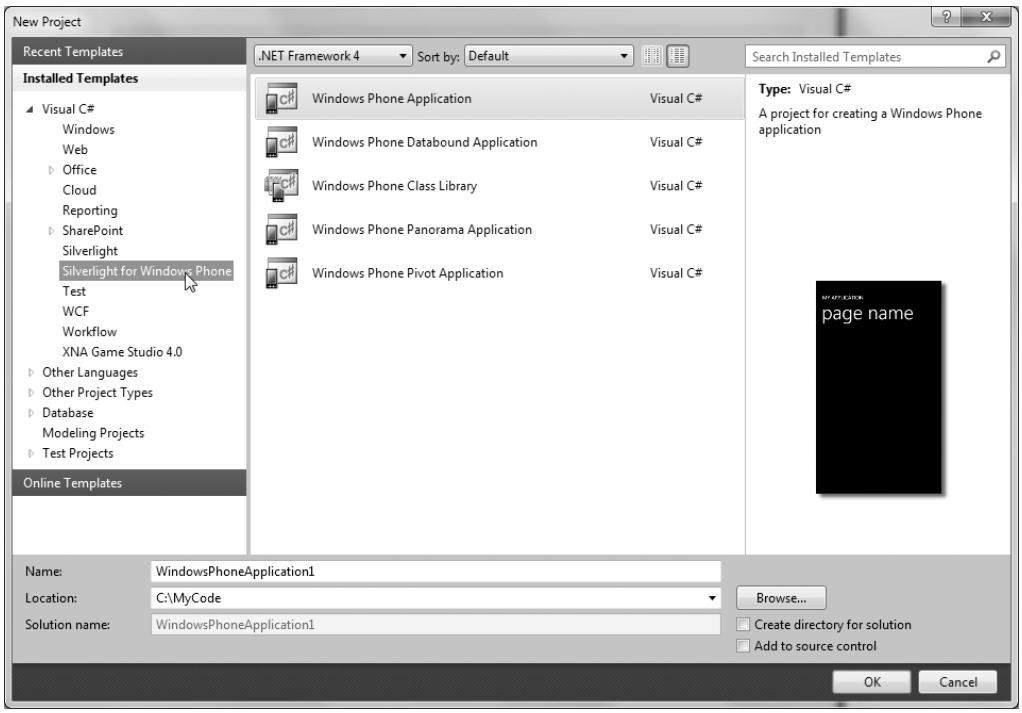


Рис. 7.10. Шаблоны проектов Windows Phone 7, доступные в Visual Studio 2010

Как видите, эти шаблоны проектов, разрабатываемых на платформе Windows Phone 7, относятся к тем же типам, что и шаблоны, доступные а Expression Blend. Новый проект приложения Windows Phone 7, как, впрочем, и приложения WPF или Silverlight, можно начать как в Expression Blend, так и в Visual Studio 2010, а затем открыть для дальнейшей разработки его базового кода в той среде, где это удобнее всего сделать[7].

Особенности разработки проектов простого типа на платформе Windows Phone 7

А теперь, когда на вашем компьютере установлены все инструментальные средства, необходимые для разработки приложений на платформе Windows Phone 7, рассмотрим конкретный пример создания проекта типа `Windows Phone Application` в среде Expression Blend IDE. Для этого в диалоговом окне New Project, открывающемся по команде File⇨New Project, выберите данный тип проекта, т.е. шаблон `Windows Phone Application`, и присвойте новому проекту имя `FirstPhoneApp`.

[7] В этой главе рассматриваются только шаблоны проектов, доступные в Expression Blend, но вы можете поупражняться в создании нового проекта и в Visual Studio 2010.

Монтажный стол на платформе Windows Phone 7

Первое, что обращает на себя внимание, — это отображение на монтажном столе "телефонной" страницы, содержащей пользовательский интерфейс разрабатываемого приложения. Пример такой страницы приведен на рис. 7.11[8].

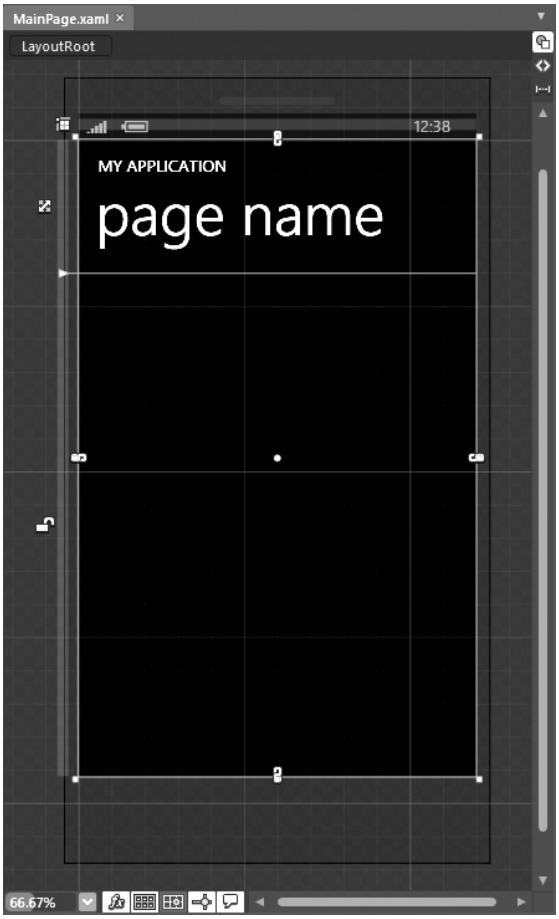


Рис. 7.11. Вид монтажного стола на платформе Windows Phone 7

Проанализировав содержимое панели Objects and Timeline, вы обнаружите, что основным диспетчером компоновки, как и в проекте приложения WPF или Silverlight, является объект `LayoutRoot` типа `Grid`. Этот объект представляет собой сетку, разбитую на две строки. В первой строке содержится блочная панель `TitlePanel` типа `StackPanel` с исходным текстом, отображаемым сверху, а во второй строке — объект `ContentPanel` подчиненной сетки типа `Grid`, где, собственно, и конструируется пользовательский интерфейс разрабатываемого приложения. Первоначальная разметка, описывающая эту систему компоновки, приведена ниже.

[8] В Silverlight for Windows Phone 7 SDK каждая отображаемая телефонная страница относится к типу `PhoneApplicationPage`, который принадлежит пространству имен `Microsoft.Controls.Phone`.

```
<!-- Объект LayoutRoot является корневой сеткой, на которой
    размещается все содержимое -->
<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>

<!-- Объект TitlePanel содержит имя приложения и заглавие страницы -->
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
  <TextBlock x:Name="ApplicationTitle" Text="MY APPLICATION"
          Style="{StaticResource PhoneTextNormalStyle}"/>
  <TextBlock x:Name="PageTitle" Text="page name" Margin="9,-7,0,0"
          Style="{StaticResource PhoneTextTitle1Style}"/>
</StackPanel>

  <!-- Объект ContentPanel служит для размещения
      дополнительного содержимого -->
  <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"/>
</Grid>
```

Системные стили, доступные на платформе Windows Phone 7

Если внимательно проанализировать определение элементов управления типа `TextBlock` в приведенной выше разметке, то можно заметить, что в них автоматически применяются стили, характерные для платформы Windows Phone 7: `PhoneTextNormalStyle` и `PhoneTextTitle1Style`. Как пояснялось в главе 5, на платформе WPF предоставляется ряд простых стилей, которые относятся к категории Simple Styles и могут служить в качестве отправной точки для последующей специальной настройки под конкретные нужды разрабатываемого приложения. Аналогичным образом в приложении Windows Phone 7 можно (и должно, как показывают наилучшие примеры из практики оформления пользовательских интерфейсов) применять ряд встроенных системных стилей, относящихся к категории System Styles. Эти применяемые по умолчанию стили можно найти и просмотреть по указанной категории в библиотеке ресурсов или на панели Assets, как показано на рис. 7.12.

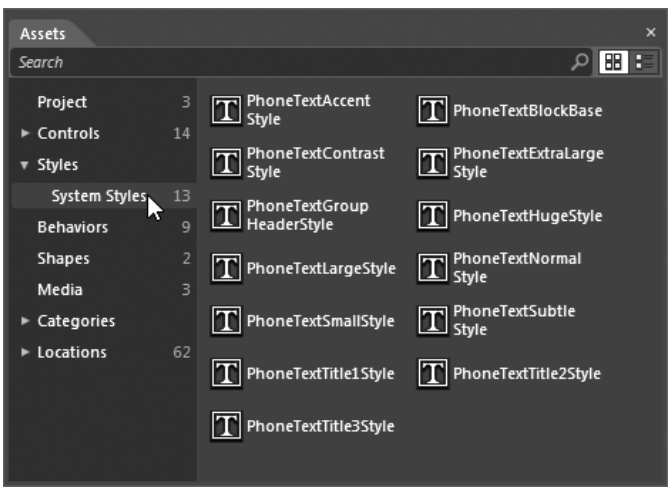


Рис. 7.12. Системные стили, доступные на платформе Windows Phone 7

В процессе разработки пользовательского интерфейса эти стили связываются с элементами управления на панели Properties, как пояснялось в главе 5. А еще проще выбрать нужный стиль из категории System Styles в библиотеке ресурсов и перетащить его на монтажный стол. В качестве упражнения попробуйте перетащить выбранные объекты стилей на объект `ContentPanel` типа `Grid` и посмотреть, что из этого получится. Но не забудьте после этого удалить эти объекты стилей.

Построение представления данных с перечислением подробностей на панели Data

На данной стадии работы над рассматриваемым здесь примером проекта можете воспользоваться любыми способами, рассмотренными в предыдущих главах, для создания пользовательского интерфейса. В частности, можете воспользоваться панелью Data для построения конкретного представления данных с перечислением подробностей, создать специальный шаблон и простую анимацию[9].

Прежде всего нужно импортировать классы `PurchaseOrder` и `PurchaseOrders`, созданные в главе 6, по команде меню Project⇨Add Existing Item[10]. После этого постройте свой проект по команде меню Project⇨Build Project и далее воспользуйтесь панелью Data для создания нового источника данных объекта, привязываемого к специальной коллекции типа `PurchaseOrders`. Как пояснялось в главе 6, новый источник данных объекта можно создать, выполнив следующие действия.

1. Щелкните на кнопке раскрывающегося списка Create data source, доступного на панели Data.
2. Выберите из этого списка вариант Create Object Data Source.
3. Выберите в открывшемся диалоговом окне нужный класс (в данном случае `PurchaseOrders`).

Определив источник данных объекта, перетащите узел `Description` из панели Data на второй объект типа `Grid`, именуемый по умолчанию `ContentPanel`. Далее щелкните на кнопке Details Mode в левом верхнем углу панели Data, выберите все узлы ниже класса `PurchaseOrders` и перетащите их из панели Data на монтажный стол, чтобы создать взаимосвязь с перечисляемыми подробностями.

Рис. 7.13. Компоновка пользовательского интерфейса с привязкой данных

[9] Все эти вопросы подробно рассматривались в предыдущих главах, и поэтому далее приводятся лишь самые общие инструкции по созданию проекта. За более подробной справкой обращайтесь к соответствующим главам и разделам книги.

[10] Напомним, что эти классы использовались в качестве источника в различных операциях привязки данных.

По завершении этих действий монтажный стол должен выглядеть так, как показано на рис. 7.13, а также на цветной вклейке. Вы вольны изменить как угодно размеры, расположение или конфигурацию любых элементов пользовательского интерфейса, воспользовавшись панелью Properties. Как показано на рис. 7.13, изменения были внесены в свойство `Background` объекта `ContentPanel` типа `Grid`, а также в расположение различных элементов пользовательского интерфейса.

Создание интерактивной графики

В проектах на платформе Windows Phone 7 имеется также возможность создавать интерактивную графику, хотя выбор готовых геометрических объектов несколько ограничен. Но вы можете по-прежнему рисовать произвольные формы инструментами Pen и Pencil (см. главу 2). Кроме того, можете выбрать элементарные геометрические формы прямоугольника (Rectangle), эллипса (Ellipse) и линии (Line) на панели Tools[11].

Для целей рассматриваемого здесь проекта добавьте новый диспетчер компоновки типа `Canvas` к объекту `ContentPanel` типа `Grid`. Напомним, что отдельные графические элементы можно перетащить из панели Assets (но не из библиотеки ресурсов) на панель Objects and Timeline или же выбрать их по отдельности и нарисовать мышью на монтажном столе[12].

Нарисуйте эллипс (графический элемент типа `Ellipse`) на новом виртуальном полотне (объекте типа `Canvas`) и добавьте к нему объект поведения типа `MouseDragElementBehavior`. Для этого найдите данный объект поведения в категории Behaviors на панели Assets, выберите его и перетащите на эллипс, находящийся в данный момент на монтажном столе.

Далее выберите новый объект поведения типа `MouseDragElementBehavior` на панели Objects and Timeline, а затем установите флажок, активизирующий свойство `ConstrainToParentBounds` на панели Properties, как показано на рис. 7.14.

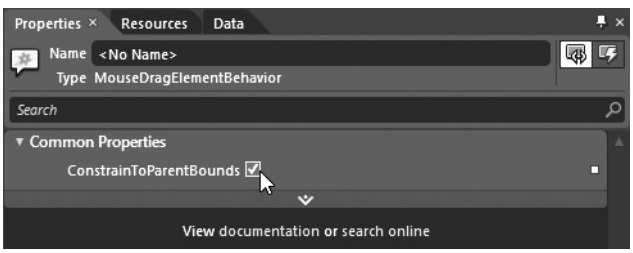


Рис. 7.14. Настройка объекта поведения типа `MouseDragElementBehavior`

Теперь можете запустить свое приложение на выполнение, нажав функциональную клавишу <F5> или комбинацию клавиш <Ctrl+F5>. Как видите, эмулятор Windows Phone 7 запускается автоматически для размещения вашего приложения! Проверьте, сможете ли вы взаимодействовать с графикой посредством мыши и просматривать подробности, перечисляемые при выборе отдельных элементов из списка.

Создание специального шаблона элемента управления

И последний прием, которым вам предстоит воспользоваться в рассматриваемом здесь примере проекта, состоит в создании шаблона элемента управления из графики,

[11] Подробнее о работе с инструментами создания графики в Expression Blend см. в главе 2.

[12] Подробнее о работе с диспетчерами компоновки и элементами управления в среде Expression Blend IDE см. в главе 4.

как пояснялось в разделе “Создание стилизованного шаблона из графики” главы 5. В качестве напоминания ниже вкратце поясняется, как это делается.

Нарисуйте сначала произвольную геометрическую форму инструментом Pen в нижней части объекта `ContentPanel` типа `Grid`, а затем щелкните на этой форме правой кнопкой мыши и выберите команду Make Into Control из всплывающего контекстного меню, как показано на рис. 7.15.

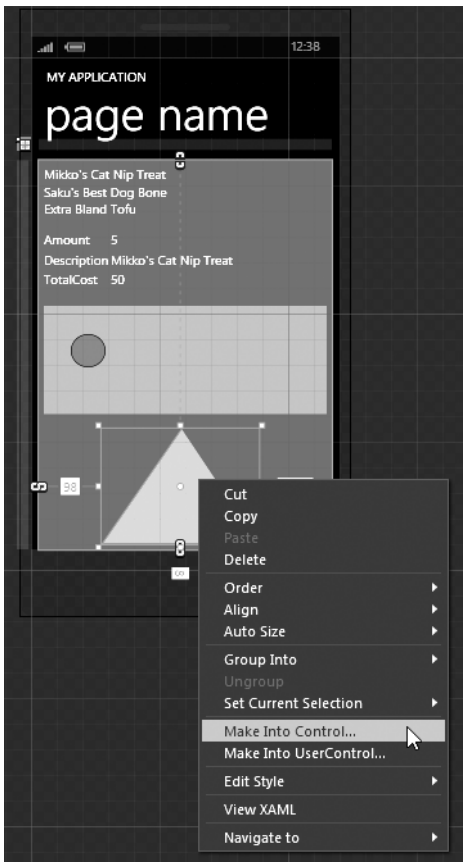


Рис. 7.15. Создание нового шаблона элемента управления из существующей графики

Выберите элемент управления типа `Button` в открывшемся диалоговом окне Make Into Control, как показано на рис. 7.16. Присвойте ему имя, например `TriangleButton` (треугольная кнопка), и сохраните этот новый ресурс объекта на уровне приложения.

В данный момент вновь созданный шаблон отображается в среде Expression Blend IDE средствами визуального конструктора шаблонов. Напомним, что вы вольны изменить свойства элементов пользовательского интерфейса, дополнить их интерактивными средствами, воспользовавшись диспетчером визуальных состояний (VSM) и панелью States, как поясняется в главе 5, а также внести любые другие коррективы в данный шаблон. Поэтому уделите немного времени правке этого шаблона, а затем, когда будете удовлетворены полученными результатами, перестройте свой проект по команде меню Project⇨Build Project.

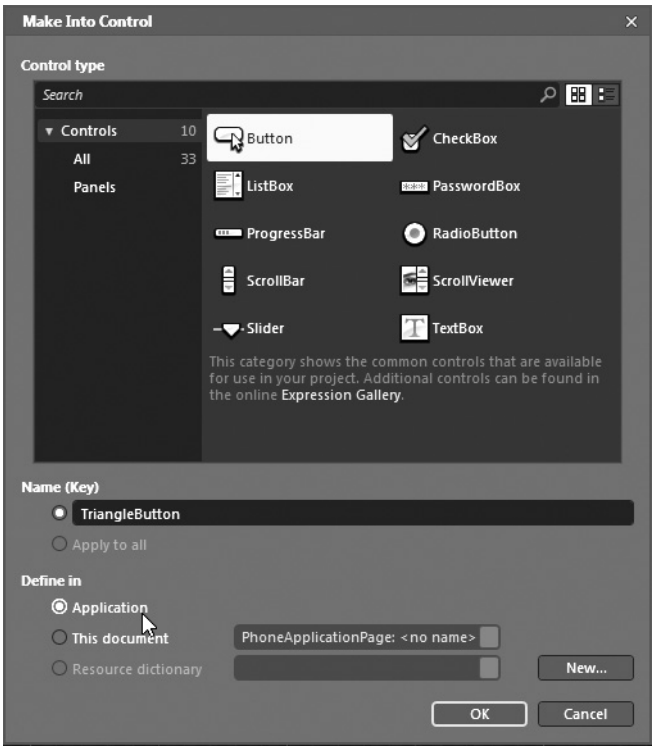


Рис. 7.16. Шаблон элемента управления `TriangleButton`

Примечание. В решении рассматриваемого здесь проекта, доступном из загружаемого архива примеров проектов к данной книге, конкретное визуальное состояние было введено в упомянутый выше шаблон. В частности, когда производится щелчок на кнопке, оформляемой по данному шаблону, ее размеры немного сокращаются посредством графического преобразования. Кроме того, в данный шаблон была введена анимация эффекта инерционности движения типа `BounceOut`, чтобы сделать более привлекательным визуально переход кнопки, оформляемой по этому шаблону, из одного состояния в другое.

Обработка события типа `Click`

И на последней стадии работы над рассматриваемым здесь примером проекта вам предстоит организовать обработку события типа `Click`, наступающего в том случае, если щелкнуть кнопкой мыши на элементе управления типа `Button`, сформированном по упомянутому выше шаблону из простой геометрической формы треугольника. Для этого щелкните на кнопке Events панели Properties и введите приведенный ниже исходный код обработчика событий типа `Click`.

```
private void Button_Click(object sender,
  System.Windows.RoutedEventArgs e)
{
  MessageBox.Show("Yippy! My phone app rocks!");
}
```

Настройка эмулятора Windows Phone 7 на панели Device

Прежде чем запускать свое приложение на выполнение, откройте файл разметки главной страницы `MainPage.xaml` в редакторе XAML на монтажном столе. А затем откройте панель Device (Устройство) из меню Window, как показано на рис. 7.17.

Как показано на рис. 7.18, на этой новой для среды Expression Blend IDE панели имеется возможность настроить целый ряд параметров, касающихся порядка обработки приложения Windows Phone 7 при выполнении проекта. По умолчанию в среде Expression Blend IDE используется эмулятор Windows Phone 7, установленный вместе с SDK, но вы можете подключиться к конкретному мобильному устройству, совместимому с платформой Windows Phone 7, для выполнения своего приложения непосредственно на нем. Кроме того, можете воспользоваться панелью Device, чтобы изменить ориентацию и цветовую палитру отображения.

Таким образом, для создания простого иллюстративного проекта приложения, предназначенного для мобильного устройства и снабженного привязкой данных, векторной графикой, реакцией на действия мыши и специальным шаблоном элемента управления, оказалось достаточно всего нескольких строк разметки и одной строки исходного кода, специально написанного на C#. В качестве примера на рис. 7.19 (а также на цветной вклейке) показано приложение, выполняющееся в эмуляторе. (В данном примере изменен текст надписей в элементах управления типа `TextBlock`, содержащихся в диспетчере компоновки `TitlePanel`.)

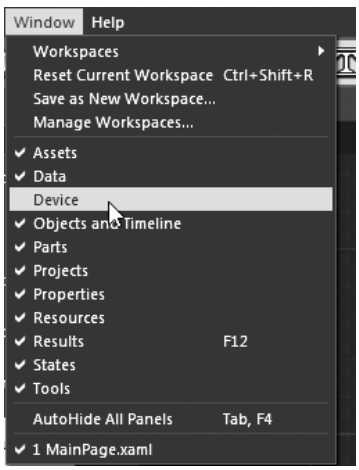


Рис. 7.17. Открытие панели Device

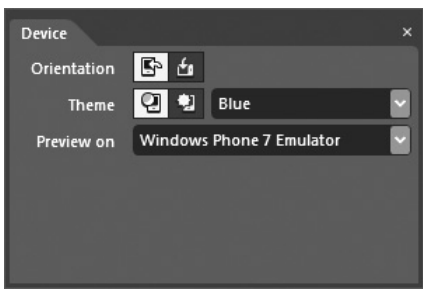


Рис. 7.18. На панели Device можно определить порядок запуска проектов на выполнение

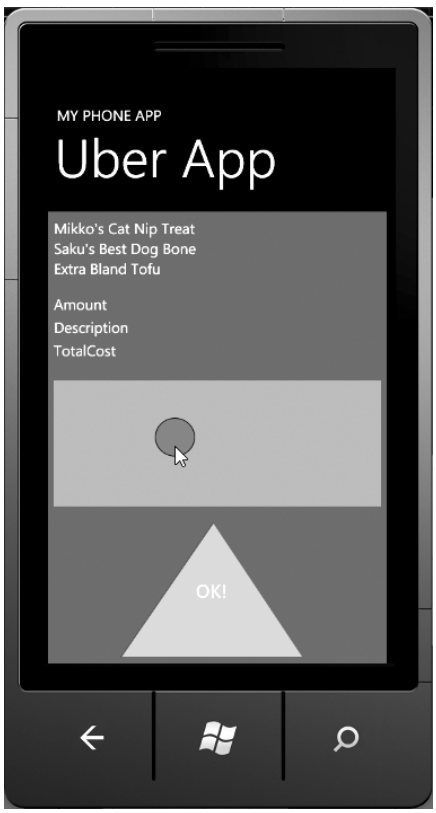


Рис. 7.19. Первое приложение, созданное на платформе Windows Phone 7

Примечание. Если вы наведете курсор мыши на правую верхнюю часть эмулятора, появится ряд элементов управления эмулятора. С помощью этих элементов управления можно изменить ориентацию эмулятора или масштаб отображения и вообще закрыть эмулятор.

Исходный код. Исходный код примера проекта `FirstPhoneApp` находится в папке `Ch 7 Code` загружаемого архива примеров проектов к данной книге.

Особенности разработки проектов панорамного типа на платформе Windows Phone 7

В предыдущем примере проекта был использован самый простой из всех доступных на платформе Windows Phone 7 шаблон проекта `Windows Phone Application`. По умолчанию приложение, разрабатываемое по этому шаблону проекта, состоит из единственной страницы отображаемого пользовательского интерфейса, которую можно скомпоновать как угодно. Разумеется, эта единственная страница может состоять из диспетчера компоновки, отображающего динамическое содержимое с помощью специальных объектов типа `UserControl`, но, вообще говоря, все содержимое пользовательского интерфейса в таком приложении не выходит за рамки экрана мобильного устройства.

В качестве альтернативы на платформе Windows Phone 7 предоставляется шаблон проекта `Windows Phone Panorama Application`. *Приложения панорамного типа* предоставляют уникальную возможность для просмотра содержимого пользовательского интерфейса на длинном прокручиваемом по горизонтали виртуальном полотне, выходящем за рамки экрана мобильного устройства.

Когда пользователь проводит пальцем по экрану мобильного устройства (в эмуляторе аналогичный эффект достигается мышью), изображение плавно прокручивается к следующей укрупненной части графического представления с помощью многослойной анимации, в ходе которой осуществляется плавное панорамирование с разной скоростью, подобно эффектам параллакса. Принцип действия панорамного отображения наглядно демонстрируется на рис. 7.20, а также на цветной вклейке. Обратите внимание на то, что вся область отображения оказывается намного больше площади экрана мобильного устройства, работающего на платформе Windows Phone 7.

Рис. 7.20. Панорамное отображение дает пользователю возможность переходить от одного графического представления к другому путем прокрутки

Для того чтобы стали более понятными особенности разработки приложений панорамного типа на платформе Windows Phone 7, достаточно видоизменить немного первоначальный код, автоматически сформированный в среде Expression Blend IDE. С этой целью создайте новый проект, выбрав сначала команду меню File⇨New Project, а затем шаблон проекта `Windows Phone Panorama`. Присвойте новому проекту имя `PanoramaDemoApp`.

Исследование первоначального иерархического представления объектов

Проанализируйте содержимое панели Objects and Timeline, обратив внимание на то, что у объекта `LayoutRoot` типа `Grid` имеется порожденный объект типа `Panorama`. Именно этот объект и отвечает за плавные переходы от одного элемента панорамного представления типа `PanoramaItem` к другому. В данный момент имеются два таких элемента, причем каждый из них содержит единственный элемент управления типа `ListBox`, как показано на рис. 7.21.

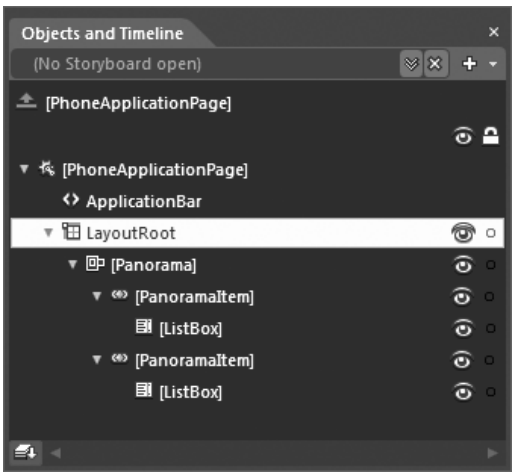


Рис. 7.21. Первоначальное иерархическое представление объектов

А теперь запустите свое приложение на выполнение, нажав функциональную клавишу <F5> или комбинацию клавиш <Ctrl+F5>. После загрузки эмулятора воспользуйтесь мышью, чтобы сымитировать проводку пальцем влево или вправо по экрану мобильного устройства для плавного прокручивания содержимого экрана и просмотра данных в каждом элементе типа `PanoramaItem`. По завершении экспериментов с панорамным представлением закройте эмулятор на некоторое время.

Просмотр разметки элементов панорамного представления

Вернитесь в Expression Blend и откройте файл разметки главной страницы `MainPage.xaml` в редакторе XAML на монтажном столе. Обратите внимание на то, что каждый объект типа `PanoramaItem` настроен на извлечение данных из связанного с ним шаблона данных, как показано в приведенном ниже фрагменте разметки.

```
<!-- Первый элемент панорамного представления -->
<controls:PanoramaItem Header="first item">
  <!-- Двухстрочный список с переносом текста на новую строку -->
  <ListBox ItemsSource="{Binding Items}" Margin="0,0,-12,0">
```

```
    <ListBox.ItemTemplate>
      <DataTemplate>
        <StackPanel Margin="0,0,0,17" Width="432">
          <TextBlock Text="{Binding LineOne}" TextWrapping="Wrap"
            Style="{StaticResource PhoneTextExtraLargeStyle}" />
          <TextBlock Text="{Binding LineTwo}" TextWrapping="Wrap"
            Margin="12,-6,12,0"
            Style="{StaticResource PhoneTextSubtleStyle}" />
        </StackPanel>
      </DataTemplate>
    </ListBox.ItemTemplate>
  </ListBox>
</controls:PanoramaItem>
```

Примечание. Подобные шаблоны данных настроены на извлечение жестко запрограммированных данных из класса `ViewModel` текущего приложения. Убедитесь в этом сами, проанализировав исходный код из файла `MainViewModel.cs`, находящегося в папке `ViewModels` вашего проекта.

Все объекты типа `PanoramaItem` содержатся в одном элементе управления типа `Panorama`, который отвечает за прокрутку анимации и правильное отображение содержимого пользовательского интерфейса на экране мобильного устройства. Помимо того что элемент управления типа `Panorama` содержит объекты типа `PanoramaItem`, он определяет также файл изображения, используемого в качестве фона. По умолчанию в новых проектах, разрабатываемых по шаблону `Windows Phone Panorama`, используется файл образцового изображения `PanoramaBackground.png`, автоматически добавляемый в проект. Как показано в приведенном ниже фрагменте разметки, объект типа `ImageBrush` (см. главу 2) используется для раскраски кистью фона поверхности виртуального полотна.

```
<!-- Элемент управления типа Panorama -->
<controls:Panorama Title="my application">
  <controls:Panorama.Background>
    <ImageBrush ImageSource="PanoramaBackground.png"/>
  </controls:Panorama.Background>

  <!-- Здесь перечисляются элементы панорамного представления типа PanoramaItem -->
  ...
</controls:Panorama>
```

Изменение фона панорамного представления

А теперь необходимо заменить используемое по умолчанию фоновое изображение специально подобранным изображением. С этой целью выберите объект типа `Panorama` на панели Objects and Timeline и найдите свойство `Background`. На рис. 7.22 приведено текущее значение этого свойства.

Прежде чем изменять исходный фон, необходимо создать для него специальную графику. С этой целью откройте избранный вами файл цифрового изображения в приложении Paint, запускаемом на выполнение в Windows по команде Пуск⇨Все программы⇨Стандартные⇨Paint. После этого измените размеры изображения до величины `1024×768` пикселей, т.е. до размеров исходного фонового изображения, щелкнув на кнопке Resize and Skew (Растяжение и наклон) или нажав комбинацию клавиш <Ctrl+W>, как показано на рис. 7.23.

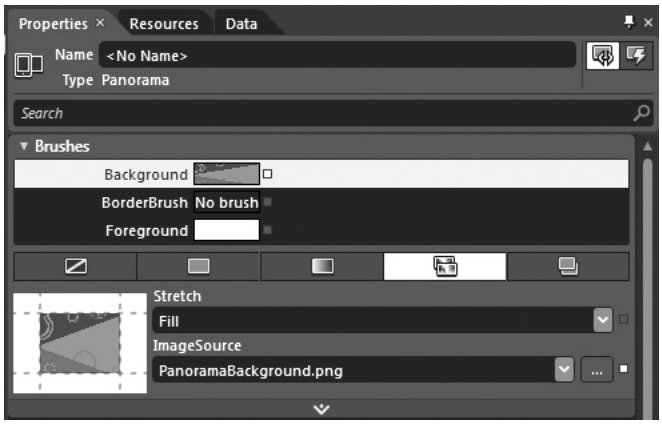


Рис. 7.22. Первоначальные данные фонового изображения для объекта типа `Panorama`

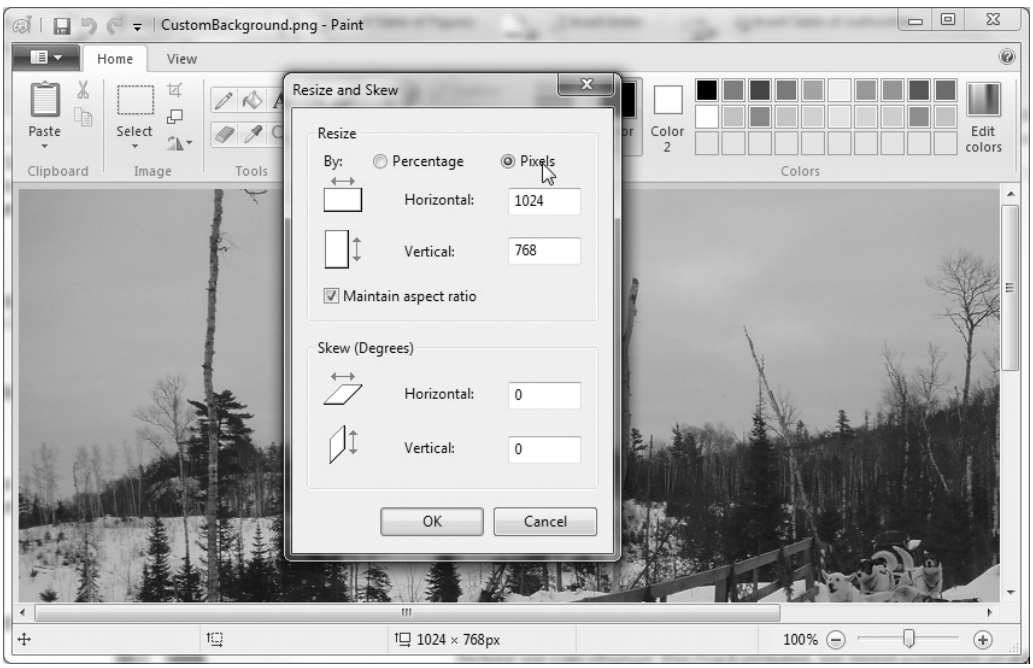


Рис. 7.23. Настройка специальной графики в прикладной программе Paint

Сохраните видоизмененное изображение по стандартной команде сохранения в файле формата PNG, выбрав для него подходящее место на жестком диске компьютера. Вернитесь в Expression Blend и внесите изменения в свойство `Background` элемента управления типа `Panorama`, чтобы заменить исходный фон на специальную графику. С этой целью щелкните на кнопке с изображением эллипса рядом с текстовым полем свойства `ImageSource`, как показано на рис. 7.24.

Примечание. В среде Expression Blend IDE рекомендуется вводить данные изображения в текущий проект приложения в виде двоичного источника. Но для целей рассматриваемого здесь примера это не имеет никакого значения.

Еще раз запустите приложение на выполнение, чтобы посмотреть, как графика прокручивается на новом фоне благодаря объекту типа `Panorama` (рис. 7.25).

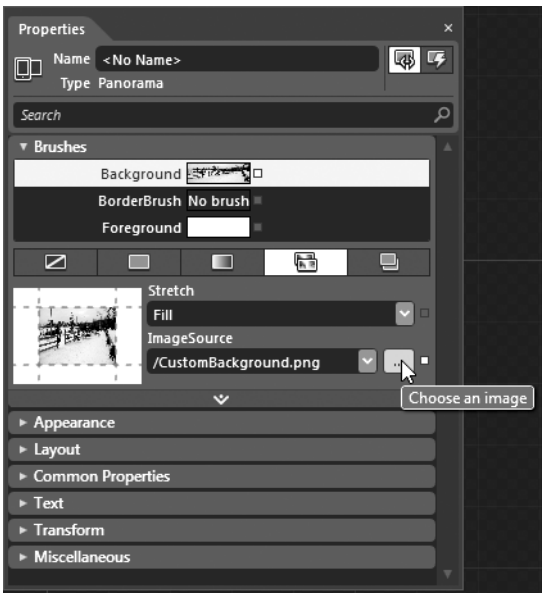


Рис. 7.24. Выбор нового фонового изображения для элемента управления типа `Panorama`

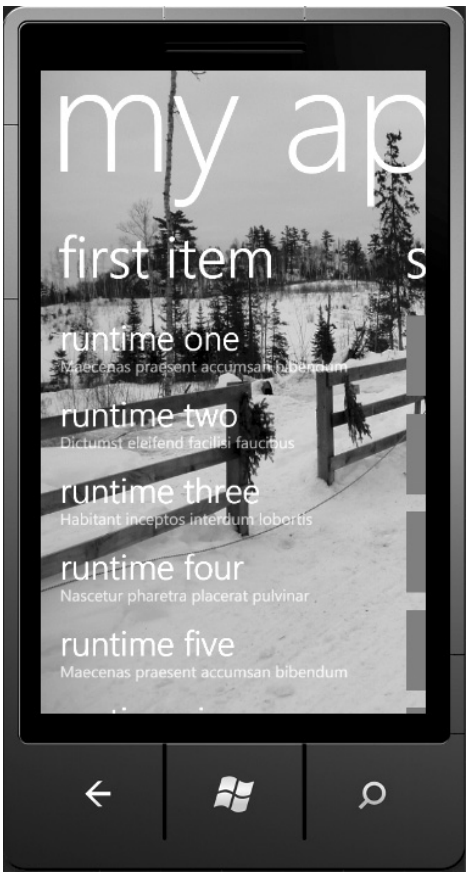


Рис. 7.25. Новый фон, специально подобранный для приложения панорамного типа

Добавление нового объекта типа `PanoramaItem`

В качестве последнего изменения в рассматриваемом здесь проекте коллекция, хранящаяся в родительском узле `Panorama`, будет дополнена третьим объектом типа `PanoramaItem`. Сделать это проще всего, щелкнув правой кнопкой мыши на узле `Panorama` в иерархическом представлении на панели Objects and Timeline и выбрав команду Add PanoramaItem из всплывающего контекстного меню (рис. 7.26).

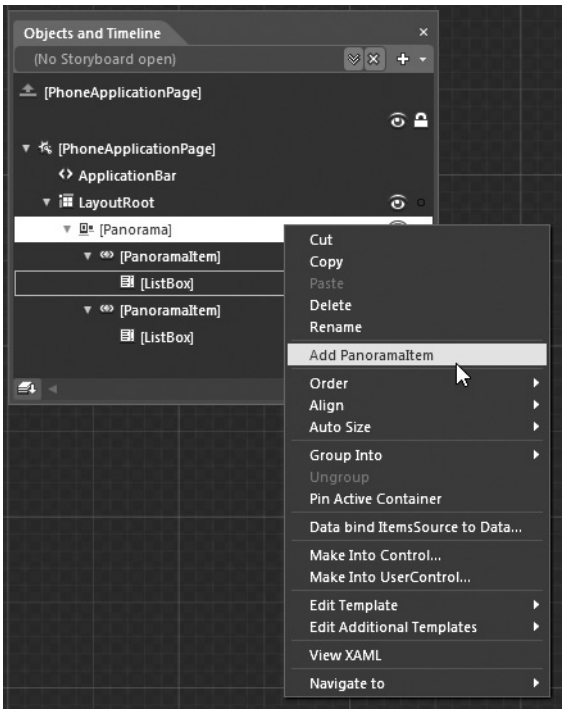


Рис. 7.26. Добавление нового объекта типа `PanoramaItem`

Как только вы выберете указанную выше команду меню, на монтажном столе появится новый объект типа `PanoramaItem`, к которому можно затем добавить любое количество элементов управления, графики и пр. Ниже приведена исходная разметка этого объекта в коде XAML.

```
<controls:PanoramaItem Header="item4">
  <Grid/>
</controls:PanoramaItem>
```

В рассматриваемом здесь простом примере проекта достаточно лишь изменить свойство `Header` объекта типа `PanoramaItem` на панели Properties или добавить новый элемент управления либо графику в диспетчер компоновки типа `Grid` просто для того, чтобы проверить панорамное представление пользовательского интерфейса в действии. В частности, можете добавить один элемент управления типа `Button` и организовать обработку события типа `Click`, чтобы вывести на экран сообщение из метода `MessageBox.Show()`, когда на кнопке производится щелчок. Внеся эти изменения в свой проект, еще раз запустите его на выполнение, чтобы проверить, прокручиваются ли объекты типа `PanoramaItem` в панорамном представлении, в то время как на заднем плане отображается графика, специально подобранная для фона.

Исходный код. Исходный код примера проекта `PanoramaDemoApp` находится в папке `Ch 7 Code` загружаемого архива примеров проектов к данной книге.

Особенности разработки проектов сводного типа на платформе Windows Phone 7

Еще одна стандартная система навигации, предоставляемая в Windows Phone 7 SDK, называется *приложением сводного типа*. Это приложение, по существу, является разновидностью системы вкладок, в которой пользователь может переходить от одного представления информации к другому связанному с ним представлению, выбирая соответствующую вкладку в верхней части экрана мобильного устройства или совершая такие жесты для ввода информации, как, например, панорамирование по горизонтали постукиванием и скольжение пальцем по экрану влево либо вправо или же резкое движение по горизонтали постукиванием и быстрой проводкой пальцем по экрану влево либо вправо. В соответствии с этой моделью программирования в корневом диспетчере компоновки типа `Grid` содержится объект типа `Pivot`, отвечающий за страничную организацию данных, где отдельные страницы представлены объектом типа `PivotItem`.

Итак, создайте новое приложение по шаблону проекта `Windows Phone Pivot Application` в диалоговом окне New Project, присвоив ему имя `PivotDemoApp`. Первоначальное иерархическое представление объектов на панели Objects and Timeline будет выглядеть так, как показано на рис. 7.27.

В объекте типа `Grid` каждого элемента сводного представления (объекта типа `PivotItem`) отображаются жестко закодированные данные, как и в рассматривавшемся ранее примере проекта приложения панорамного типа. Таким образом, исходный монтажный стол, соответствующий разметке главной страницы в файле `MainPage.xaml`, будет выглядеть так, как показано на рис. 7.28.

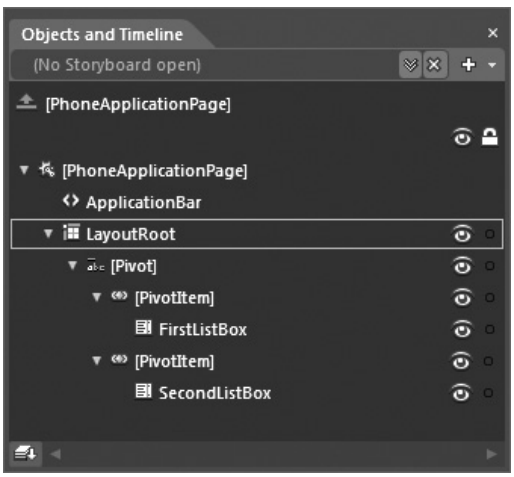


Рис. 7.27. Первоначальное иерархическое представление объектов в новом приложении сводного типа

Рис. 7.28. Исходный вид монтажного стола в приложении сводного типа

Добавление нового объекта типа PivotItem

Как и объект типа PanoramaItem, каждый объект типа PivotItem может быть настроен на панели Properties. В частности, настраивая свойство Header этого объекта, вы можете изменить заглавие на вкладке каждой страницы сводного представления. Для этого достаточно выбрать каждый объект типа PivotItem по очереди на панели Objects and Timeline и соответственно установить его свойство Properties. Кроме того, в каждом из объектов подчиненной сетки типа Grid можете удалить элементы управления типа ListBox, чтобы скомпоновать пользовательский интерфейс приложения по своему желанию любыми способами и средствами, представленными в предыдущих главах.

А для того чтобы добавить новый объект типа PivotItem, щелкните правой кнопкой мыши на узле Pivot в иерархическом представлении на панели Objects and Timeline и выберите команду Add PivotItem из всплывающего контекстного меню, как показано на рис. 7.29.

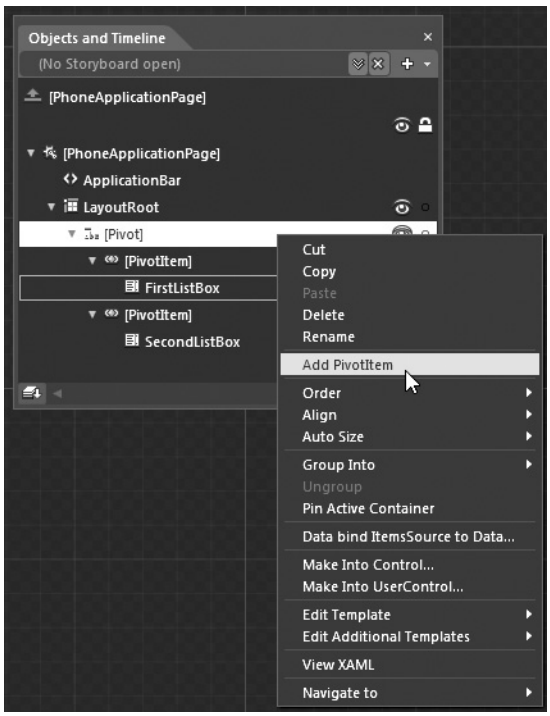


Рис. 7.29. Добавление объекта типа PivotItem

Компоновка графического пользовательского интерфейса приложения сводного типа

На данной стадии работы над рассматриваемым здесь проектом вы вольны скомпоновать пользовательский интерфейс приложения сводного типа так, как вам заблагорассудится. В качестве примера на рис. 7.30 показано простое графическое изображение улыбающегося лица, нарисованное для нового объекта типа Pivot инструментами Ellipse и Pen. Отдельные графические элементы были сгруппированы в новый диспетчер компоновки типа Grid, для чего достаточно нажать клавишу <Shift>, щелкнуть правой кнопкой мыши на каждом графическом элементе непосредственно на монтажном столе и выбрать команду Group Info из всплывающего контекстного меню (см. рис. 7.30).

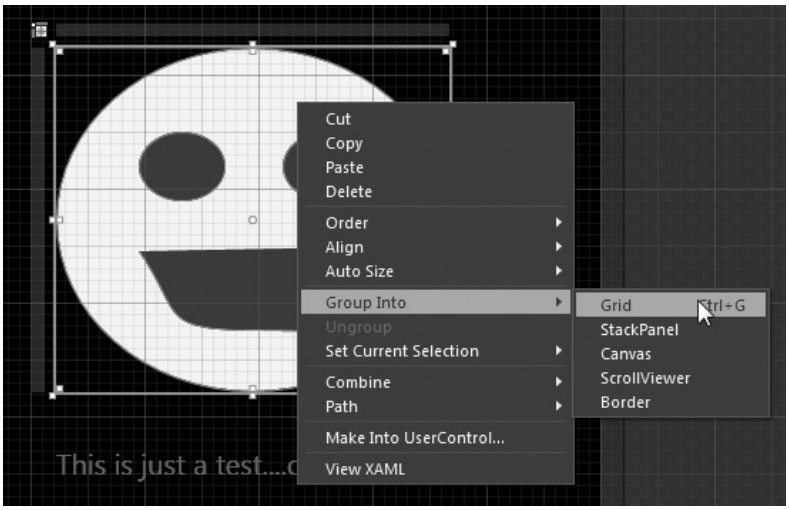


Рис. 7.30. Компоновка пользовательского интерфейса новой страницы сводного представления данных

Сгруппировав графические элементы в диспетчер компоновки типа `Grid`, присвойте этому объекту имя `mrHappyGrid` на панели Properties. Вам придется далее организовать взаимодействие с этим диспетчером компоновки, используя объект поведения типа `ControlStoryboardAction`. Как показано на рис. 7.30, в родительский объект типа `Grid` добавлен также элемент управления типа `TextBlock`, в котором отображается текст `This is just a test....only a test` (Это всего лишь текст, и ничего больше), выполняющий элементарную информативную функцию.

Преобразование сетки

А теперь воспользуйтесь интегрированным в Expression Blend редактором анимации, чтобы создать новый объект раскадровки для анимации преобразования как самого диспетчера компоновки, так и содержащейся в нем графики. Более подробно редактор анимации рассматривается в главе 3, а ниже приводится краткая процедура анимации преобразования сетки для целей рассматриваемого здесь примера, но вы вольны, конечно, сделать это по-своему.

1. Перейдите к панели Objects and Timeline, щелкните на кнопке New Storyboard и присвойте новой раскадровке имя `FlipHappyDude` (переворачивание изображения счастливца).
2. Выберите диспетчер компоновки `mrHappyGrid` на панели Objects and Timeline.
3. Введите на временной шкале анимации два ключевых кадра анимации на нульсекундной и двухсекундной отметках, воспользовавшись кнопкой с яйцевидной пиктограммой.
4. Перейдите к области Transform панели Properties и измените на **360** значения свойств X и Y преобразования вращением, как показано на рис. 7.31.
5. Выберите последний ключевой кадр анимации и примените эффект инерционности движения на панели Properties. (В качестве примера на рис. 7.32 показано применение эффекта инерционности движения `Elastic Out` (Упругость в конце), но вы вольны выбрать для этой цели любой другой эффект.)

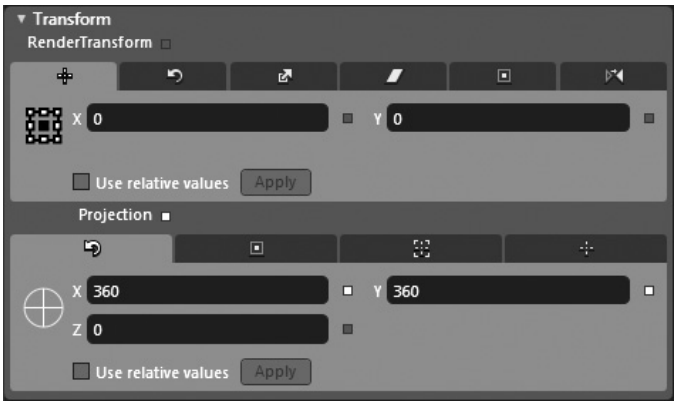


Рис. 7.31. Настройка раскадровки на анимацию преобразования объекта типа `Grid` вращением

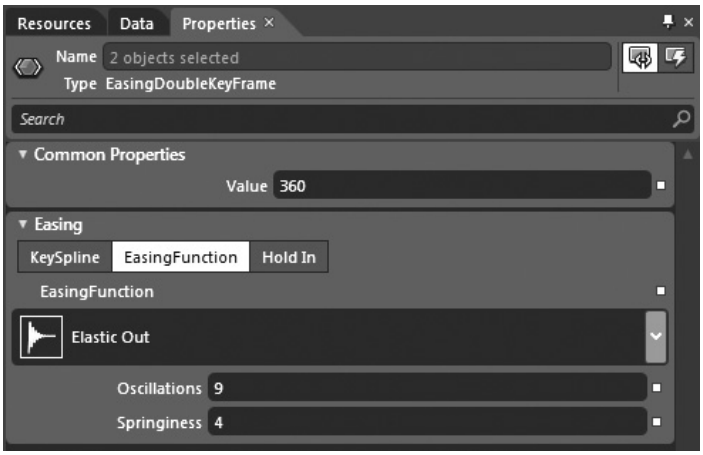


Рис. 7.32. В последнем ключевом кадре анимации применяется эффект инерционности движения

В редакторе анимации щелкните на кнопке Play панели Objects and Timeline, чтобы проверить анимацию преобразования сетки и находящихся на ней графических элементов. Удовлетворившись достигнутыми результатами, перейдите к следующему разделу этой главы.

Управление анимацией по раскадровке в разметке XAML

Для управления анимацией по созданной ранее раскадровке можно было бы написать исходный код на C# или VB, но для этой цели в рассматриваемом здесь примере проекта будет использован один из встроенных в Expression Blend объектов поведения. Итак, найдите объект поведения `ControlStoryboardAction`[13] на панели Assets, как показано на рис. 7.33, и перетащите его на объект `mrHappyGrid`, находящийся на панели Objects and Timeline.

[13] Более подробно этот объект рассматривался в главе 3.

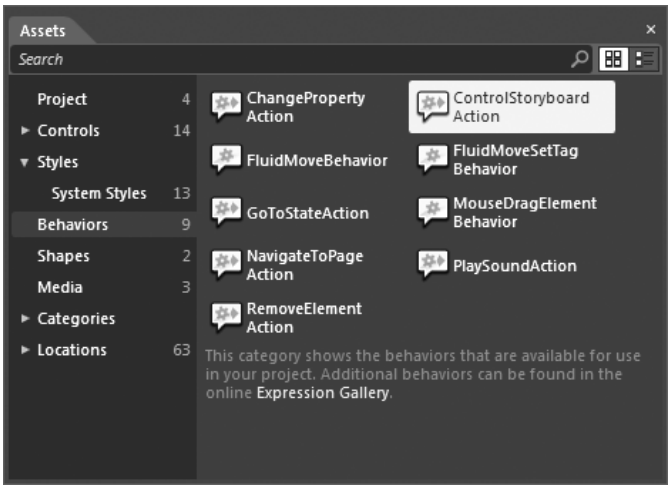


Рис. 7.33. Добавление объекта поведения типа `ControlStoryboardAction` к объекту `mrHappyGrid`

Выберите новый узел `ControlStoryboardAction` на панели Objects and Timeline, перейдите к панели Properties и щелкните на пиктограмме инструмента Artboard element picker (Селектор элементов на рабочем столе) справа от поля свойства `SourceName` (Имя источника), как показано на рис. 7.34.

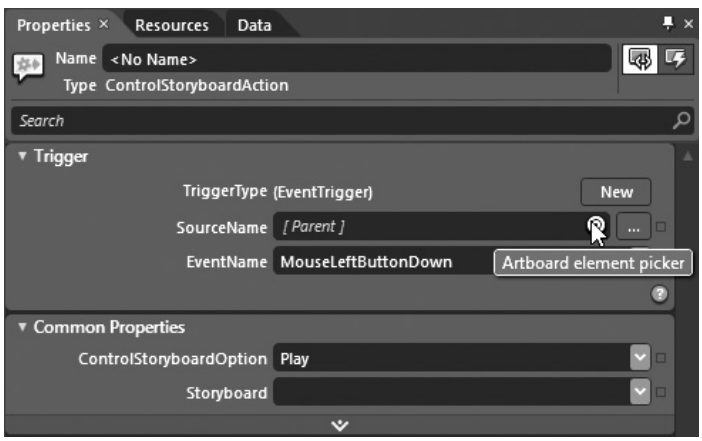


Рис. 7.34. Выбор графического элемента с помощью селектора элементов на рабочем столе для организации взаимодействия с раскадровкой

Далее щелкните на одном из графических элементов (например, на левом глазе улыбающегося лица) и выберите инициирующее событие в свойстве `EventName` (Имя события), установив подходящие значения свойств `ControlStoryboardOption` (Режим управления анимацией по раскадровке) и `Storyboard` (Раскадровка). В качестве примера на рис. 7.35 показана настройка объекта поведения на запуск анимации по раскадровке `FlipHappyDude`, когда на выбранном графическом объекте типа `Ellipse` производится щелчок левой кнопкой мыши.

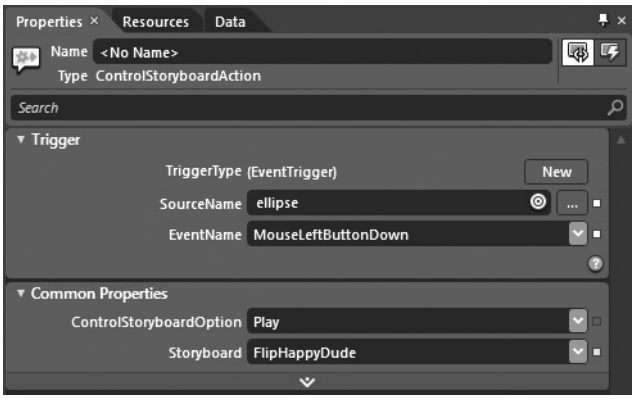


Рис. 7.35. Полная настройка объекта поведения типа `ControlStoryboardAction`

Вот, собственно, и все! Если вы запустите свое приложение на выполнение и щелкнете сначала на заглавии новой вкладки (или пролистаете страницы сводного представления мышью), а затем на левом глазу графического изображения счастливца (в данном примере проекта), то увидите воспроизводящуюся в итоге анимацию переворачивания этого изображения.

Исходный код. Исходный код примера проекта `PivotDemoApp` находится в папке `Ch 7 Code` загружаемого архива примеров проектов к данной книге.

Дополнительные ресурсы по изучению особенностей разработки приложений на платформе Windows Phone 7

Несмотря на всю простоту примеров, представленных в этой главе, они наглядно демонстрируют целый ряд полезных способов и средств разработки в Expression Blend приложений на платформе Windows Phone 7. Тем не менее вы вряд ли ограничитесь проработкой этих примеров и в конечном счете захотите разработать нечто более реальное, что, безусловно, подразумевает написание специального исходного кода. Но этот вопрос, к сожалению, выходит за рамки данной книги. Поэтому в заключение этой главы будет указан ряд полезных ресурсов, к которым вы можете обратиться в Интернете, чтобы поближе ознакомиться с особенностями разработки приложений на платформе Windows Phone 7.

Примеры проектов на платформе Windows Phone 7, доступные в MSDN

Свободное доступное через Интернет собрание документов компании Microsoft (MSDN; `http://msdn.microsoft.com`) служит главным источником сведений по разработке приложений на платформе Windows Phone 7. Перейдя на начальную страницу MSDN, вы найдете на ней графическую ссылку `phone` (телефон) на портал, посвященный разработке. Щелкните на этой ссылке, как показано на рис. 7.36.

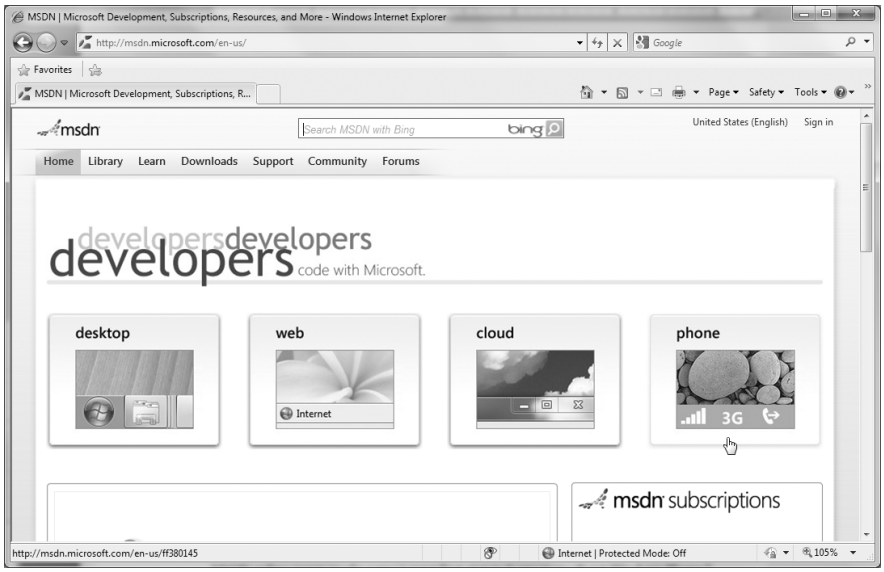


Рис. 7.36. MSDN в Интернете — главный источник сведений по разработке приложений на платформе Windows Phone 7

Выполните прокрутку вниз на открывшейся в итоге веб-странице и щелкните на ссылке `Windows Phone Development: MSDN library`, направляющей на портал, посвященный разработке приложений на платформе Windows Phone 7 (рис. 7.37)[14].

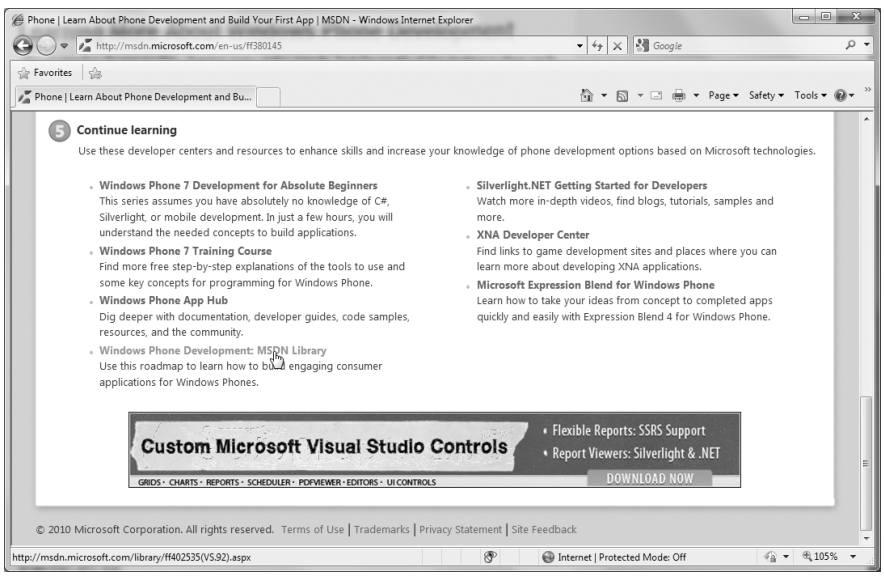


Рис. 7.37. Переход по ссылке на документацию MSDN по разработке приложений на платформе Windows Phone 7

[14] В русскоязычном варианте следует щелкнуть сначала на ссылке `Центр разработчиков Windows Phone` под рубрикой `Продолжите обучение`, а затем на ссылке `Библиотека` в верхней части открывшейся веб-страницы. — *Примеч. ред.*

Как показано на рис. 7.38, щелкнув на ссылке `Code Samples` (Примеры кода), вы обнаружите целый ряд примеров проектов, доступных для загрузки и последующего открытия в Expression Blend или Visual Studio 2010.

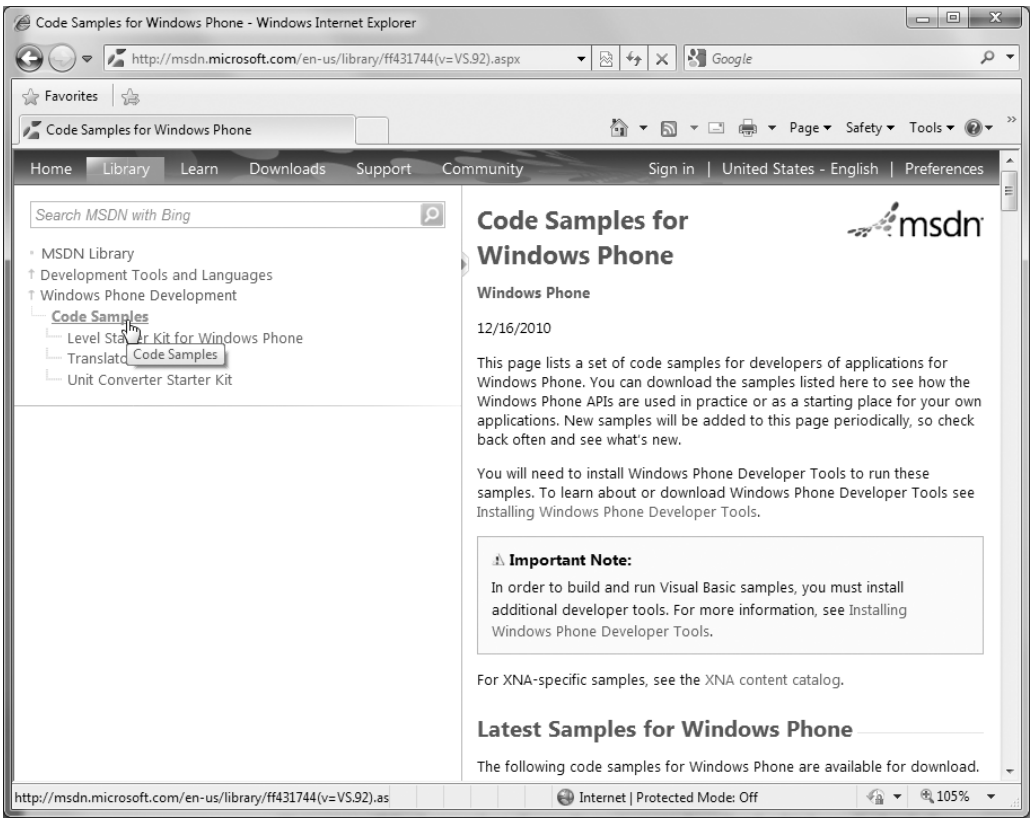


Рис. 7.38. Примеры проектов на платформе Windows Phone 7

В папке `Ch 7 Code\WeatherForecast` загружаемого архива примеров проектов к данной книге находится видоизмененный вариант проекта `Weather Forecast Sample`, загружаемого по указанной ссылке. Если вы откроете этот проект и выполните его в Expression Blend, то запустится эмулятор Windows Phone, позволяющий проверить погодные условия в различных регионах через XML-ориентированную веб-службу. (Исходный код этого проекта был немного изменен, чтобы показывать погоду в Миннеаполисе, шт. Миннесота, — любимом городе автора книги; рис. 7.39.)

Исходный код. Видоизмененный исходный код примера проекта `Weather Forecast Sample` (`WeatherForecast`) находится в папке `Ch 7 Code` загружаемого архива примеров проектов к данной книге.

Веб-сайт App Hub

Еще один полезный веб-сайт App Hub для тех, кто интересуется разработкой приложений на платформе Windows Phone 7 (а также XNA Game Studio), находится по адресу `http://create.msdn.com`. На этом веб-сайте вы найдете обширное

сообщество независимых разработчиков, интересующихся созданием и реализацией (с надеждой на прибыль) приложений на платформах Windows Phone 7 и Xbox 360. Посетив этот веб-сайт, вы сможете также создать свой собственный профиль, а за весьма умеренную плату (99 долларов в год) выставить свои приложения на продажу в интернет-магазине Windows Phone Marketplace или Xbox LIVE Marketplace. На рис. 7.40 приведена начальная страница веб-сайта App Hub.

Уже этих двух ресурсов будет достаточно для дальнейшего самостоятельного изучения вопросов разработки приложений на платформе Windows Phone 7. Вооружившись приобретенными с их помощью знаниями, вы можете смело переходить к следующей, завершающей эту книгу главе, где будет рассмотрено назначение инструментального средства SketchFlow. Из следующей главы вы узнаете, что в среде Expression Blend IDE предоставляется целый набор дополнительных инструментальных средств для быстрого создания прототипов приложений на платформах WPF и Silverlight.

Рис. 7.39. Прогноз погоды в Миннеаполисе, шт. Миннесота

Рис. 7.40. Веб-сайт App Hub — еще один полезный ресурс в Интернете для тех, кто интересуется разработкой приложений на платформе Windows Phone 7

Резюме

В этой относительно короткой, но занимательной главе было показано, что процесс построения пользовательских интерфейсов приложений на платформе Windows Phone 7 в Expression Blend мало чем отличается от аналогичного процесса для приложений на платформе WPF или Silverlight. Тем не менее в этой главе были рассмотрены некоторые важные особенности разработки приложений для мобильных устройств на платформе Windows Phone 7.

Сначала было показано, каким образом устанавливается комплект инструментальных средств разработки Windows Phone 7 SDK, а затем рассмотрены различные шаблоны проектов Windows Phone 7, доступные в Expression Blend. В первом примере проекта приложения простого типа было показано, что на платформе Windows Phone 7 можно пользоваться теми же самыми способами и средствами привязки данных, анимации и создания специальных шаблонов управления, которые были представлены в предыдущих главах.

А в двух других примерах проектов были продемонстрированы обе стандартные для приложений на платформе Windows Phone 7 системы навигации: панорамного и сводного типа. По ходу работы над этими примерами проектов вы имели возможность немного видоизменить исходную разметку и код, чтобы получить некоторое представление о том, как создаются пользовательские интерфейсы подобных приложений. И в заключение главы были представлены полезные и свободно доступные в Интернете ресурсы, дающие возможность углубленно изучить различные вопросы и особенности разработки полноценных, реальных приложений на платформе Windows Phone 7.

ГЛАВА 8

Создание прототипов средствами SketchFlow

В состав пакета Expression Studio Ultimate Edition 4 входит компонент SketchFlow, который позволяет разработчикам создавать интерактивные прототипы приложений на платформах WPF и Silverlight. В начале этой главы представлено краткое введение в прототипы в контексте жизненного цикла разработки программного обеспечения. А затем рассматриваются основные вопросы разработки средствами SketchFlow на конкретном примере проекта приложения, создаваемого в Expression Blend.

В остальной части главы рассматривается процесс создания рабочего прототипа типичного для электронной коммерции приложения на платформе Silverlight[1]. Прорабатывая материал этой главы, вы научитесь работать с различными инструментальными средствами SketchFlow, в том числе с панелью SketchFlow Map, ознакомитесь с тем, как составляются требования к проекту, исходя из созданного прототипа, а также узнаете, как пользоваться прототипом в качестве отправной точки для разработки предполагаемого программного обеспечения промышленного образца.

Примечание. Напомним, что компонент SketchFlow доступен только в составе пакета Expression Studio Ultimate Edition 4. Правда, в пробную версию этого пакета, действующую в течение 60 дней, также входит среда SketchFlow для создания прототипов.

Для чего нужно создание прототипов приложений

Время от времени разработчикам выпадает случай создать мелкомасштабное приложение, например, простой конфигуратор для внутреннего употребления, видеоигру для себя, цифровой организатор мультимедийных средств или диспетчер контактов. Подобные проекты обычно разрабатываются в одиночку и не требуют составления планов на будущее, словаря данных, технических требований или другой проектной документации. Вместо этого разработчик обращается непосредственно к модели разработки программного обеспечения по принципу “программируй и исправляй ошибки”, т.е. он просто начинает писать исходный код, отлаживая и исправляя по ходу дела программные ошибки.

[1] Как и следовало ожидать, порядок работы с компонентом SketchFlow для создания прототипа приложения на платформе WPF оказывается по большей части таким же, как и при создании прототипа приложения на платформе Silverlight.

Правда, большинство разработчиков (и их руководители) хорошо понимают особое значение официально утверждаемого жизненного цикла разработки проекта для организации крупномасштабных работ по созданию программного обеспечения. В частности, во многих коммерческих организациях для этой цели применяется модель RUP (Rational Unified Process — Рационально унифицированный процесс разработки), в которой формализуются основные повторяющиеся стадии типичного проекта разработки программного обеспечения. Модель RUP и многие другие методологии разработки программного обеспечения стимулируют *создание прототипа* приложения на ранней стадии, чтобы сымитировать внешний вид и функции программного продукта, готового для промышленной эксплуатации.

Создание прототипов приложений — отличный способ получить отзывы клиентов и базового контингента пользователей относительно удобства компоновки пользовательского интерфейса и полноты функциональных возможностей программного обеспечения. С годами у разработчиков накопился немалый опыт создания прототипов различными инструментальными средствами, включая визуальные конструкторы графического пользовательского интерфейса в среде Visual Studio. Все это, конечно, замечательно, но создание прототипов таит в себе немало скрытых и опасных препятствий, на некоторые из которых приходится наталкиваться буквально сразу. Ниже перечислены подобные препятствия.

- Клиент может легко принять прототип за реальное программное обеспечение, поскольку его пользовательский интерфейс зачастую очень похож на пользовательский интерфейс полноценной прикладной программы.
- Прототипы нередко отвергаются, как только поступают отзывы от клиентов. Это вынуждает разработчиков, по существу, воссоздавать тот же самый пользовательский интерфейс, но уже на уровне проекта программного обеспечения промышленного образца.
- Многие инструментальные средства не позволяют ни учесть отзывы клиента в самом проекте, ни сформировать официальную документацию из подобных замечаний к проекту.
- Если клиент пожелает внести собственные изменения в прототип, для этого ему понадобится не только инструментальное средство создания прототипов, но и умение им пользоваться[2].

В идеальном случае в распоряжении разработчиков должно быть инструментальное средство, которое может быть использовано для построения пользовательского интерфейса, *наглядно* демонстрирующего самому пользователю, что это лишь первоначальный вариант рабочего решения, а не полностью готовый программный продукт. Такое инструментальное средство должно также предоставлять клиентам возможность вносить свои личные комментарии в проект, а разработчикам — использовать эту информацию при составлении документации. Более того, такое инструментальное средство должно давать клиентам возможность просматривать прототип и взаимодействовать с ним, даже если у них отсутствует собственная копия инструментального средства для создания прототипов. К счастью, именно эти и многие другие средства предоставляются в среде Expression Blend IDE благодаря компоненту SketchFlow.

[2] А если этим инструментальным средством окажется Visual Studio, то можете себе представить, насколько сложной для освоения будет эта интегрированная среда разработки для неподготовленного пользователя.

Назначение компонента SketchFlow

Как подразумевает само название компонента SketchFlow, он представляет собой набор инструментальных средств, с помощью которых группа разработчиков и заказчиков программного обеспечения может работать над черновым вариантом прототипа приложения на платформе WPF или Silverlight.

При создании проекта на платформе WPF или Silverlight средствами SketchFlow вы будете приятно удивлены, обнаружив возможность сконструировать пользовательский интерфейс теми же самыми инструментальными средствами и способами, с которыми вы ознакомились в предыдущих главах. В частности, вы можете добавлять элементы управления на монтажный стол из библиотеки ресурсов, настраивать их внешний вид на панели Properties, внедрять графику и анимацию в прототип и делать многое другое.

Особенность SketchFlow заключается в том, что этот компонент предоставляет дополнительные инструментальные средства для составления блок-схемы пользовательского интерфейса приложения и взаимодействия с ней, а также для организации перехода из одного состояния приложения в другое. Так, на панели SketchFlow Map можно быстро составить ряд компоновок пользовательского интерфейса, которые будут отображаться, как только конечный пользователь начнет выбирать пункты меню или последовательно проходить экраны мастера компоновки, щелкая на кнопках Next (Далее) и Previous (Назад). Кроме того, можно легко изменить последовательность подобных переходов, исходя из отзывов клиента.

Помимо составления блок-схем пользовательского интерфейса приложения, корпорация Microsoft намеренно разработала стандартный стиль каждого элемента управления, чтобы проект, разрабатываемый средствами SketchFlow, имел простой общий вид, ясно указывающий на то, что это действительно *прототип*, а не готовый программный продукт. Используя заготовки, выбираемые из категорий Sketch Styles (Стили эскизов), Sketch Shapes (Формы эскизов) и Mockup Controls (Имитируемые элементы управления) в библиотеке ресурсов, вы сможете сконструировать пользовательский интерфейс, который поможет пользователю сосредоточить основное внимание на общей блок-схеме и компоновке прототипа, вместо того чтобы отвлекаться на такие мелочи, как размеры шрифтов, цвета переднего и заднего плана и т.п. В качестве примера на рис. 8.1 показана блок-схема главного окна приложения на платформе WPF, построенная с помощью стилей SketchFlow и заготовок из категории Mockup Controls.

Еще одна замечательная особенность компонента SketchFlow состоит в том, что он построен таким образом, чтобы отзывы клиентов могли собираться и регистрироваться в процессе построения и демонстрации прототипа. Безусловно, для этой цели можно воспользоваться и аннотациями, составляемыми в Expression Blend (см. главу 1), но проигрыватель SketchFlow Player предоставляет более эффективные и интерактивные средства регистрации отзывов конечных пользователей. Клиенты могут опробовать многие варианты и снабдить их комментариями для группы разработчиков, аннотируя свой опыт взаимодействия с прототипом. О том, как подобные инструментальные средства действуют на практике, речь пойдет далее.

Примечание. Проигрыватель SketchFlow Player распространяется свободно и не требует установки полной версии Expression Blend на компьютере клиента. Более того, он допускает размещение прототипов как настольных приложений на платформе WPF, так и веб-ориентированных приложений на платформе Silverlight.

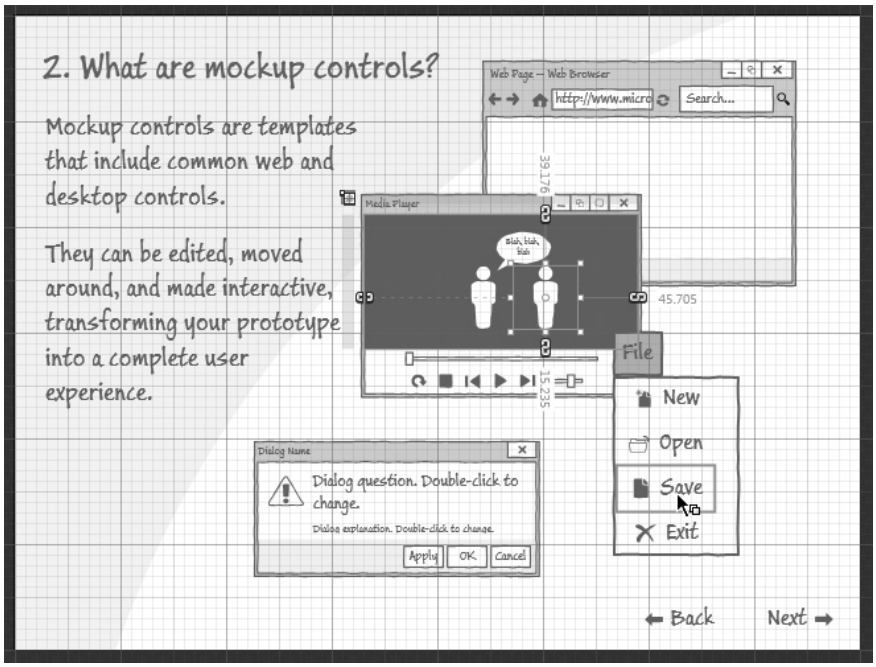


Рис. 8.1. Стили SketchFlow намеренно сделаны простыми, чтобы помочь клиенту сосредоточить свое внимание на главном, а не на мелочах

Прорабатывая материал этой главы, вы также заметите, что прототип SketchFlow может быть использован в качестве отправной точки для разработки реального приложения. Следовательно, вместо того чтобы выбрасывать прототип, можете заменить стили SketchFlow по своему усмотрению, перенести все отзывы клиентов в специальный документ Microsoft Word и начать детальную проработку проекта своего приложения. Не следует забывать, что прототипы SketchFlow — это не простые рисунки, а приложения, полностью функционирующие на платформе WPF или Silverlight. Внеся необходимые коррективы в прототип, можете преобразовать его в рабочий проект и продолжить работу над окончательным вариантом приложения.

Приложение. Следует также иметь в виду, что проигрыватель SketchFlow Player может быть настроен на автоматическую выгрузку аннотаций клиентов и разработчиков на именованный сервер SharePoint, хотя данный вопрос в этой главе не рассматривается. Такая возможность может оказаться очень полезной в том случае, если требуется сохранить важные данные в каком-то центральном месте. Более подробно об этом можно узнать из раздела “Публикация на сервере SharePoint” (Publish to SharePoint) руководства пользователя Expression Blend.

Рассмотрение SketchFlow на конкретном примере

А теперь, когда вам стали более понятными функциональные возможности компонента SketchFlow, рассмотрим его применение на конкретном примере. Но вместо создания специального прототипа воспользуемся одним из готовых примеров проектов, чтобы сделать краткий обзор основных инструментальных средств SketchFlow. Благодаря этому станут понятнее особенности работы с SketchFlow в контексте более

крупного, реального проекта. Итак, запустите Expression Blend на выполнение и выберите команду Help⇨Welcome Screen из главного меню. Затем щелкните на вкладке Samples и выберите пример проекта `PCGamingSketch` (Эскиз игрового проекта для ПК), как показано на рис. 8.2[3].

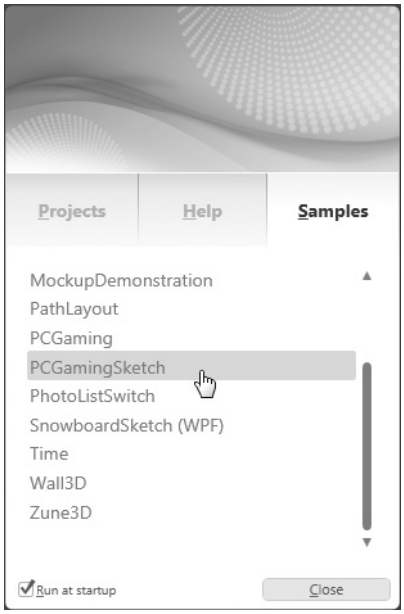


Рис. 8.2. В состав Expression Blend входит несколько примеров проектов, разработанных средствами SketchFlow

Исследование панели SketchFlow Map

После загрузки выбранного примера проекта вы непременно обнаружите новый элемент пользовательского интерфейса в среде Expression Blend IDE — панель SketchFlow Map (Составление блок-схемы эскизного проекта), пристыкованную ниже монтажного стола. На рис. 8.3 (а также на цветной вклейке) показано содержимое панели SketchFlow Map для текущего проекта `PCGamingSketch`.

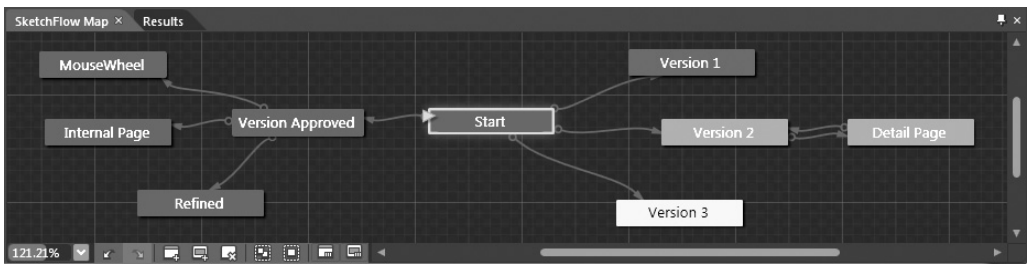


Рис. 8.3. Панель SketchFlow Map

[3] По ходу чтения этой главы рекомендуется исследовать каждый пример проекта, разработанного средствами SketchFlow и содержащего слово `Mockup` (Макет) или `Sketch` (Эскиз) в своем имени. Все эти примеры наглядно демонстрируют преимущества и эффективность данной технологии создания прототипов.

Панель SketchFlow Map выполняет роль графического редактора, в котором можно определить структуру, блок-схему, навигационные средства и составные части разрабатываемого приложения. В отличие от монтажного стола, который дает возможность уделить основное внимание содержимому пользовательского интерфейса главного окна (объекта типа `Window`) или элемента управления на начальной веб-странице (объекта типа `UserControl`), панель SketchFlow Map позволяет сосредоточиться на общей структуре приложения.

Каждый узел на панели SketchFlow Map обозначает конкретный экран в прототипе приложения, который обычно соотносится с отдельным объектом типа `Window` или `UserControl`. В рассматриваемом здесь примере проекта `PCGamingSketch` три крайних слева узла называются `MouseWheel` (Колесико мыши), `Internal Page` (Внутренняя страница), `Refined` (Уточненный вариант) и `Version Approved` (Утвержденный вариант) соответственно.

Все эти узлы обозначают ряд экранов, на которых демонстрируется окончательно утвержденный вариант первоначальных макетов графического пользовательского интерфейса приложения[4]. Если дважды щелкнуть на любом из этих узлов (или вообще на любом узле блок-схемы проекта в SketchFlow), на монтажном столе появится соответствующий экран для последующего просмотра. В качестве примера на рис. 8.4 приведен экран, представляющий узел `Version Approved`.

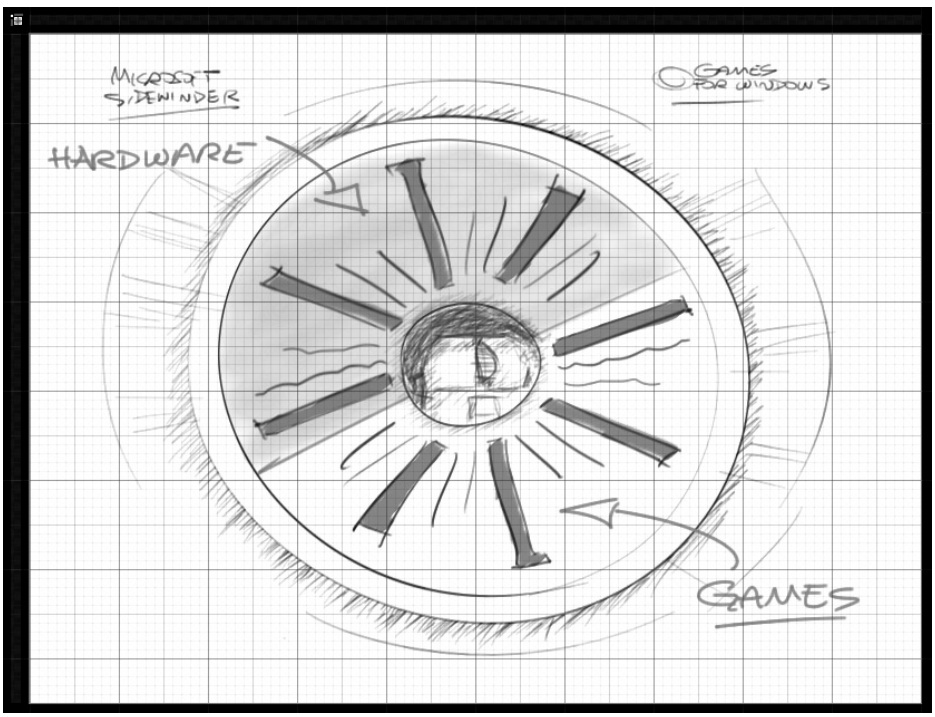


Рис. 8.4. Каждый узел блок-схемы проекта в SketchFlow обозначает отдельный экран в прототипе разрабатываемого приложения

[4] Тем не менее большинство подобных экранов составлены не с помощью элементов управления SketchFlow, а с использованием элементов управления типа `Image`, в которых содержатся файлы изображений формата PNG и JPEG, сформированные в таких специализированных графических инструментальных средствах, как Expression Design или Adobe Illustrator.

Так, если вы откроете узел `Refined` (рис. 8.5), то обнаружите формализованную компоновку пользовательского интерфейса, составленную из диспетчеров компоновки, которые обычно применяются в приложениях на платформе Silverlight (например, `Grid` и `Canvas`), а также из целого ряда элементов управления пользовательского интерфейса (типа `TextBlock`, специальных компонентов и пр.) и объектов поведения, доступных в Expression Blend.

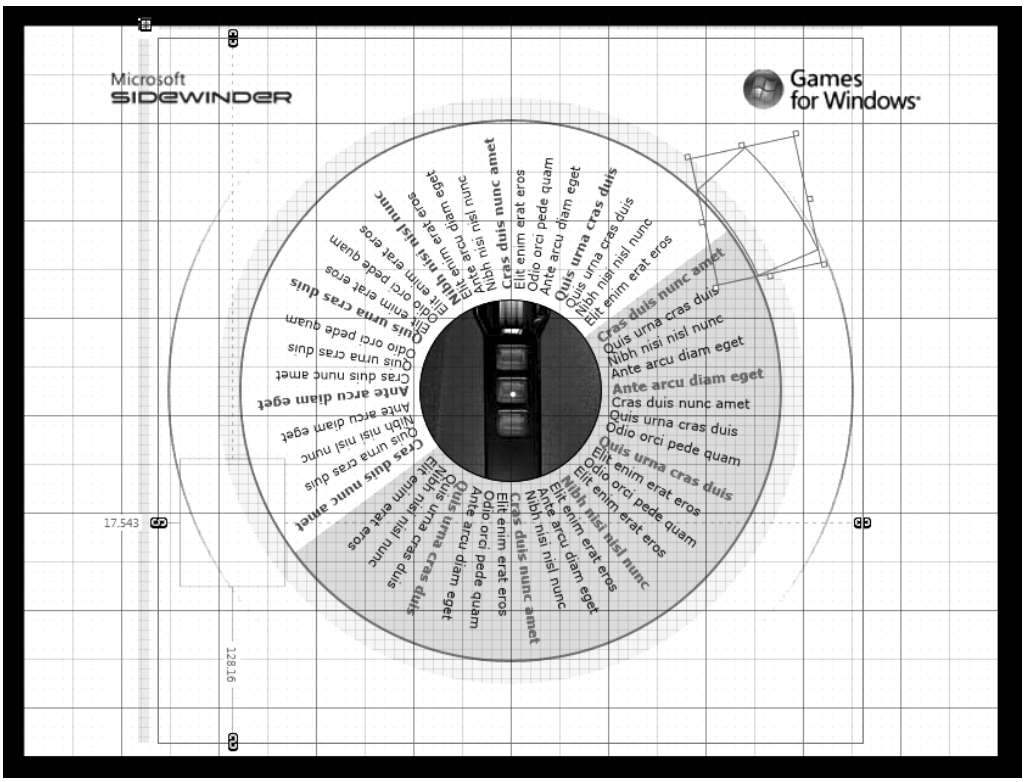


Рис. 8.5. Прототипы в SketchFlow могут содержать экраны, составленные из настоящих компонентов пользовательского интерфейса

Как показано на рис. 8.3, рассматриваемые здесь узлы связаны друг с другом направленными стрелками на панели SketchFlow Map. Так, из узла `Version Approved` исходят три стрелки, связывающие его с тремя крайними слева узлами. Этими стрелками обозначаются взаимосвязи в элементарной блок-схеме пользовательского интерфейса. Подобные взаимосвязи могут быть проверены в редакторе навигации проигрывателя SketchFlow Player (о том, как это делается, речь пойдет далее).

Вы можете также воспользоваться различными объектами навигационного поведения, чтобы быстро смоделировать реакцию на действия пользователя (щелчки на кнопках, выбор пунктов меню и пр.). Но самое замечательное, что все эти навигационные взаимосвязи можно оперативно изменять по ходу дела средствами Expression Blend. Более подробно применение редакторов навигации будет рассматриваться по мере изложения материала этой главы.

Назначение узла `Start`

Как показано на рис. 8.3, а на рис. 8.6 (и на цветной вклейке к книге) — более подробно, слева к узлу `Start` (Пуск) подходит зеленая стрелка. При создании нового прототипа приложения в SketchFlow автоматически формируется обозначаемый подобным образом первоначальный экран, который появляется после запуска прототипа приложения на выполнение.

Рис. 8.6. Узел `Start` обозначается входящей зеленой стрелкой

Если хотите заменить узел запуска, щелкните правой кнопкой мыши на любом другом узле блок-схемы прототипа и выберите команду Set as Start (Установить в качестве узла запуска) из всплывающего контекстного меню. После этого входящая зеленая стрелка переместится к выбранному узлу. В качестве упражнения назначьте узел `Version Approved` в качестве узла запуска, как показано на рис. 8.7. Но прежде чем перейти к следующему подразделу этой главы, верните узел `Start` в исходное состояние для запуска прототипа приложения.

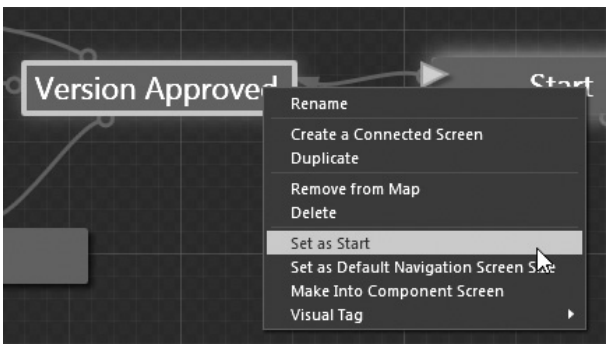


Рис. 8.7. Замена узла запуска прототипа приложения

Цветовое кодирование

Для каждого узла блок-схемы прототипа может быть назначен особый цвет, что упрощает визуальное различение ряда связанных друг с другом узлов. И хотя цвет узла не оказывает никакого влияния на поведение прототипа приложения во время его выполнения, по умолчанию голубым цветом обозначается так называемый экран *навигации* (Navigation), а зеленым цветом — экран *компонента* (Component). Как поясняется далее в главе, на экранах компонентов определяются фрагменты повторяющегося содержимого пользовательского интерфейса, например, навигационное меню, которое отображается на многих экранах навигации.

Для того чтобы изменить цвет любого узла блок-схемы прототипа, наведите на него курсор мыши, чтобы появилась всплывающая панель редактирования узла, как показано на рис. 8.8, а также на цветной вклейке. Затем выберите нужный цвет из раскрывающегося справа внизу списка доступных цветов.

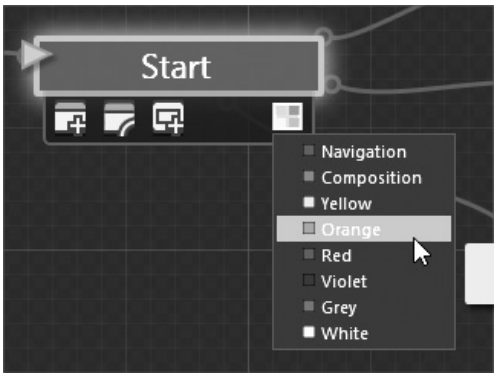


Рис. 8.8. Для каждого узла блок-схемы прототипа может быть назначен произвольный цвет

Создание и связывание узлов

В редакторе узлов имеется также возможность создать новые связываемые вместе узлы и соответствующий экран, связать текущий узел с уже существующим и сформировать новый экран компонента (подробнее об экранах компонентов речь пойдет ниже). Наведя курсор мыши на любой узел, вы увидите всплывающую панель с кнопками, для которых назначены все эти функции, как показано на рис. 8.9, а также на цветной вклейке.

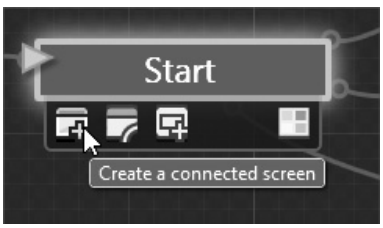


Рис. 8.9. В редакторе узлов можно создавать и связывать вместе отдельные узлы

Примечание. В SketchFlow допускается переход из одного узла в другой и обратно. В качестве примера на рис. 8.3 показано установление подобной двунаправленной связи между узлами `Version 2` и `Detail Page` на панели SketchFlow Map.

По мере добавления и связывания узлов на панели SketchFlow Map в процессе создания прототипа приложения у вас, безусловно, возникнет необходимость внести изменения в блок-схему пользовательского интерфейса. Для этого щелкните на любом конце линии, соединяющей узлы, и переместите ее в соответствующее место блок-схемы. Но делая это, имейте в виду, что кружком на одном из концов соединительной линии обозначается экран, *из* которого осуществляется переход, тогда как стрелкой — экран, *к* которому осуществляется переход.

Примечание. Для того чтобы полностью удалить соединительную линию из блок-схемы прототипа, щелкните на ней правой кнопкой мыши и выберите команду Delete из всплывающего контекстного меню.

Присваивание стилей отдельным переходам

Создав ряд связанных вместе узлов блок-схемы прототипа, можете присвоить эффекты отдельным переходам в виде стилей, чтобы придать создаваемому прототипу более привлекательный вид. Прежде чем запускать прототип данного приложения на выполнение, щелкните правой кнопкой мыши на линии, соединяющей узлы `Start` и `Version 1`, а затем выберите стиль, наиболее подходящий, на ваш взгляд, для перехода между этими узлами, как показано на рис. 8.10, а также на цветной вклейке. Повторите эти действия для трех других соединительных линий, исходящих из узла `Start`.

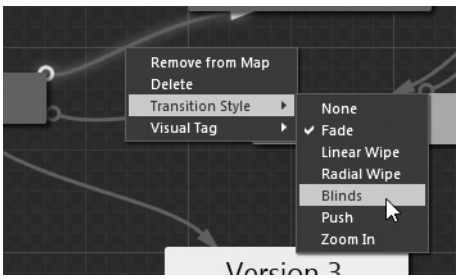


Рис. 8.10. Переходам к любым экранам навигации можно присвоить различные эффекты в виде соответствующих стилей

Проверка прототипа в проигрывателе SketchFlow Player

А теперь проверьте рассматриваемый здесь прототип приложения в проигрывателе SketchFlow Player, нажав функциональную клавишу <F5> или комбинацию клавиш <Ctrl+F5>. Прототип `PCGamingSketch` относится к приложению на платформе Silverlight, а следовательно, проигрыватель SketchFlow Player будет загружен в используемый по умолчанию браузер, в окне которого должен появиться активный экран, связанный с узлом `Start`.

Экраны навигации

Левая часть пользовательского интерфейса проигрывателя SketchFlow Player разделена на две основные области. В верхней области находится навигационная панель Navigate для отображения экранов на основании связей и переходов между узлами, созданными на панели SketchFlow Map. Напомним, что из экрана запуска `Start` можно перейти к одному из исходящих экранов: `Version 1`, `Version 2`, `Version 3` или `Approved Version`. По желанию можете также щелкнуть на кнопке Map, чтобы просмотреть доступное только для чтения представление той же самой блок-схемы, как показано на рис. 8.11.

Уделите немного времени исследованию панели Navigate, выбирая по очереди каждую из ссылок на этой панели или соответствующий узел на блок-схеме прототипа в SketchFlow.

Примечание. Можете также щелкнуть на кнопке Home (Домой), расположенной выше панели Navigate, чтобы вернуться к экрану запуска прототипа приложения на выполнение.

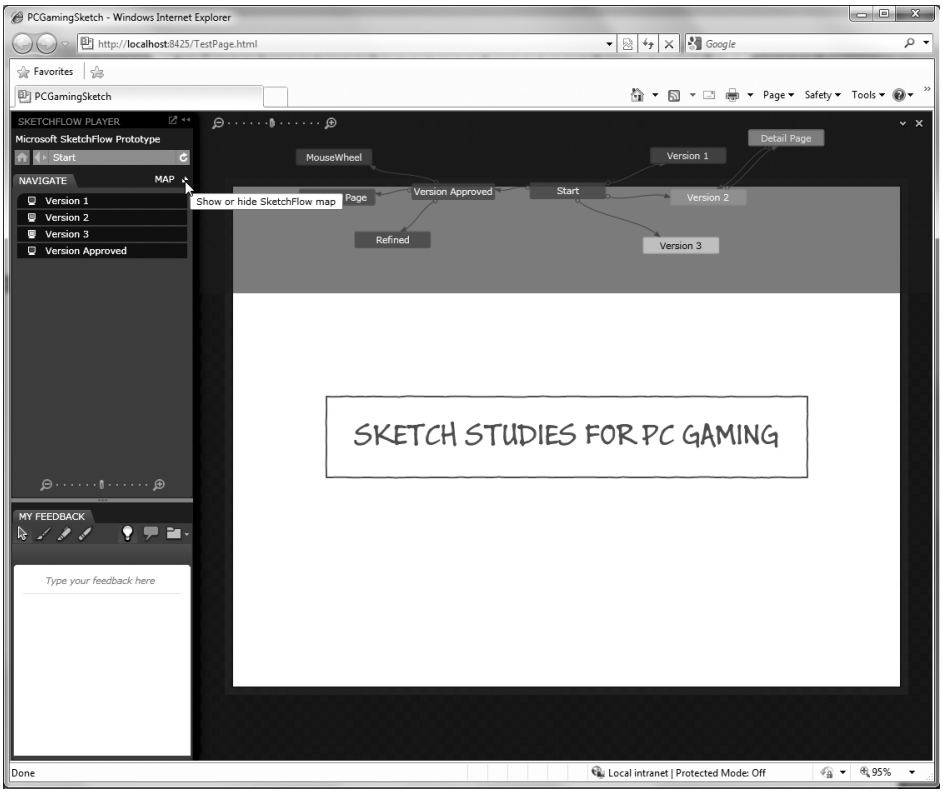


Рис. 8.11. Проигрыватель SketchFlow Player в действии

Перейдя к нужному экрану, обратите внимание на то, что на панели Navigate появляются виды поведения или анимационные последовательности переходов, которые могут быть активизированы щелчком кнопкой мыши. Подобные эффекты переходов добавлены к экрану с помощью различных объектов поведения, доступных в SketchFlow, а также специальных анимационных последовательностей, создаваемых в среде Expression Blend IDE. В качестве примера на рис. 8.12 приведены переходы, обнаруживаемые на экране `Version 2`.

Примечание. Любую анимацию или поведение можно воспроизвести, щелкнув на той части экрана, где они определены в прототипе для применения. Так, если щелкнуть на узле `Version 2` блок-схемы прототипа, то в вертикальной области `Games` (Игры) или `Hardware` (Аппаратные средства) экрана `Version 2` можно просмотреть ту же самую анимационную последовательность.

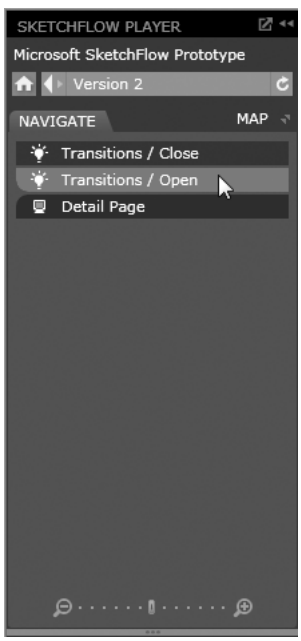


Рис. 8.12. На панели Navigate могут быть запущены экраны с разными видами поведения и анимации

Ввод отзывов в прототип

Напомним, что основное назначение проигрывателя SketchFlow Player состоит в том, чтобы предоставлять клиентам возможность снабжать прототип отзывами, которые могут оказаться полезными для дальнейшей разработки приложения. Для упрощения данной процедуры клиенты могут воспользоваться панелью My Feedback (Мой отзыв), расположенной слева внизу в окне проигрывателя SketchFlow Player, как показано на рис. 8.13, а также на цветной вклейке. На этой панели клиенту предоставляется возможность ввести текст своего отзыва вручную или же выбрать один из инструментов рисования (раскраски, выделения цветом и стирания), чтобы нарисовать графические знаки замечаний к отдельным частям экрана и снабдить их текстовыми аннотациями. После выбора инструмента раскраски или выделения цветом клиенту предоставляются также различные возможности для изменения размеров и цвета визуальной аннотации. В качестве примера на рис. 8.13 приведена крупная оранжевая стрелка, указывающая на отдельный экран прототипа, а также показана возможность пристыковывать и отстыковывать панель инструментов проигрывателя SketchFlow Player.

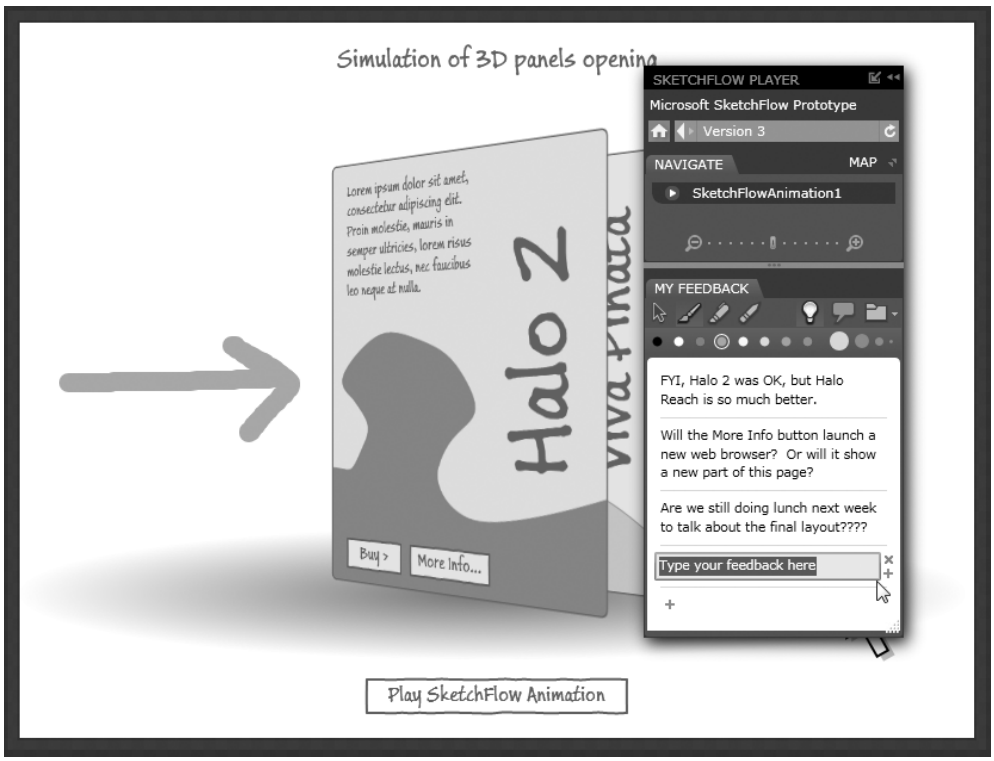


Рис. 8.13. Ввод отзывов в графической и текстовой форме в прототип

Экспорт отзывов клиентов из прототипа

Отзывы клиентов регистрируются автоматически и могут быть оставлены как в текстовой, так и в графической форме на каждом экране прототипа. По завершении ввода отзывов клиент может сохранить их, выбрав кнопку с пиктограммой папки на панели My Feedback. В качестве упражнения воспользуйтесь инструментами ввода текста и рисования, чтобы снабдить элементарными отзывами несколько экранов прототипа,

руководствуясь рис. 8.13 как образцом. По завершении выберите команду Export Feedback (Экспортировать отзыв) из всплывающего меню, как показано на рис. 8.14. В итоге ваш отзыв будет сохранен в файле `MyFeedback` на рабочем столе ОС Windows или в указанном вами месте[5].

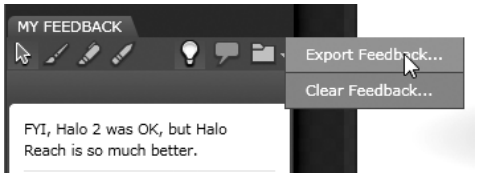


Рис. 8.14. Экспорт отзывов во внешний файл

Импорт отзывов клиентов в Expression Blend

Для просмотра отзывов клиентов в среде Expression Blend IDE необходимо сначала активизировать режим работы, выбираемый по команде меню Window⇨SketchFlow Feedback, как показано на рис. 8.15.

После этого панель SketchFlow Feedback становится активной в среде Expression Blend IDE. Для того чтобы добавить данные формата XML, предназначенные для просмотра на этой панели, а следовательно, и на монтажном столе выбранной страницы, щелкните сначала на кнопке со знаком +, а затем на кнопке Add и найдите файл с экспортированными отзывами, как показано на рис. 8.16.

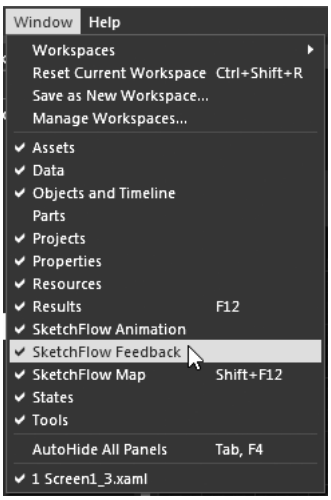


Рис. 8.15. Отзывы клиентов могут быть просмотрены в Expression после активизации соответствующего режима работы

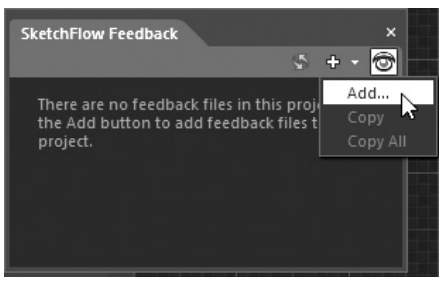


Рис. 8.16. Отзывы из файлов с расширением `*.feedback` могут быть добавлены на панели SketchFlow Feedback

На монтажном столе появится визуализированный отзыв, а на панели SketchFlow Map — пиктограмма лампочки рядом с каждым экраном, содержащим отзывы клиента. В качестве примера на рис. 8.17 показано, что текстовые аннотации клиента могут быть просмотрены на панели SketchFlow Feedback, пристыкованной к левому краю рабочего окна Expression Blend.

[5] Файлы отзывов к прототипам сохраняются с расширением `*.feedback`. И нет ничего удивительного в том, что такой файл содержит описание данных в формате XML.

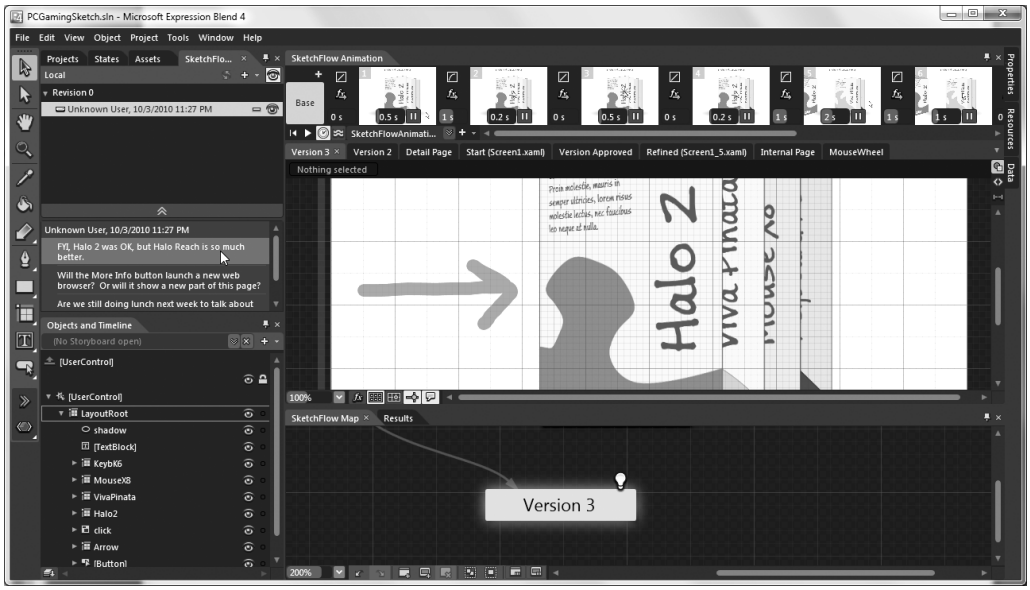


Рис. 8.17. Просмотр отзывов клиента в проекте, разрабатываемом в Expression Blend

Составление проектной документации в формате Microsoft Word

Прежде чем перейти непосредственно к процессу создания прототипов сызнова, следует отметить, что в среде Expression Blend SketchFlow IDE предоставляется также возможность составлять проектную документацию в формате текстового редактора Microsoft Word, фиксируя в ней каждый экран и аннотации клиентов к прототипу. Нетрудно догадаться, что такая документация может послужить в качестве первоначального образца для рабочей документации на программный продукт. Хотя для составления такой документации недостаточно выбрать только команду меню File⇨Export to Microsoft Word (Файл⇨Экспорт в Microsoft Word), как показано на рис. 8.18.

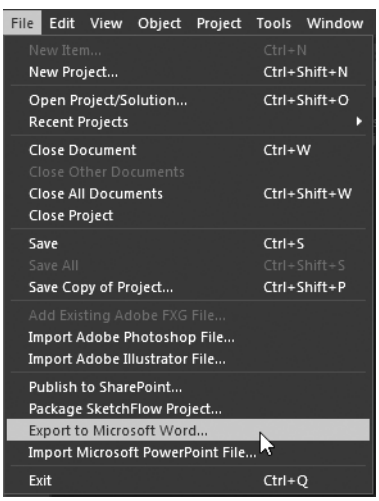


Рис. 8.18. Экспорт прототипа из SketchFlow в Microsoft Word

В состав SketchFlow входит собственный шаблон Microsoft Word. Но как показано на рис. 8.19, вы вольны указать специальный шаблон, а также настроить несколько дополнительных параметров, и самым главным из них является включение отзывов клиента в документацию.

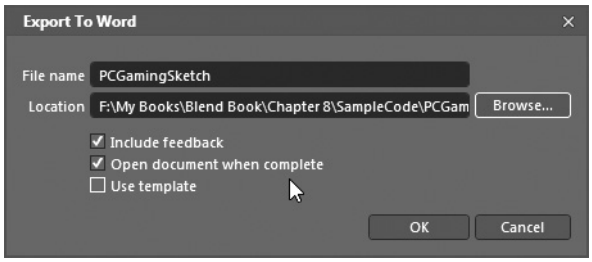


Рис. 8.19. Настройка параметров экспорта данных в Microsoft Word

После экспорта данных в документ Microsoft Word вы вольны дополнить и расширить его так, как считаете нужным.

Исходный код. Исходный код примера проекта `PCGamingSketch` находится в папке `Ch 8 Code` загружаемого архива примеров проектов к данной книге.

Создание прототипа приложения на платформе Silverlight

А теперь перейдем непосредственно к созданию в SketchFlow прототипа приложения на платформе Silverlight, моделирующего ряд типичных экранов, которые можно нередко встретить в приложениях электронной коммерции. С этой целью создайте новый проект по шаблону `Silverlight SketchFlow Application`, присвоив ему имя `OnlineStoreApp` в диалоговом окне New Project, как показано на рис. 8.20.

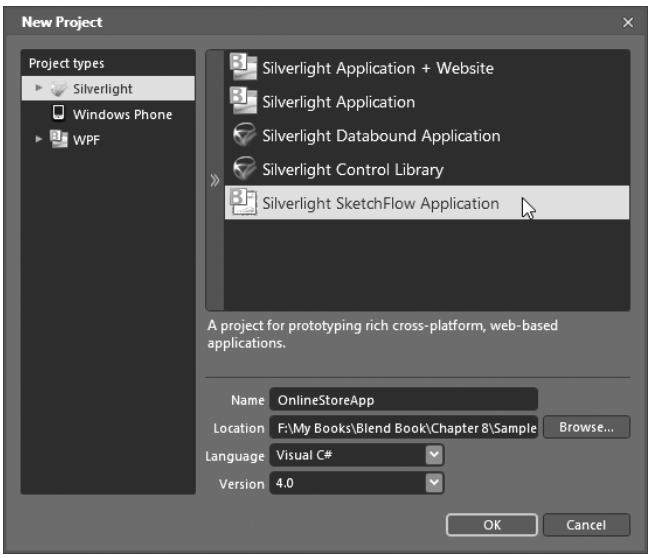


Рис. 8.20. Создание нового проекта приложения по шаблону `Silverlight SketchFlow Application`

После создания нового проекта появится единственный экран `Screen 1`, составленный как начальный экран нового прототипа. Перейдите к панели SketchFlow Map и переименуйте этот начальный экран на `MainScreen` (главный экран), щелкнув на его узле правой кнопкой мыши в блок-схеме прототипа и выбрав команду Rename из всплывающего контекстного меню, как показано на рис. 8.21.

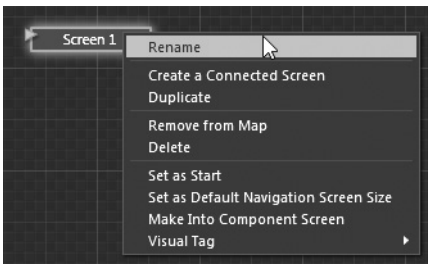


Рис. 8.21. Выбор команды переименования узлов отдельных экранов

Исследование файлов проекта

Организация проекта приложения в SketchFlow немного отличается от организации проекта типичного приложения на платформе WPF или Silverlight. Поэтому перейдите к панели Projects, обратив внимание на то, что текущее решение содержит два связанных вместе проекта, как показано на рис. 8.22. Первый проект `OnlineStoreApp` содержит общую логику приложения в файлах разметки `App.xaml` и исходного кода `App.xaml.cs`. Кроме того, в этом проекте имеются ссылки на ряд сборок Expression Blend SketchFlow с префиксом `Microsoft.Expression` в их именах.

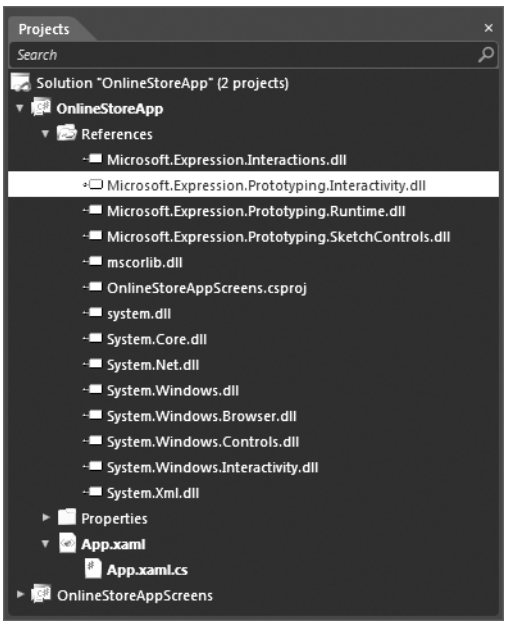


Рис. 8.22. Первый проект содержит общую логику приложения

Во втором проекте `OnlineStoreAppScreens` содержатся ссылки на те же самые сборки, предназначенные для создания прототипов. Именно в нем и будет размещаться каждый экран, добавляемый в приложение на панели SketchFlow Map. В настоящий момент в нем присутствует только экран `MainScreen`. Кроме того, в данном проекте будет определен ряд дополнительных компонентов, представляющих особый интерес. К их числу относятся следующие компоненты.

- Файл разметки `SketchStyles.xaml`. Этот XAML-файл содержит стили для оформления элементов пользовательского интерфейса прототипа приложения в SketchFlow.
- Файл `Sketch.Flow`. Этот документ содержит данные, считываемые для отображения на панели SketchFlow Map узлов и линий, которые их соединяют. Обычно этот файл не правится, поскольку он обновляется автоматически во время работы в среде Expression Blend SketchFlow IDE.
- *Папка со шрифтами*. В ней хранится несколько типов шрифтов, применяемых в стилях оформления, входящих в состав SketchFlow.

Структура второго проекта приведена на рис. 8.23.

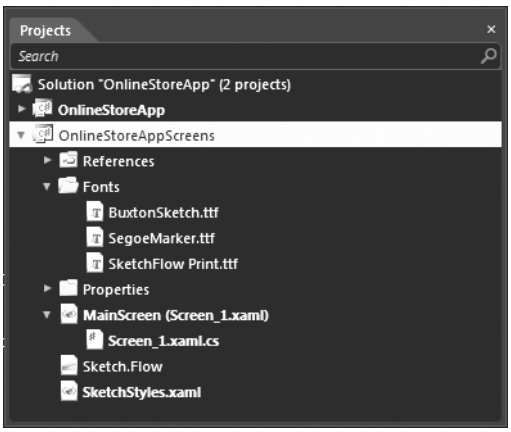


Рис. 8.23. Второй проект содержит экраны, стили, шрифты и данные для составления блок-схемы прототипа

Стили оформления в SketchFlow будут применяться автоматически при выборе элементов управления созданием прототипов из библиотеки ресурсов. Но для особо любопытных сообщим — можно просмотреть каждый из этих стилей на панели Resources. Как пояснялось в главе 2, можно щелкнуть на любом из этих ресурсов, чтобы поправить ли видоизменить их внешний вид, хотя в рассматриваемом здесь примере предполагается, что вам не придется этого делать.

Создание экрана компонента

При разработке приложения на платформе WPF или Silverlight обычно принято создавать систему навигации, общую для ряда связанных с ней объектов типа `Window` или `UserControl`. Такая система навигации может представлять собой традиционную систему меню, специальную схему навигации, состоящую из оживляемых графических элементов, древовидную или иную систему навигации.

На *экранах компонентов* в SketchFlow можно определить схему навигации, чтобы связать ее с любым экраном, которому для навигации требуется соответствующий пользовательский интерфейс. По своей внутренней структуре экран компонента представляет собой объект типа `UserControl` (в проектах на обеих платформах WPF и Silverlight), добавляемый в диспетчер компоновки того элемента пользовательского интерфейса, в котором его предполагается использовать. Это, конечно, удобно, поскольку для изменения внешнего вида системы навигации достаточно лишь внести коррективы в пользовательский интерфейс экрана компонента, а остальные узлы будут обновлены при последующем построении проекта.

Примечание. Применение экранов компонентов не ограничивается только навигационными системами. Экраны компонентов можно также использовать для моделирования любого повторяющегося содержимого пользовательского интерфейса, в том числе колонтитулов у нижнего края окна, общепринятых элементов управления вводом информации и т.д.

Прежде всего наведите курсор мыши на узел `MainScreen`, чтобы открыть всплывающую панель инструментов, а затем щелкните на кнопке Create and insert a Component screen (Создать и ввести экран компонента), как показано на рис. 8.24.

Сделав это, переименуйте новый узел в `NavSystem` (система навигации) и перетащите его в сторону от узла `MainScreen` для удобства просмотра. Обратите внимание на то, что узлы экранов компонентов выделяются по умолчанию зеленым цветом и соединяются зеленой стрелкой, направленной *от* узла экрана компонента *к* использующему его экрану, как показано на рис. 8.25, а также на цветной вклейке.

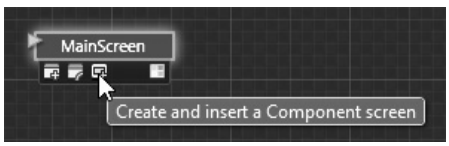


Рис. 8.24. Создание нового экрана компонента

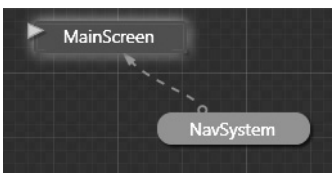


Рис. 8.25. Экран компонента, связанного с главным экраном прототипа приложения

Если вы перейдете к монтажному столу узла `MainScreen`, то обнаружите на нем систему навигации. А если проанализируете приведенную ниже исходную разметку в коде XAML, то заметите, что объект типа `UserControl`, представляющий эту систему, введен в корневой диспетчер компоновки, хотя эта разметка может оказаться у вас несколько иной в зависимости от расположения элементов на экране.

```
<Grid x:Name="LayoutRoot" Background="White">
  <local:NavSystem HorizontalAlignment="Left" VerticalAlignment="Top"
      d:IsPrototypingComposition="True" Margin="83,8,0,0"/>
</Grid>
```

А теперь активизируйте экран `NavSystem` в интегрированной среде разработки и найдите раздел Sketch Styles в категории Styles библиотеки ресурсов. Здесь вы обнаружите элементы управления созданием прототипов, которые обычно используются в проектах, разрабатываемых в SketchFlow. Вы, конечно, вольны использовать какой угодно элемент управления в прототипе своего приложения, но не забывайте, что главное преимущество этих специально стилизованных для SketchFlow элементов управления заключается в том, что они явно дают клиенту понять, что он имеет дело с *прототипом*, а не с готовым к эксплуатации приложением. На рис. 8.26 приведены эти специально стилизованные для SketchFlow элементы управления. Для них характерно наличие в их именах суффикса `Sketch`.

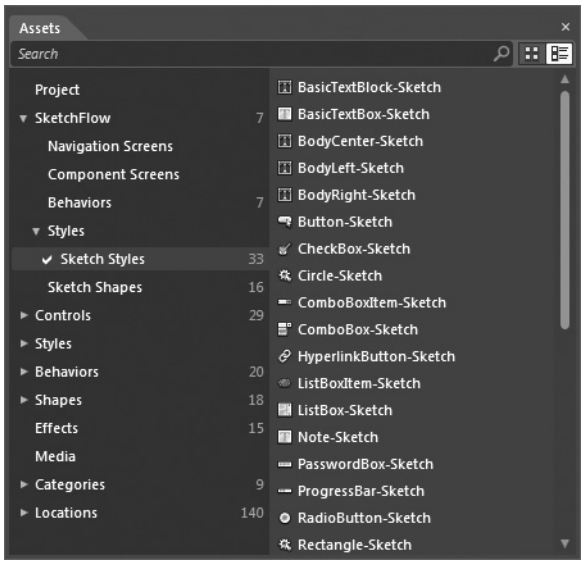


Рис. 8.26. Элементы управления, доступные в разделе Sketch Styles библиотеки ресурсов

Обратите внимание на то, что в категории Styles библиотеки ресурсов имеется также целый ряд стилизованных форм (эллипса, прямоугольника и т.д.), в которых соблюдается подобный внешний вид прототипа. А поскольку эти стили задаются по умолчанию[6], то они применяются автоматически при выборе соответствующего элемента на панели Tools. Принимая во внимание все сказанное выше, воспользуйтесь панелью Tools (или библиотекой ресурсов) для построения на экране `NavSystem` системы навигации, состоящей их трех заполненных разным цветом произвольных геометрических форм. В качестве примера на рис. 8.27 (а также на цветной вклейке) приведен вариант компоновки системы навигации из геометрических форм звезды и элементов управления типа `TextBox`, выполняющих исключительно информативные функции.

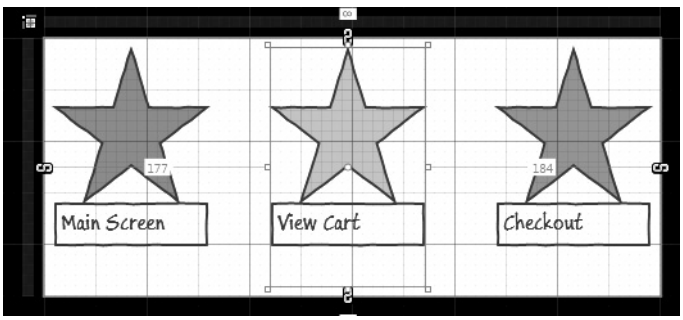


Рис. 8.27. Простая система навигации

Цель, преследуемая этой системой навигации, состоит в том, чтобы предоставить пользователю возможность переходить к трем разным экранам, производя щелчок на соответствующей форме звезды. Обработку событий от мыши в подобной системе навигации можно, конечно, организовать в специально написанном вручную коде, но в

[6] Подробнее о том, как стили задаются по умолчанию, см. в главе 5.

среде Expression Blend IDE имеется возможность автоматизировать этот процесс с помощью различных объектов поведения в SketchFlow. Прежде чем перейти к построению дополнительных экранов, завершите компоновку экрана `MainScreen`, введя простой описательный текст этого экрана в поле элемента управления типа `TextBlock`. В качестве примера на рис. 8.28 (а также на цветной вклейке) показано обозначение главного экрана текстовой надписью `Main Screen`.

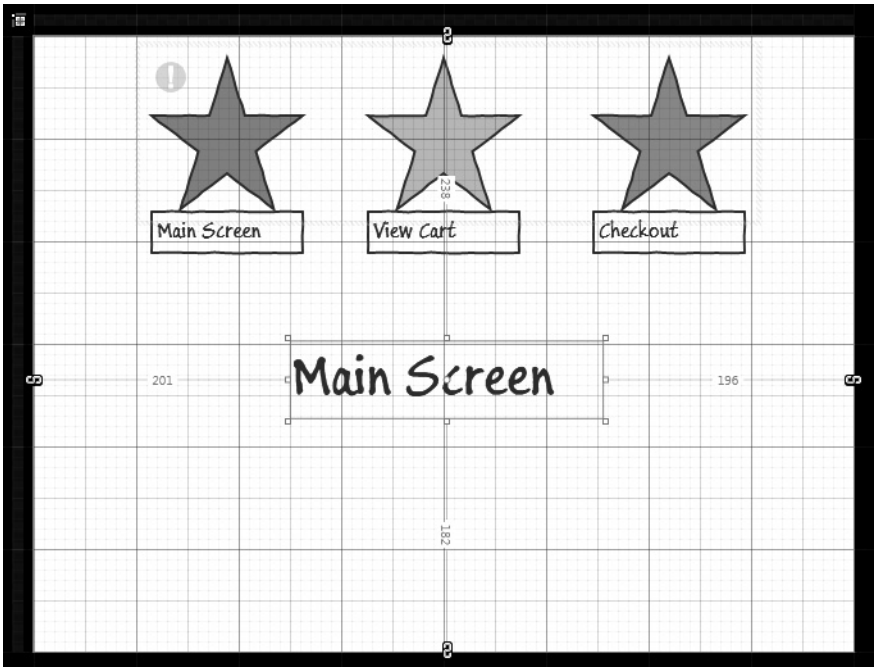


Рис. 8.28. Завершенная компоновка главного экрана

Создание дополнительных экранов

Перейдя к панели SketchFlow Map, сформируйте два новых экрана, щелкнув правой кнопкой мыши на любом пустом участке и выбрав команду Create a Screen (Создать экран) из всплывающего контекстного меню, как показано на рис. 8.29. По завершении переименуйте эти экраны в `ViewCart` (Просмотр содержимого тележки для покупок) и `Checkout` (Подсчет сделанных покупок).

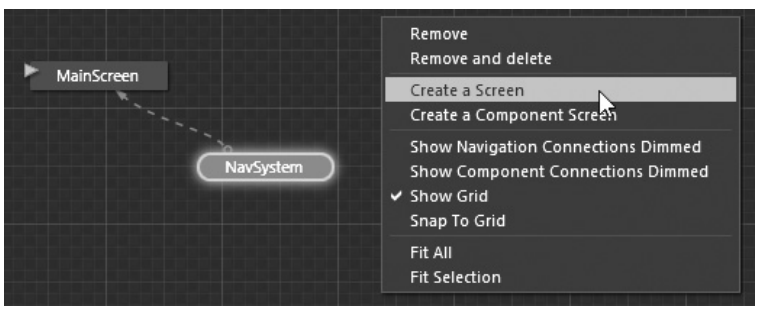


Рис. 8.29. Два несвязанных пока еще экрана

Введите на экране `ViewCart` текст, описывающий назначение этого экрана, графический элемент, например, простой серый квадрат, заменяющий тележку для покупок, а также кнопку, представленную элементом управления типа `Button`. На рис. 8.30 приведен один из возможных вариантов компоновки экрана `ViewCart`.

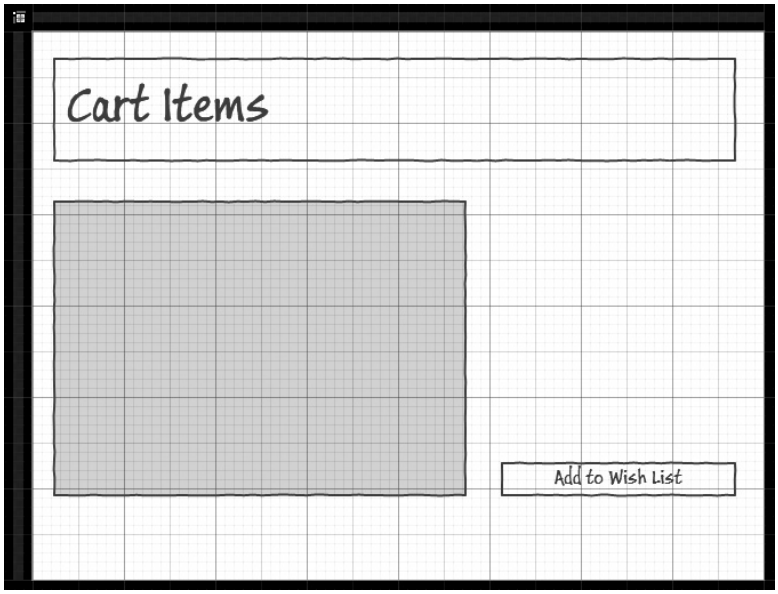


Рис. 8.30. Компоновка экрана `ViewCart`

А на рис. 8.31 приведен один из возможных вариантов столь же простой компоновки экрана `Checkout`.

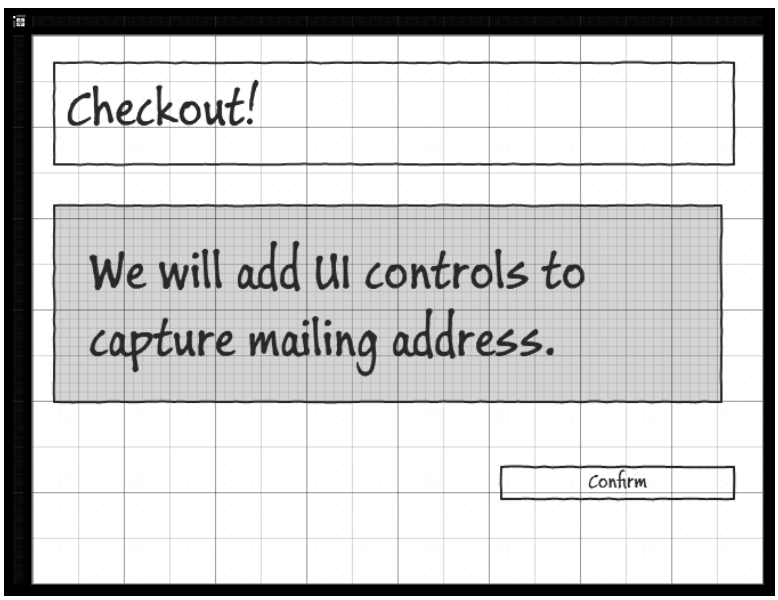


Рис. 8.31. Компоновка экрана `Checkout`

Воспроизведение навигационной системы графического пользовательского интерфейса

Теперь, когда экраны `ViewCart` и `Checkout` скомпонованы, можно ввести элементы навигации графического пользовательского интерфейса в диспетчер компоновки каждого из этих экранов, еще раз воспользовавшись панелью SketchFlow Map. В частности, воспользуйтесь мышью, чтобы перетащить *весь* узел экрана компонента на тот экран, где он будет использоваться, связав их друг с другом. По завершении блок-схема прототипа вашего приложения должна выглядеть таким же образом, как и на рис. 8.32.

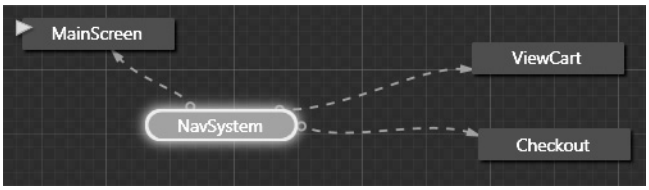


Рис. 8.32. Обновленная блок-схема прототипа

Примечание. После того как вы снабдили экраны `ViewCart` и `Checkout` системой навигации, вам, конечно, ничто не мешает изменить размеры и расположение их компонентов по своему усмотрению.

Применение объекта поведения типа `NavigateToScreenAction`

А теперь необходимо организовать появление нужного экрана, когда пользователь щелкает кнопкой мыши на выбранной геометрической форме (в данном случае — звезды). Сделать это проще всего, опираясь на особые контекстные меню, доступные на монтажном столе. С этой целью активизируйте монтажный стол экрана компонента `NavSystem` и щелкните правой кнопкой мыши на той геометрической форме звезды, которая должна переносить пользователя на главный экран. Выберите из всплывающего контекстного меню команду Navigate to⇨MainScreen (Перейти⇨На главный экран), как показано на рис. 8.33. Повторите эти же действия для остальных геометрических форм и связанных с ними экранов.

Что же предпринято в среде Expression Blend IDE в ответ на эти действия? Если вы внимательно проанализируете разметку, автоматически сформированную в коде XAML для описания экрана компонента `NavSystem`, то заметите, что вся логика навигации реализуется объектом поведения типа `NavigateToScreenAction` в SketchFlow. Ниже приведен фрагмент разметки, описывающей логику поведения первой геометрической формы звезды.

```
<ed:RegularPolygon ed:GeometryEffect.GeometryEffect="Sketch"
   InnerRadius="0.47211" PointCount="5" Stretch="Fill"
   Stroke="{StaticResource BaseBorder-Sketch}"
   StrokeThickness="2"
   UseLayoutRounding="False" Height="120" Fill="#FFCE8585">
  <i:Interaction.Triggers>
    <i:EventTrigger EventName="MouseLeftButtonDown">
      <pi:NavigateToScreenAction
         TargetScreen="OnlineStoreAppScreens.Screen_1"/>
    </i:EventTrigger>
  </i:Interaction.Triggers>
</ed:RegularPolygon>
```

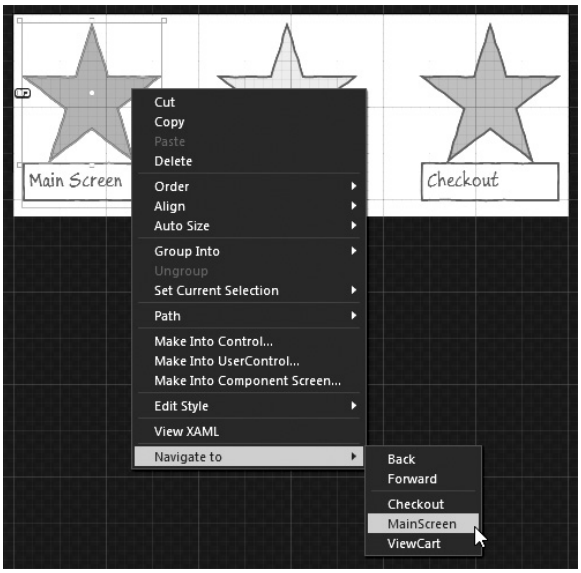


Рис. 8.33. Связывание элементов пользовательского интерфейса с соответствующим навигационным действием

Для выполнения навигационных действий можете связать отдельные элементы пользовательского интерфейса и вручную, выбрав соответствующий объект поведения из раздела Behaviors в категории SketchFlow библиотеки ресурсов, а затем настроив его на панели Properties.

Аналогично любым другим элементам, объекты поведения можно переместить с панели Objects and Timeline на монтажный стол и затем изменить их исходные установки. В качестве примера на рис. 8.34 показана исходная настройка объекта поведения типа `NavigateToScreenAction`. Как видите, событие `MouseLeftButtonDown`, наступающее при нажатии левой кнопки мыши на выбранном элементе, инициируется триггером, но вы вольны изменить такую логику поведения по своему усмотрению.

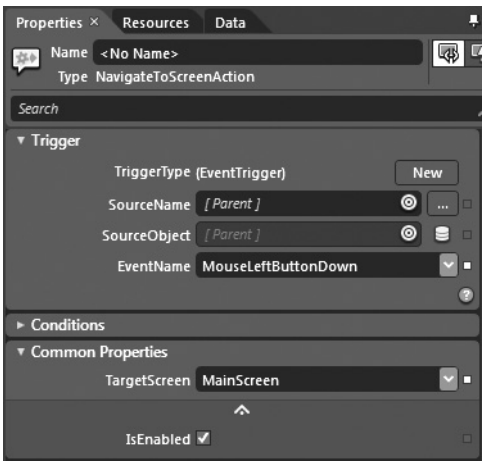


Рис. 8.34. Любой объект поведения в SketchFlow можно настроить на панели Properties

Примечание. Более подробно с каждым объектом поведения в SketchFlow можно ознакомиться в руководстве пользователя Expression Blend. Нет ничего удивительного в том, что объекты поведения действуют в SketchFlow аналогично, а то и совершенно одинаково с упоминавшимися ранее объектами поведения в Expression Blend.

Так или иначе, на данной стадии работы над прототипом приложения в его блок-схеме на панели SketchFlow Map должен быть отражен не только тот факт, что на каждом из экранов используется экран компонента, но и навигационные связи между ними, обозначаемые соединительными линиями, как показано на рис. 8.35, а также на цветной вклейке.

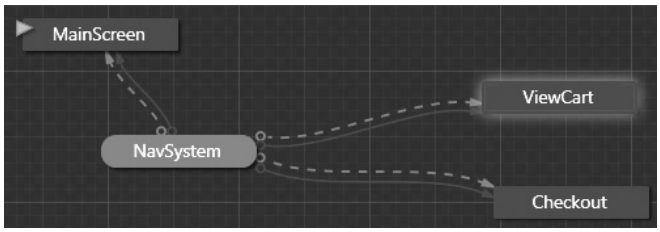


Рис. 8.35. Окончательный вид блок-схемы прототипа

Запустите прототип своего приложения на выполнение, нажав функциональную клавишу <F5>. Теперь вы сможете переходить от одного экрана к другому, пользуясь панелью Navigate проигрывателя SketchFlow Player и щелкая кнопкой мыши на выбранной геометрической форме (в данном случае — звезды), как показано на рис. 8.36, а также на цветной вклейке.

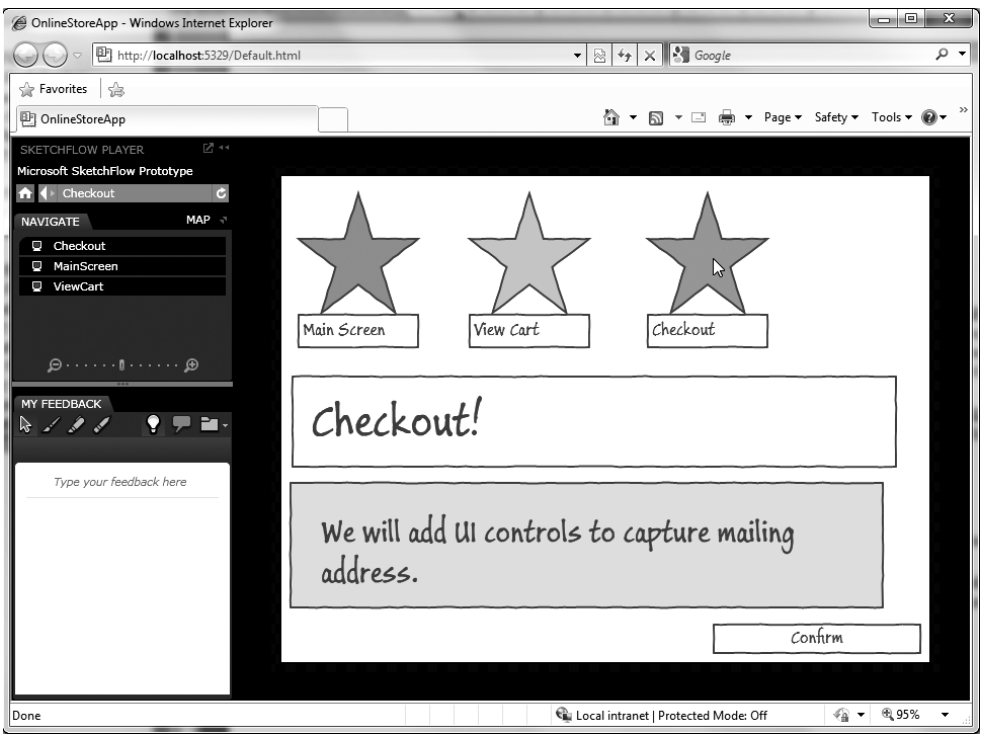


Рис. 8.36. Прототип приложения в действии

Внедрение интерактивных средств в прототип

В среде SketchFlow IDE доступен упрощенный вариант интегрированного в Expression Blend редактора анимации (см. главу 3), с помощью которого можно внедрить интерактивные средства в отдельный прототип, созданный в SketchFlow. Конечно, для этой цели можно воспользоваться и полнофункциональным редактором анимации, но панель SketchFlow Animation настолько интегрирована в среду создания прототипов в SketchFlow, что пользоваться ею удобнее.

В качестве примера введите ряд интерактивных средств на экране `Checkout`. Для этого активизируйте монтажный стол экрана `Checkout` и добавьте форму `Block Arrow Up - Sketch`, которая обозначает, хотя и в несколько преувеличенном виде, курсор мыши, наводимый пользователем (рис. 8.37).

Перейдите к панели SketchFlow Animation, которая по умолчанию располагается в верхней части визуального конструктора (рис. 8.38).

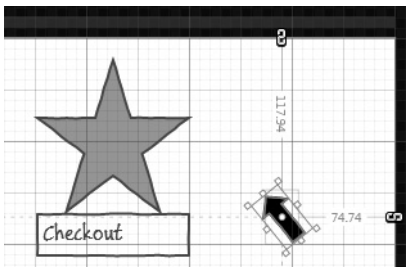


Рис. 8.37. Добавление формы, имитирующей курсор мыши

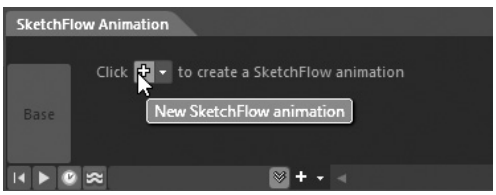


Рис. 8.38. Вид панели SketchFlow Animation

Щелкните на кнопке со знаком + (см. рис. 8.38), чтобы создать новую раскадровку анимации в SketchFlow. После этого вы вольны переименовать эту раскадровку, присвоив ей более подходящее описательное имя (например, `AnimateArrow` — анимация стрелки), выбрав соответствующий вариант из раскрывающегося списка в нижней части панели SketchFlow Animation.

Так или иначе, анимация на панели SketchFlow Animation, как и в полнофункциональном редакторе анимации (см. главу 3), организуется по принципу изменения свойств элементов пользовательского интерфейса в промежутках между ключевыми кадрами. Поэтому для ввода новых ключевых кадров в анимационную последовательность выберите отдельный ключевой кадр на панели SketchFlow Animation и щелкните на кнопке со знаком + в правом верхнем углу всплывающей панели выбранного ключевого кадра (рис. 8.39). А для того чтобы удалить выбранный ключевой кадр, достаточно щелкнуть на кнопке со знаком минус (–).

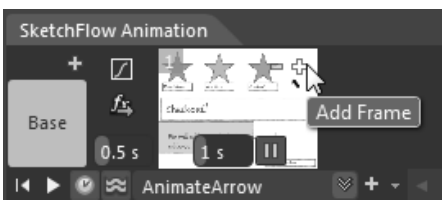


Рис. 8.39. Ввод ключевых кадров в анимацию

Введите три ключевых кадра анимации в текущую раскадровку. Самое главное, что можно выбрать любой ключевой кадр на панели SketchFlow Animation и внести изменения в раскадровку по своему усмотрению. При внесении изменений на монтажном столе

выбранного ключевого кадра данные анимации будут регистрироваться автоматически, как и в полнофункциональном редакторе анимации.

В качестве примера на рис. 8.40 показана анимационная последовательность после ввода трех ключевых кадров. В первом ключевом кадре фиксируется первоначальное состояние экрана со стрелкой, имитирующей курсор мыши в исходном положении. Во втором ключевом кадре стрелка, имитирующая курсор мыши, перемещается на кнопку `Confirm` (Подтвердить). А в третьем и последнем ключевом кадре изменяется цвета фона кнопки `Confirm`, указывая на то, что эта кнопка выбрана.

Рис. 8.40. Анимация перемещения по экрану стрелки, имитирующей курсор мыши

По желанию можете сделать еще более изящной анимацию в прототипе своего приложения, применив в отдельных ключевых кадрах эффекты инерционности движения (отскакивания, упругости и пр.). Для этого достаточно щелкнуть на кнопке с пиктограммой, обозначающей данную разновидность эффектов (рис. 8.41), а затем настроить выбранный эффект по своему усмотрению.

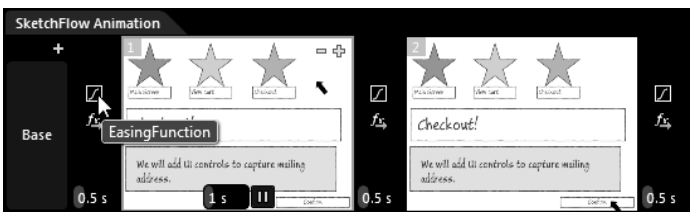


Рис. 8.41. В каждый кадр анимации можно ввести настраиваемые эффекты инерционности движения, используя соответствующие функции

А если хотите добавить эффект перехода (наплывом, волной и т.д.) в отдельный кадр, щелкните на кнопке `fx` (рис. 8.42).

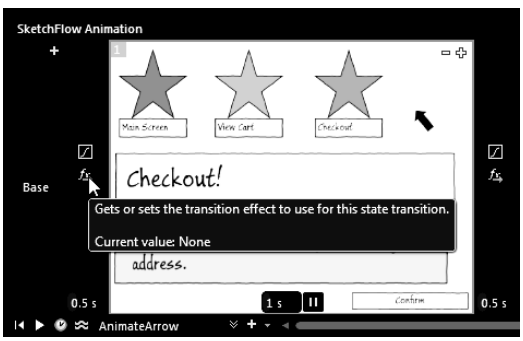


Рис. 8.42. В каждый кадр анимации можно также ввести настраиваемые эффекты переходов

Применение объекта поведения типа `PlaySketchFlowAnimationAction`

Настроив различные эффекты анимации, еще раз запустите прототип своего приложения на выполнение. Если вы затем перейдете к экрану `Checkout`, то сможете воспроизвести анимацию на панели Navigate, как показано на рис. 8.43.

Рис. 8.43. Воспроизведение анимации в прототипе

Если же хотите организовать запуск этой (или любой другой) анимации в зависимости от информации, вводимой пользователем, то воспользуйтесь объектом поведения типа `PlaySketchFlowAnimationAction`, доступном в разделе SketchFlow⇨Behaviors библиотеки ресурсов. Выберите этот объект и перетащите его на панель Objects and Timeline, опустив на самом верхнем узле `[UserControl]` иерархического представления экрана `Checkout`, как показано на рис. 8.44.

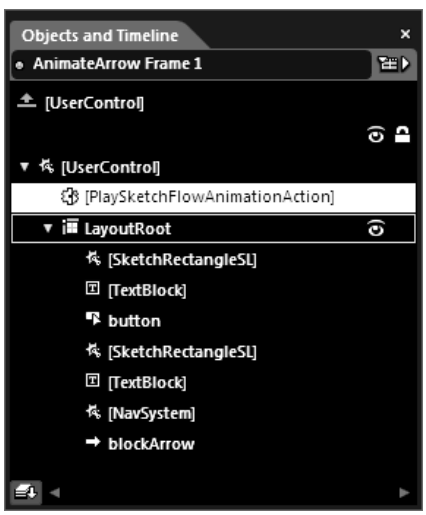


Рис. 8.44. Ввод объекта поведения типа `PlaySketchFlowAnimationAction` в иерархическое представление экрана `Checkout`

Выберите этот объект поведения, перейдите к панели Properties и настройте триггер на инициирование события `Loaded`, запускающего интересующую вас анимацию

(рис. 8.45). Если вы запустите снова прототип своего приложения и перейдете к экрану `Checkout`, то заметите, что анимация на этом экране запускается автоматически.

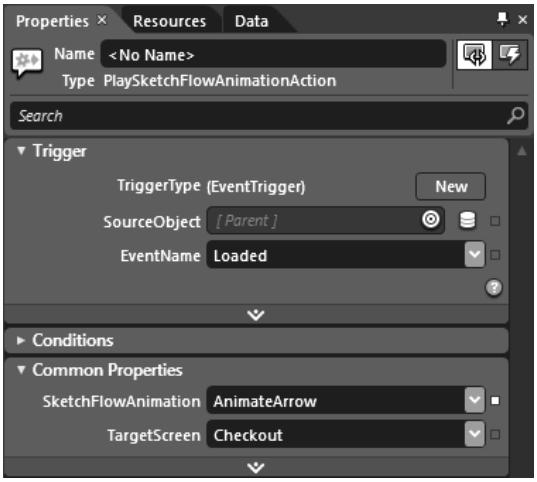


Рис. 8.45. Настройка объекта поведения типа `PlaySketchFlowAnimationAction`

Вот, собственно, и все! На этом рассмотрение простого примера создания прототипа, демонстрирующего применение различных средств SketchFlow, завершается. И хотя этот пример демонстрирует намного более простую ситуацию, чем в реальном проекте приложения, он дает ясное представление об основных стадиях процесса создания прототипов средствами SketchFlow, включая фиксацию комментариев клиентов, экспорт их комментариев в специальный документ формата Microsoft Word и составление блок-схемы прототипа на панели SketchFlow Map.

Примечание. В качестве упражнения попробуйте сами ввести комментарии клиента в прототип своего приложения, как было показано при рассмотрении примера проекта `PCGamingSketch` ранее в главе. Попробуйте также сформировать документ формата Microsoft Word из прототипа своего приложения, руководствуясь пояснениями, приведенными выше.

В заключение этой главы будут рассмотрены следующие вопросы.

- Оформление прототипа в отдельный пакет для автономного применения проигрывателя SketchFlow Player.
- Применение прототипа в качестве отправной точки для разработки приложения промышленного образца.

Исходный код. Исходный код примера проекта `OnelineStoreApp` находится в папке `Ch 8 Code` загружаемого архива примеров проектов к данной книге.

Оформление прототипа в отдельный пакет

Прорабатывая материал данной главы, вы могли не раз убедиться в том, что при запуске в среде Expression Blend IDE прототипа приложения, созданного средствами SketchFlow, автоматически запускается и проигрыватель SketchFlow Player. Но что, если требуется предоставить прототип заказчику, у которого *отсутствует* копия Expression

Blend? К счастью, имеется возможность создать по команде меню File⇨Package SketchFlow Project (Файл⇨Оформить проект SketchFlow в отдельный пакет) автономный вариант данного проигрывателя, чтобы затем отправить его вместе с прототипом на просмотр заказчику (рис. 8.46).

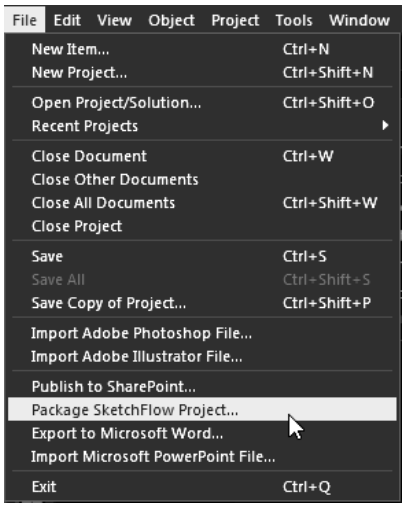


Рис. 8.46. Оформление прототипа в отдельный пакет для автономного применения проигрывателя SketchFlow Player

После выбора упомянутой выше команды меню вам останется лишь указать место и имя автоматически формируемого проигрывателя. А поскольку у вас уже имеется прототип приложения, созданный средствами SketchFlow на платформе Silverlight, то с помощью данной команды будет в итоге сформирован следующий ряд файлов: файл разметки в формате HTML, файл двоичного кода в формате XAP и файл веб-страницы в формате ASP.NET. Получив все эти файлы, заказчик может дважды щелкнуть на HTML-файле, чтобы просмотреть прототип в избранном браузере (рис. 8.47).

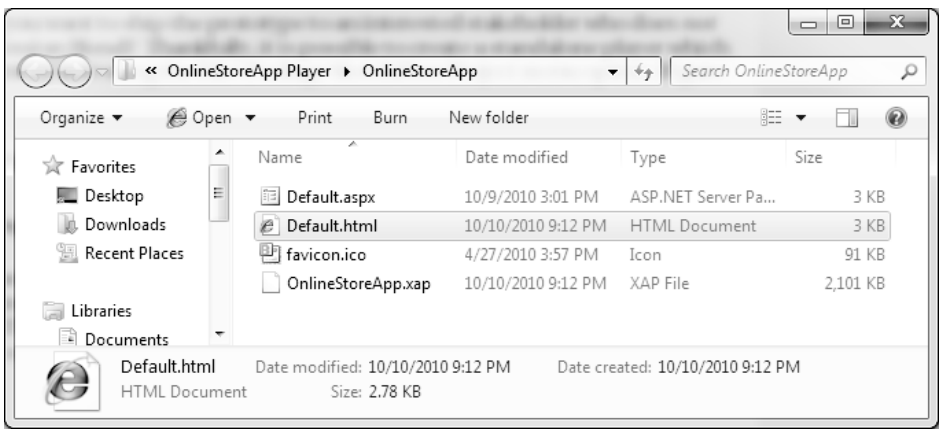


Рис. 8.47. Ряд файлов, сформированных для просмотра прототипа приложения в автономном режиме

Примечание. Если бы вы оформляли в отдельный пакет прототип приложения, созданный средствами SketchFlow на платформе WPF, то в конечном итоге получили бы исполняемый файл с расширением `*.exe`.

Перенос прототипа на почву реального проекта

Собрав и задокументировав все отзывы клиента, а затем видоизменив соответственно прототип своего приложения и утвердив его у заказчика, можете преобразовать прототип, созданный средствами SketchFlow, в реальный проект на платформе Silverlight или WPF. С этой целью сохраните, прежде всего, копию текущего прототипа, выбрав команду меню File⇨Save Copy of Project (Файл⇨Сохранить копию проекта).

Работая далее с копией текущего прототипа, имейте в виду, что в среде Expression Blend IDE отсутствует возможность автоматического преобразования прототипа в рабочий вариант приложения. Элементы базового кода, имеющие непосредственное отношение к прототипу, вам придется вычленять вручную. Ниже поясняется, как это делается, а более подробно процедура преобразования прототипов приложений на платформах Silverlight и WPF в рабочий вариант исходного кода описывается в руководстве пользователя Expression Blend.

Примечание. Процесс преобразования прототипа приложения на платформе WPF несколько отличается от аналогичного процесса на платформе Silverlight. За дополнительными сведениями по данному вопросу обращайтесь к разделу “Преобразование в рабочий вариант проекта” (Convert into a production project) руководства пользователя Expression Blend.

Видоизменение файлов с расширением `*.csproj`

Первая стадия процесса преобразования прототипа в рабочий вариант приложения состоит в том, чтобы видоизменить в текущем решении файл проекта с расширением `*.csproj`, в котором содержатся ссылки на файлы разметки (`App.xaml`) и исходного кода (`App.xaml.cs`) приложения. В рассматриваемом здесь примере проекта таким файлом является `OnlineStoreApp.csproj`. Откройте этот файл в обычном текстовом редакторе, например в WordPad, и удалите из него приведенные ниже элементы разметки в коде XML[7].

```
<ExpressionBlendPrototypingEnabled>
  False
</ExpressionBlendPrototypingEnabled>
<ExpressionBlendPrototypeHarness>
  True
</ExpressionBlendPrototypeHarness>
```

После этого сохраните и закройте данный файл проекта. Затем откройте в текущем решении файл вспомогательного проекта с расширением `*.csproj`, в котором содержатся ссылки на экраны прототипа (в данном примере это файл проекта `OnlineStoreApp-Screens.csproj`). Найдите и удалите те же самые строки разметки в коде XML, а затем сохраните видоизмененный файл проекта.

[7] Вполне возможно, что в последующих версиях Expression Blend и Visual Studio исходная структура файла проекта на C# претерпит изменения. Поэтому здесь предполагается, что вы пользуетесь версией Expression Blend 4 или Visual Studio 2010. В противном случае обращайтесь за справкой к руководству пользователя той интегрированной среды разработки, которой пользуетесь в настоящий момент.

Обновление ссылок на сборки в корневом проекте

Внеся упомянутые выше изменения в текущее решение, вернитесь к среде Expression Blend IDE и перейдите на вкладку Projects, чтобы удалить сборку `Microsoft.Expression.Prototyping.Runtime.dll`, как показано на рис. 8.48.

Затем выберите команду Project⇨Add Reference из главного меню, чтобы добавить ссылку на сборку `System.Windows.Controls.Navigation.dll` из платформы Silverlight 4.0, которая по умолчанию находится в папке `C:\Program Files\Microsoft SDKs\Silverlight\v4.0\Libraries\Client`.

Примечание. Если на вашем компьютере установлена 64-разрядная версия Windows, замените `Program Files` на `Program Files(x86)` в пути к файлу данной сборки.

Аналогичным образом удалите ссылку на сборку `Microsoft.Expression.Prototyping.Runtime.dll` и добавьте ссылку на сборку `System.Windows.Controls.Navigation.dll` во вспомогательном проекте (напомним, что в данном примере это файл `OnlineStoreAppScreens.csproj`).

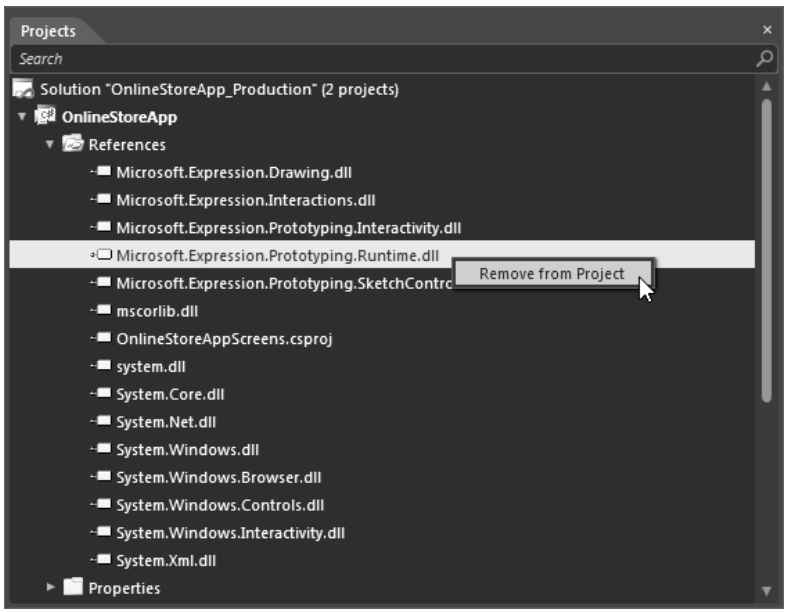


Рис. 8.48. Удаление библиотеки рабочих программ SketchFlow

Откройте для редактирования файл исходного кода приложения `App.xaml.cs`. В самом начале этого файла перед объявлением типа класса `App` вы обнаружите приведенный ниже атрибут уровня сборки. Закомментируйте весь этот атрибут.

```
[assembly: Microsoft.Expression.Prototyping.Services.SketchFlowLibraries
  ("OnlineStoreApp.Screens")]
```

Кроме того, найдите текущую реализацию обработчика событий `Startup`, наступающих при запуске прототипа приложения. Удалите или закомментируйте текущую строку кода в этом обработчике событий, заменив ее приведенным ниже фрагментом кода, а имя XAML-файла (в данном случае `Screen _ 1.xaml`) — именем любого XAML-ресурса, который, на ваш взгляд, должен отображаться при запуске приложения. И наконец,

замените имя проекта (в данном примере `/OnlineStoreApp.Screens`) на имя того проекта, в котором содержатся отдельные экраны.

```
private void Application_Startup(object sender, StartupEventArgs e)
{
  // Закомментируйте текущую строку кода!
  // this.RootVisual = new
  // Microsoft.Expression.Prototyping.Workspace.PlayerWindow();
  // Введите эту строку кода, обновив имя XAML-файла!
  this.RootVisual = new Frame()
  {
    Source = new Uri("/OnlineStoreApp.Screens;component/Screen_1.xaml",
           UriKind.Relative)
  };
}
```

Примечание. Имя файла, указываемое после косой черты в символьной строке `Uri`, должно быть таким же, как и в закомментированном атрибуте `SketchFlowLibraries`.

На данной стадии проигрыватель SketchFlow Player уже не применяется в проекте. Так, если вы запустите свое приложение Silverlight на выполнение, оно будет загружено в окно браузера подобно типичному XAP-файлу. Единственная правка, которая может вам еще понадобиться, заключается в замене стилей SketchFlow на ваши собственные, но это вам придется сделать самостоятельно! В качестве примера на рис. 8.49 (а также на цветной вклейке) приведен конечный результат преобразования прототипа в рабочий вариант приложения.

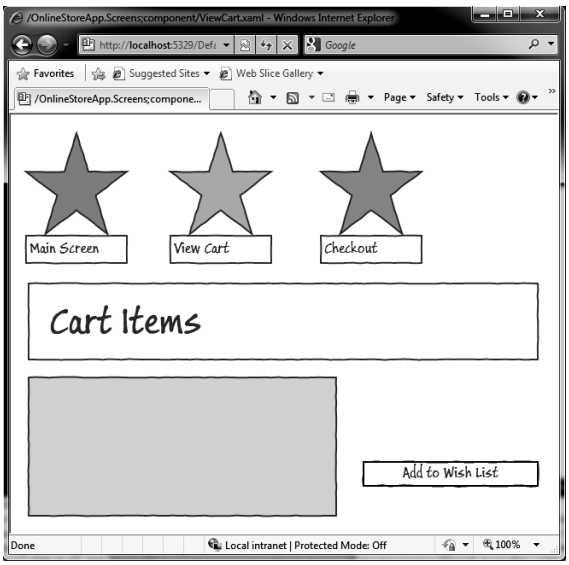


Рис. 8.49. Рабочий вариант приложения, загруженный в браузера в виде XAP-файла

Исходный код. Исходный код примера проекта `OnlineStoreApp _ Production` находится в папке `Ch 8 Code` загружаемого архива примеров проектов к данной книге.

Резюме

В заключительной главе этой книги было рассмотрено место компонента SketchFlow в жизненном цикле разработки приложений в среде Expression Blend IDE. Как упоминалось в начале главы, процесс создания прототипов приложений относится к числу самых распространенных задач, решаемых во время сбора отзывов и формирования требований заказчика. В Expression Blend имеется возможность создавать интерактивные прототипы приложений на платформах WPF и Silverlight, видоизменяя и расширяя их по ходу разработки, а в идеальном случае — при непосредственном участии заказчика.

Как было показано на конкретных примерах проектов, типичное решение в SketchFlow состоит из двух проектов, причем один из них содержит все экраны (например, для объектов типа `Window` и `UserControl`), которые могут быть связаны вместе на панели SketchFlow Map. А с помощью экрана компонента можно скомпоновать специальный пользовательский интерфейс на основе объекта типа `UserControl`, чтобы внедрить его на любых дополнительных экранах. В примерах проектов, представленных в этой главе, с помощью экранов компонентов была смоделирована простая система навигации, хотя подобным образом можно построить неоднократно используемый пользовательский интерфейс любой сложности. В этой главе были также рассмотрены функциональные возможности упрощенного варианта редактора анимации в SketchFlow и объектов поведения, упрощающих процесс создания прототипов.

Кроме того, было показано, каким образом отзывы пользователей фиксируются в проигрывателе SketchFlow Player. И в заключение главы было рассмотрено применение решения, построенного средствами SketchFlow, в качестве отправной точки для формирования требований к проекту и формальной документации на него, а также типичная процедура переноса прототипа приложения на почву реального проекта.

Предметный указатель

И

К

М

Н

О

П

Р

С

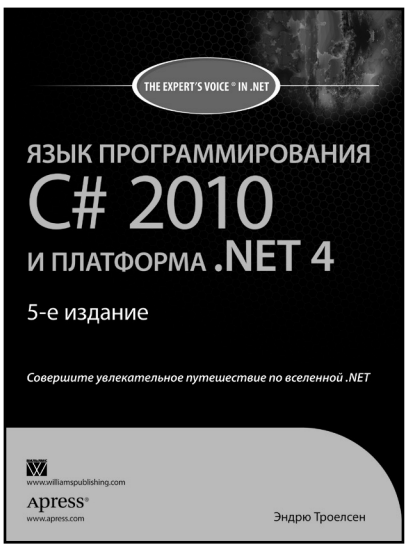
Т

Ш

Э

ЯЗЫК ПРОГРАММИРОВАНИЯ C# 2010 И ПЛАТФОРМА .NET 4
5-е издание

Эндрю Троелсен



www.williamspublishing.com

Версия .NET 4 привнесла множество новых API-интерфейсов в библиотеках базовых классов, а также новых синтаксических конструкций в языке C#.
В этой книге вы найдете полное описание всех нововведений в характерной для автора дружественной к читателю манере. Помимо общих вопросов, подробно рассматривается среда Dynamic Language Runtime (DLR); библиотека Task Parallel Library (TPL, включая PLINQ); технология ADO.NET Entity Framework (а также LINQ to EF); расширенное описание API-интерфейса Windows Presentation Foundation (WPF); улучшенная поддержка взаимодействия с COM.
В книге рассматриваются следующие темы

- Особенности платформы .NET 4 и языка Visual C# 2010
- Детали технологии .NET – лидера в производстве современного программного обеспечения
- Полезные советы по разработке от эксперта в .NET, который изучает эту платформу, начиная с ее первой версии
- Полное описание технологий WPF, WCF и WF, поддерживаемых ядром платформы .NET

ISBN 978-5-8459-1682-2

в продаже

MICROSOFT ASP.NET 4 С ПРИМЕРАМИ НА C# 2010
ДЛЯ ПРОФЕССИОНАЛОВ
4-Е ИЗДАНИЕ

Мэтью Мак-Дональд
Адам Фримен
Марио Шпушта

www.williamspublishing.com

Книга известных специалистов в области технологий .NET представляет собой учебное и справочное пособие для разработчиков .NET-приложений, использующих новую версию ASP.NET 4. Настоящее издание было полностью обновлено и дополнено с учетом последней версии ASP.NET, и теперь включает описание ASP. NET MVC, ASP.NET AJAX 4, ASP.NET Dynamic Data и Silverlight 3. Предложенный авторами практический подход к изложению материала не является простым повторением документации, а позволяет сконцентрироваться на решении конкретных задач, связанных с разработкой веб-приложений разного уровня сложности. Глубина изложения материала превращает эту книгу в незаменимый источник информации для разработчиков приложений ASP.NET 4.

ISBN 978-5-8459-1702-7 **в продаже**

C# 4 И ПЛАТФОРМА .NET 4 ДЛЯ ПРОФЕССИОНАЛОВ

Кристиан Нейгел
Билл Ивьен
Джей Глинн
Карли Уотсон
Морган Скиннер

www.dialektika.com

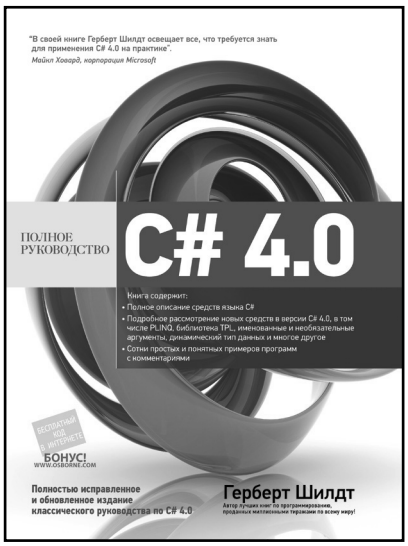
Книга известных специалистов в области разработки приложений с использованием .NET Framework посвящена программированию на языке C# 2010 в среде .NET Framework 4 и в предшествующих версиях. Книгу отличает простой и доступный стиль изложения, изобилие примеров и рекомендаций по написанию высококачественных программ. Подробно рассматриваются такие вопросы, как основы языка программирования C#, организация среды .NET, работа с данными, написание Windows- и веб-приложений, взаимодействие через сеть, создание веб-служб и многое другое. Немалое внимание уделено проблемам безопасности и сопровождения кода. Тщательно подобранный материал позволит без труда разобраться с тонкостями использования Windows Forms и построения веб-страниц. Читатели ознакомятся с работой в Visual Studio 2010, а также с применением различных технологий, встроенных в .NET. Книга рассчитана на программистов разной квалификации.

ISBN 978-5-8459-1656-3

в продаже

C# 4.0
ПОЛНОЕ РУКОВОДСТВО

Герберт Шилдт



www.williamspublishing.com

В этом полном руководстве по C# 4.0 — языку программирования, разработанному специально для среды .NET, — детально рассмотрены все основные средства языка: типы данных, операторы, управляющие операторы, классы, интерфейсы, методы, делегаты, индексаторы, события, указатели, обобщения, коллекции, основные библиотеки классов, средства многопоточного программирования и директивы препроцессора. Подробно описаны новые возможности C#, в том числе PLINQ, библиотека TPL, динамический тип данных, а также именованные и необязательные аргументы. Это справочное пособие снабжено массой полезных советов авторитетного автора и сотнями примеров программ с комментариями, благодаря которым они становятся понятными любому читателю независимо от уровня его подготовки. Книга рассчитана на широкий круг читателей, интересующихся программированием на C#.

ISBN 978-5-8459-1684-6 **в продаже**

LINQ
ЯЗЫК ИНТЕГРИРОВАННЫХ ЗАПРОСОВ В C# 2010 ДЛЯ ПРОФЕССИОНАЛОВ

Адам Фримен
Джозеф Раттц-мл.

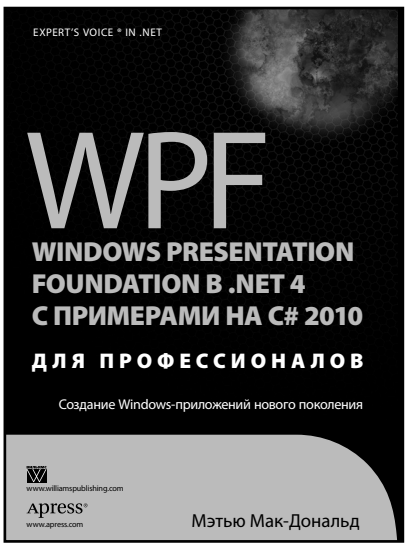
www.williamspublishing.com

LINQ представляет собой часть платформы .NET Framework, которая обеспечивает обобщенный подход к запросам данных из различных источников. Знание LINQ быстро становится необходимым для всех разработчиков приложений .NET. Эта книга посвящена написанию кода с помощью LINQ. Многие книги предлагают простые примеры использования методов и зачастую ими ограничиваются. Настоящая книга не такая. За счет демонстрации особенностей применения широкого разнообразия операций и прототипов LINQ она становится неоценимым источником реальных примеров использования LINQ. Книга написана в дружественном к читателю стиле и позволит эффективно освоить применение LINQ в разработке приложений.

ISBN 978-5-8459-1701-0 **в продаже**

WPF
WINDOWS PRESENTATION FOUNDATION В .NET 4 С ПРИМЕРАМИ НА C# 2010 ДЛЯ ПРОФЕССИОНАЛОВ

Мэтью Мак-Дональд



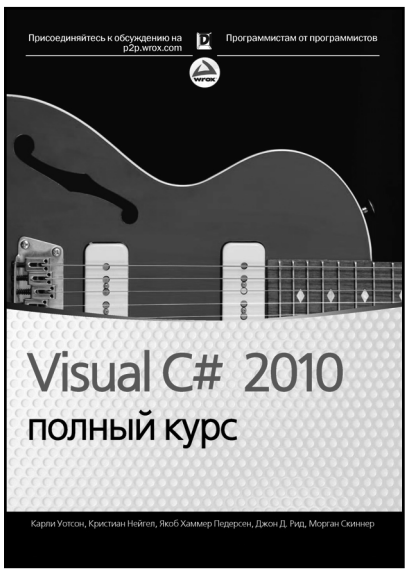
www.williamspublishing.com

Книга ведущего специалиста в области технологий .NET представляет собой учебное и справочное пособие по технологии WPF, входящей в состав .NET 4, для разработчиков высококлассных приложений, которые ориентированы на семейство операционных систем семейства Windows. Данная технология позволяет сочетать в рамках приложения пользовательский интерфейс, документы и медиасодержимое, располагая при этом полной поддержкой со стороны операционной системы. Функциональность WPF расширена поддержкой Tablet PC и других форм устройств ввода. Теперь WPF предоставляет более развитый конвейер рисования и печати, инфраструктуру доступности и автоматизации пользовательских интерфейсов, управляемые данными интерфейсы и виртуализацию, а также точки интеграции приложений с командной оболочкой Windows. В этой книге описано, как в действительности работает WPF.

ISBN 978-5-8459-1657-0 **в продаже**

VISUAL C# 2010 ПОЛНЫЙ КУРС

Карли Уотсон и др.



www.dialektika.com

Этот исчерпывающий источник для начинающих поможет подготовиться к работе с новым выпуском языка программирования C#. Книга предлагает исчерпывающее описание синтаксиса C#. Читатели узнают о таких фундаментальных основах, как переменные, управление потоком выполнения и объектно-ориентированном программировании. Кроме того, будет показано, как строить Windows- и веб-приложения, формы Windows и работать с данными. Пошаговые упражнения позволят лучше усвоить материал, предлагаемый в каждой главе. Прочтя эту книгу, читатели сразу же смогут приступить к написанию собственного кода для решения разнообразных реальных задач.
Книга рассчитана на начинающих программистов, а также будет полезна студентам и преподавателям дисциплин, связанных с программированием и разработкой для .NET.

ISBN 978-5-8459-1699-0 **в продаже**

SILVERLIGHT 3 С ПРИМЕРАМИ НА C# ДЛЯ ПРОФЕССИОНАЛОВ

Мэтью Мак-Дональд

www.williamspublishing.com

Silverlight 3 — это революционная технология, которая позволяет создавать мощные клиентские приложения, выполняемые в браузерах. Подобно Adobe Flash, Silverlight поддерживает обработку событий, двухмерное рисование, воспроизведение мультимедийного содержимого, сетевые функции, а также анимацию. В то же время технология Silverlight предназначена для платформы .NET и основана на кодах C#.

Наиболее важное преимущество Silverlight заключается в кроссплатформенности. В отличие от обычных приложений .NET, приложения Silverlight могут свободно выполняться в браузерах не от Microsoft (таких, как Firefox) и на других платформах (например, Mac OS X). По сути, Silverlight 3 — это усеченная реализация инфраструктуры .NET, выполняемая в контексте браузера, что делает ее одной из наиболее мощных технологий, представленных компанией Microsoft за последние годы.

ISBN 978-5-8459-1637-2

в продаже

C#: ПРОГРАММИРОВАНИЕ ДЛЯ ПРОФЕССИОНАЛОВ
2-Е ИЗДАНИЕ

Джон Скит

www.williamspublishing.com

Эта книга о языке C# версии 2 и выше. Язык C# 1, библиотеки инфраструктуры .NET и общеязыковая исполняющая среда (CLR) рассматриваются только в связи с языком C#. Такое решение существенно отличает данную книгу от большинства книг по C# и .NET.
Второе издание этой книги полностью пересмотрено и обновлено, здесь рассматриваются новые возможности языка C# 4, а также такие средства, как Code Contracts. Вы изучите нюансы программирования на C# на практике, узнаете, как работать с высокоуровневыми средствами, которые будете рады иметь в своем инструментарии. Книга поможет читателям избежать скрытых недостатков языка C# и ознакомиться с его "внутренними" проблемами.

ISBN 978-5-8459-1555-9 **в продаже**

Рис. 1.1

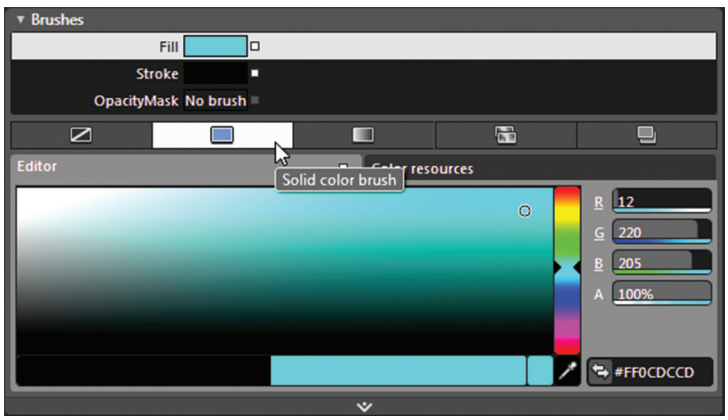


Рис. 2.13

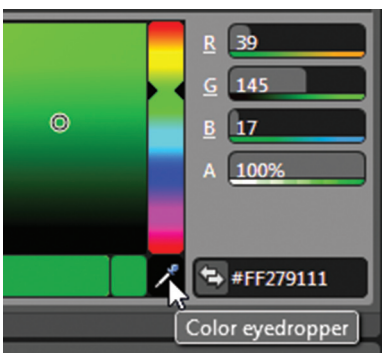


Рис. 2.14

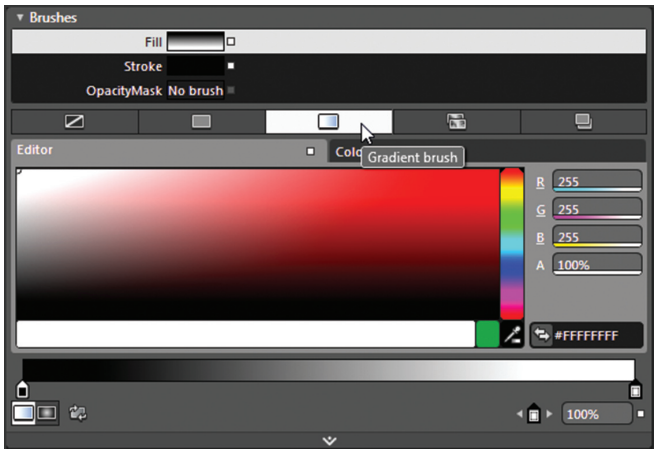


Рис. 2.15

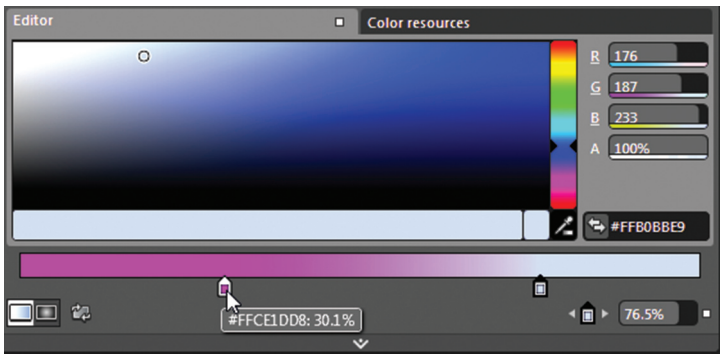


Рис. 2.16

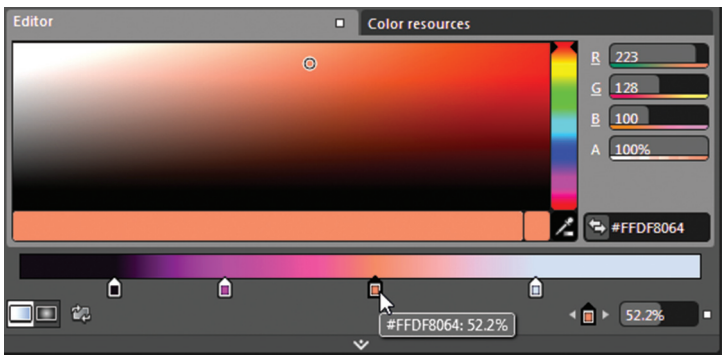


Рис. 2.17

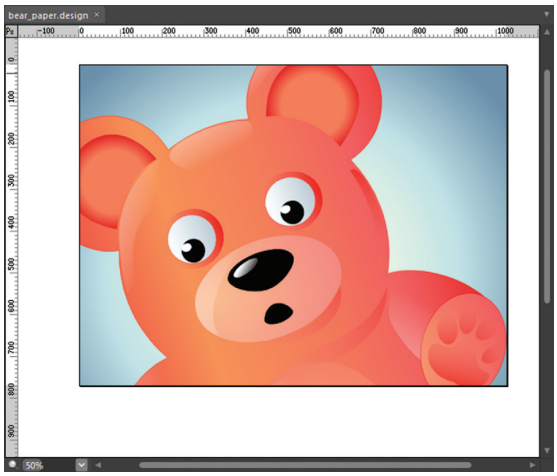


Рис. 2.33

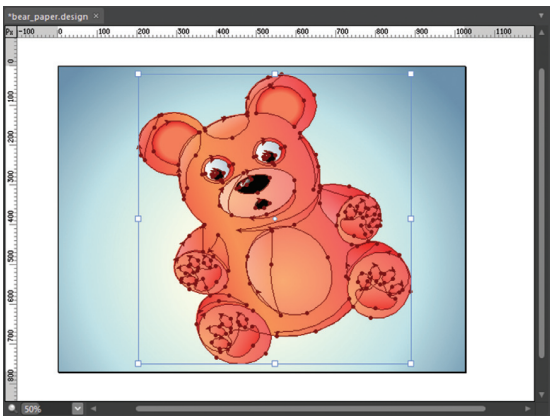


Рис. 2.35

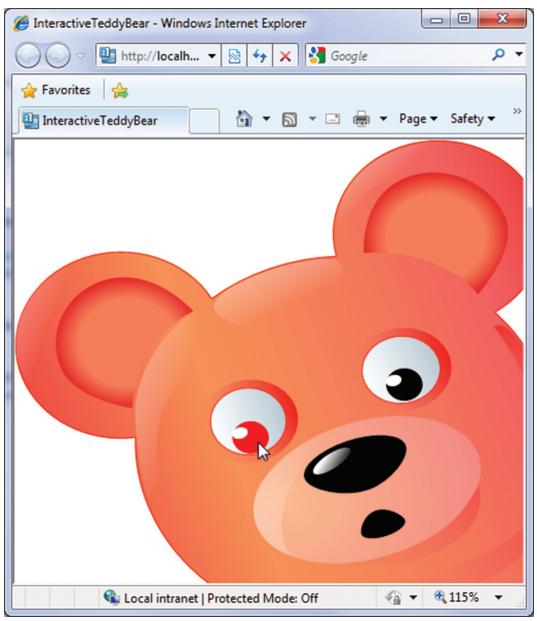


Рис. 2.41

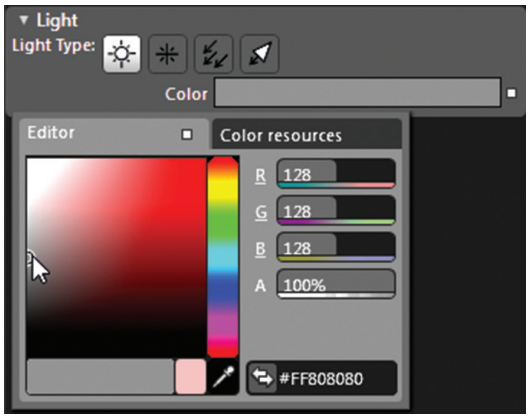


Рис. 2.56

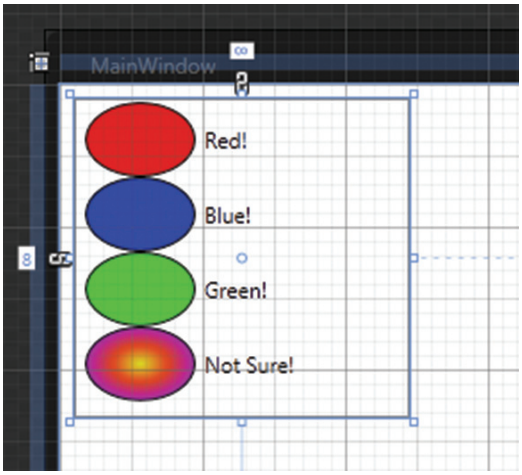


Рис. 4.13

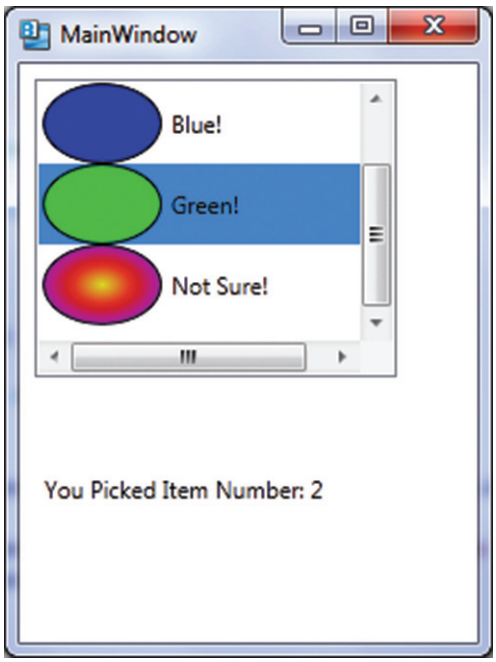


Рис. 4.15

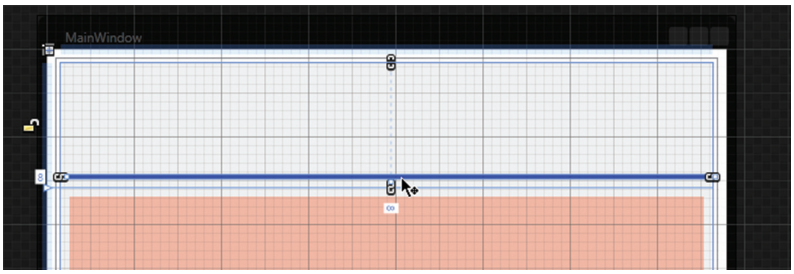


Рис. 4.29

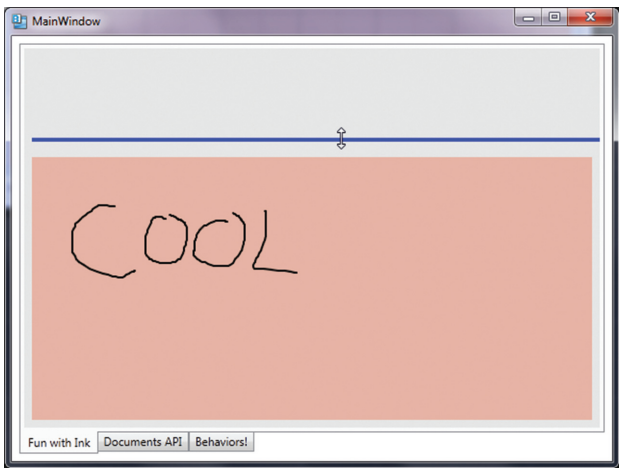


Рис. 4.30

Рис. 4.31

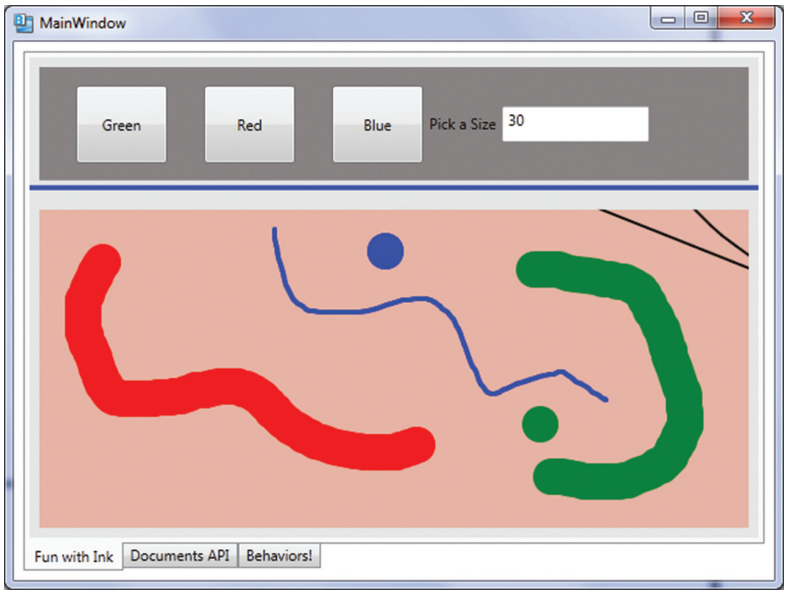


Рис. 4.32

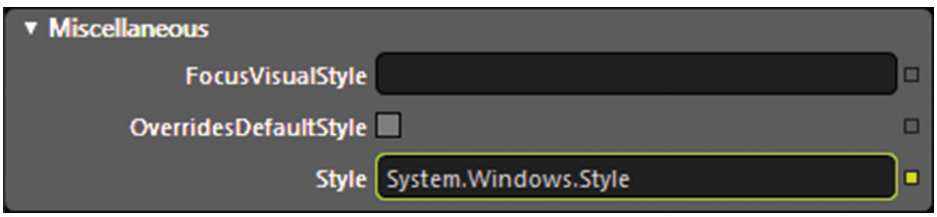


Рис. 5.2

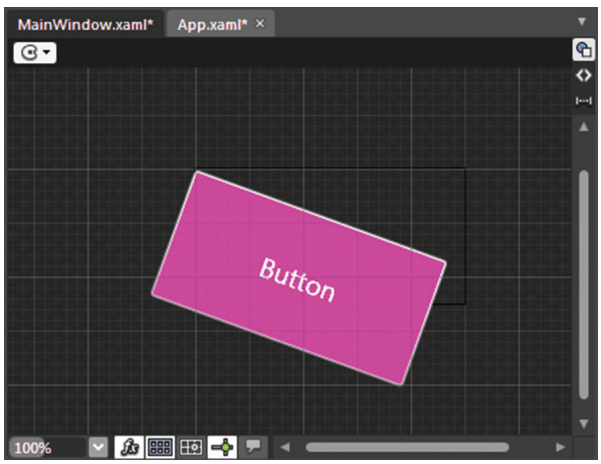


Рис. 5.6

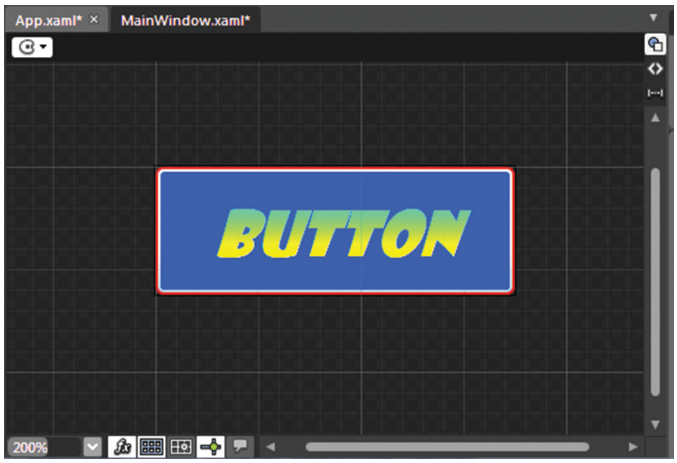


Рис. 5.11

Рис. 5.12

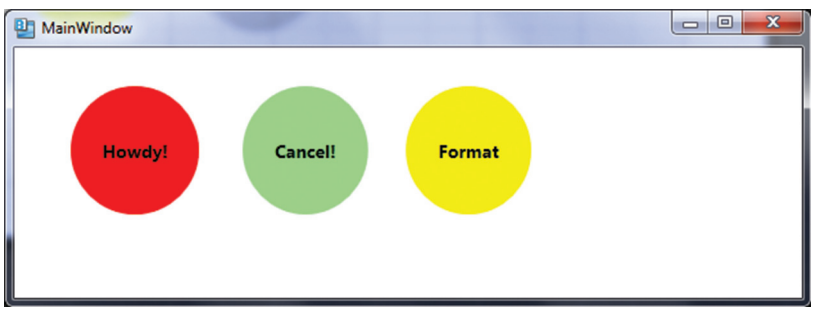


Рис. 5.23

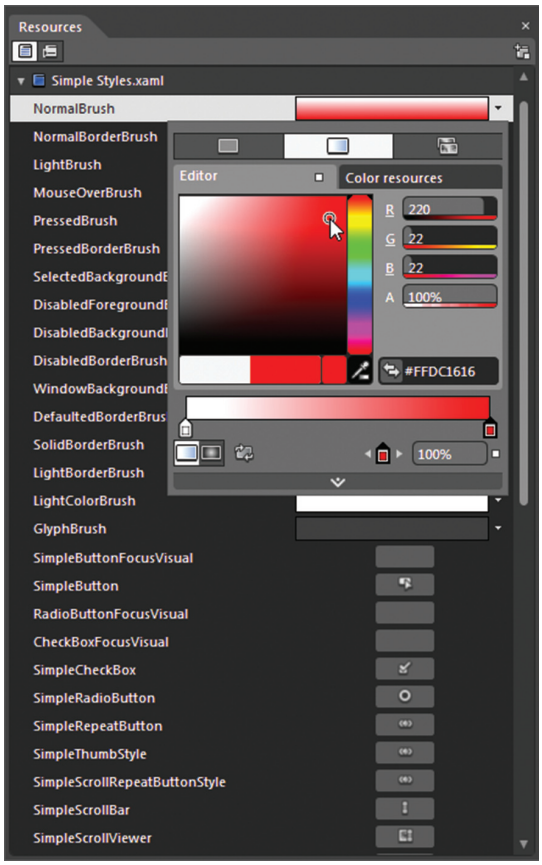


Рис. 5.16

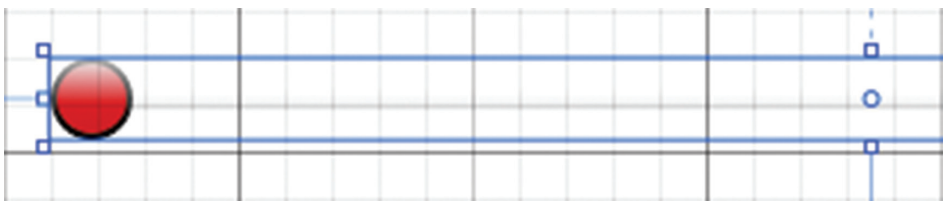


Рис. 5.17

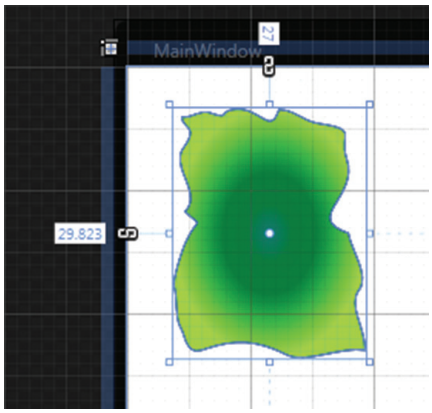


Рис. 5.33

Рис. 5.34

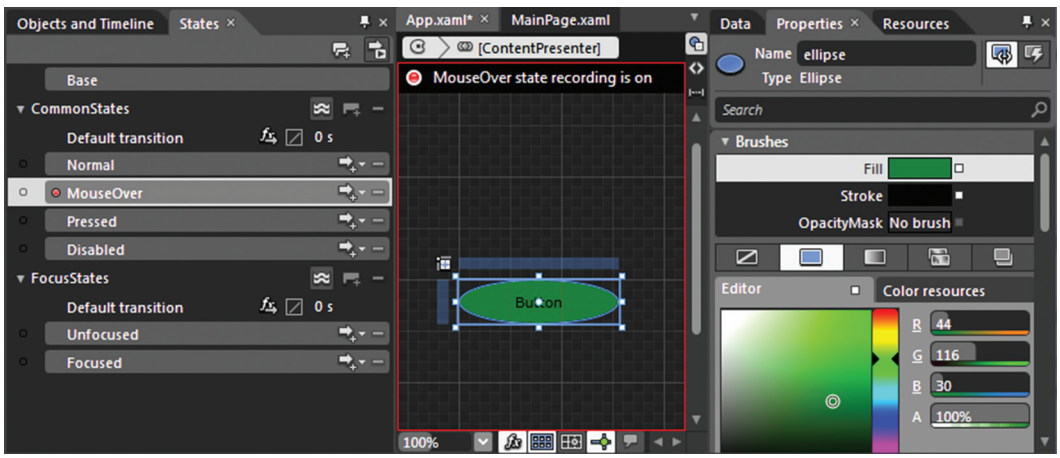


Рис. 5.46

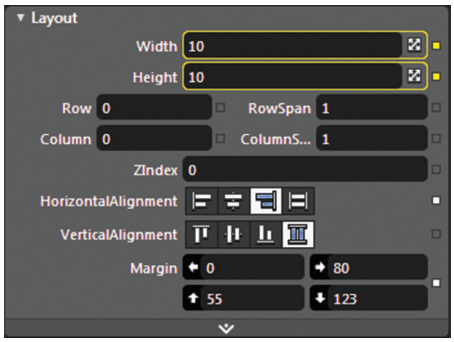


Рис. 6.6

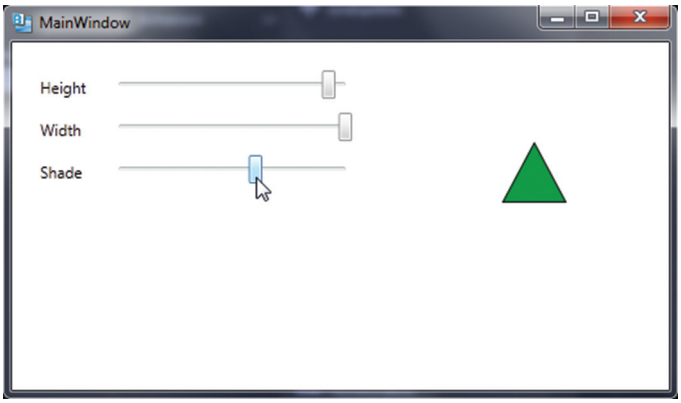


Рис. 6.11

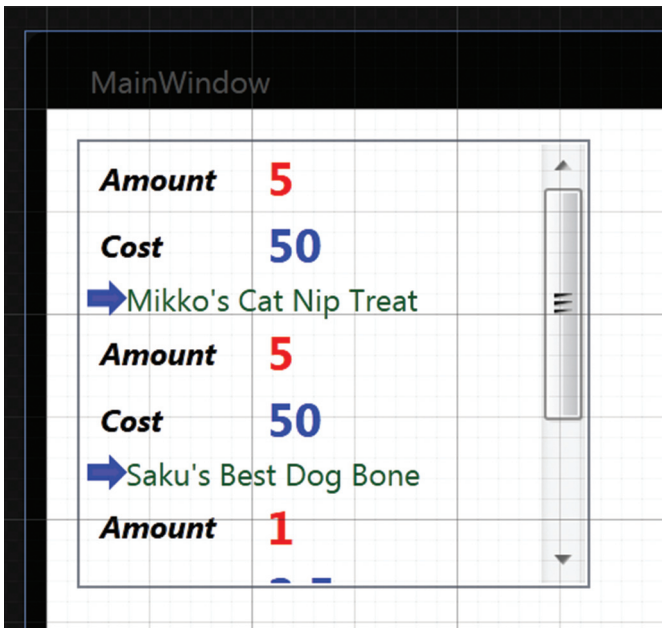


Рис. 6.30

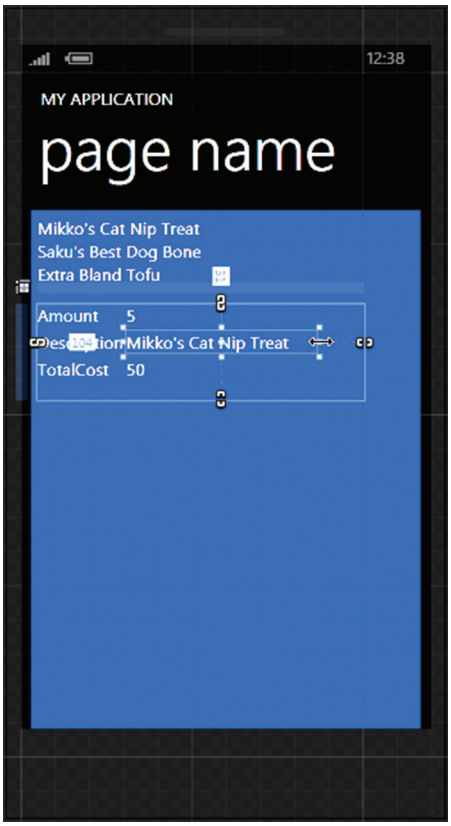


Рис. 7.13

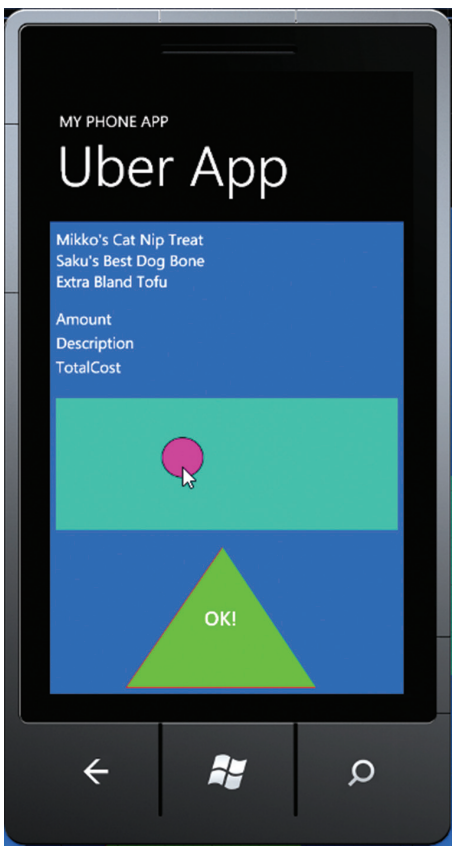


Рис. 7.19

Рис. 7.20

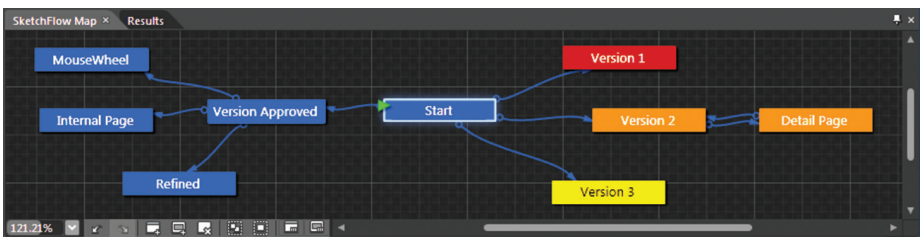


Рис. 8.3

Рис. 8.6

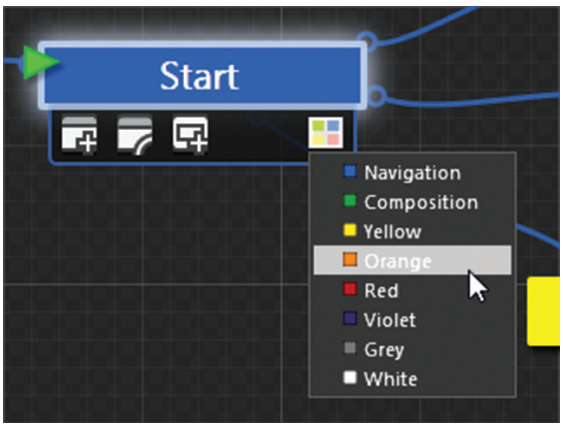


Рис. 8.8

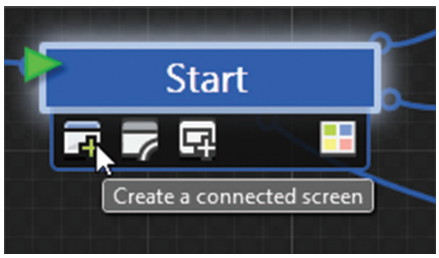


Рис. 8.9

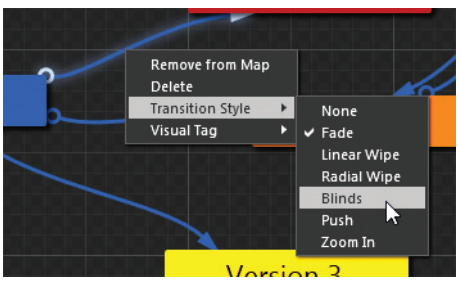


Рис. 8.10

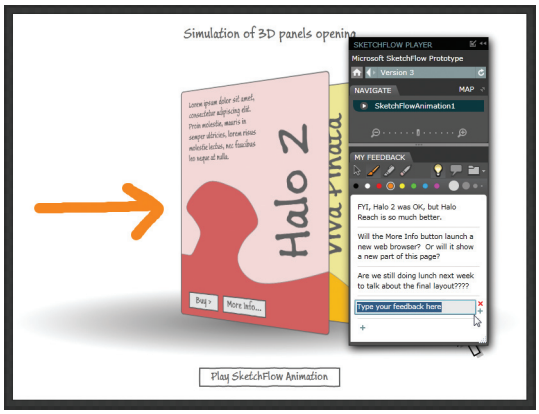


Рис. 8.13

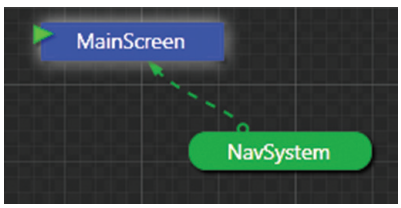


Рис. 8.25

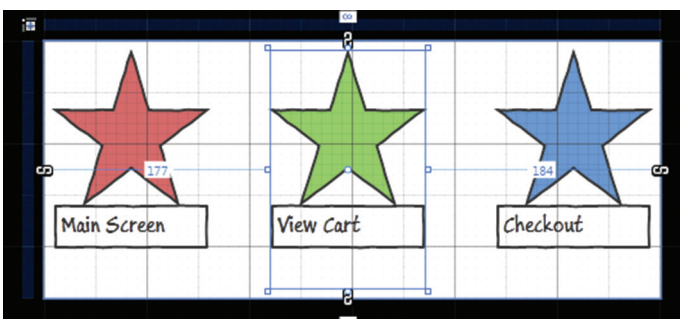


Рис. 8.27

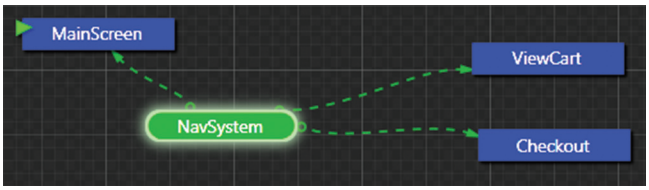


Рис. 8.32

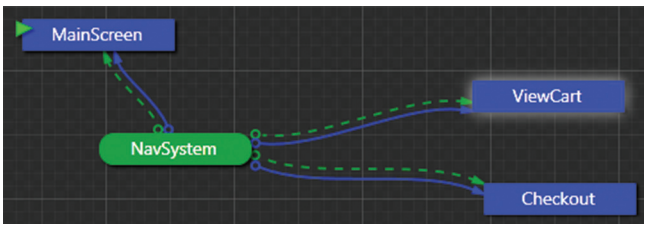


Рис. 8.35

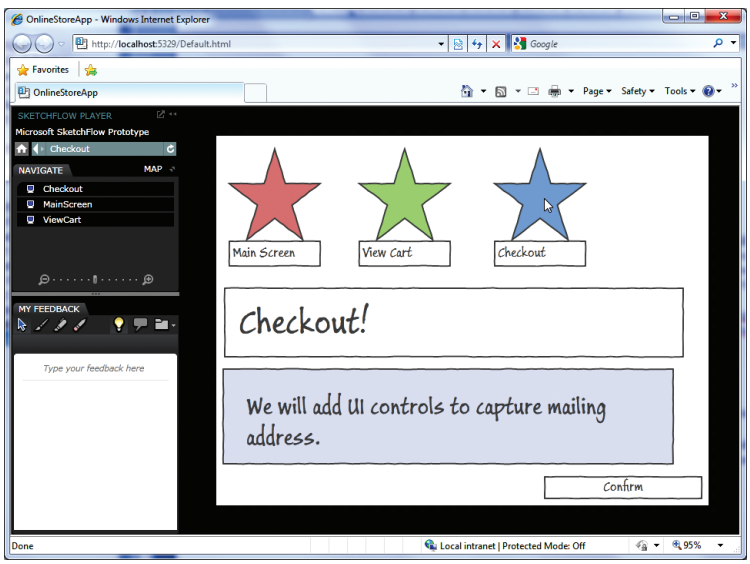


Рис. 8.36

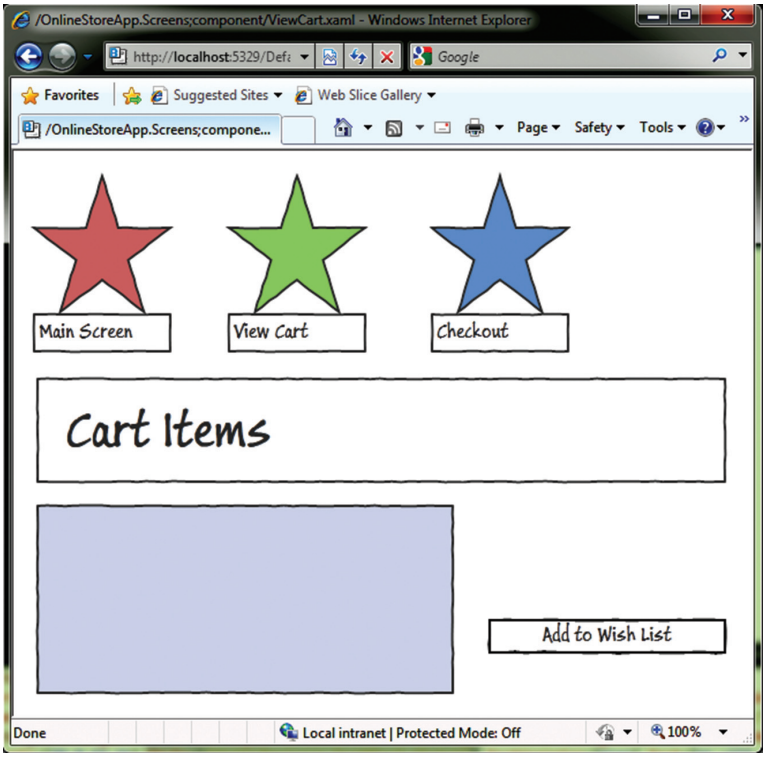

/OnlineStoreApp.Screens;component/ViewCart.xaml - Windows Internet Explorer
http://localhost:5329/Defa
Google
Favorites
Suggested Sites
Web Slice Gallery
/OnlineStoreApp.Screens;compone...
Page
Safety
Tools
Main Screen
View Cart
Checkout
Cart Items
Add to Wish List
Done
Local intranet | Protected Mode: Off
100%

Рис. 8.49