18

Подсистема Windows Presentation Foundation (WPF)

В ЭТОЙ ГЛАВЕ...

- Основы языка XAML
- Создание WPF-приложения
- Настройка стиля WPF-приложения
- Внедрение WPF-контента в проект Windows Forms
- Внедрение контента проекта Windows Forms в WPF-проект
- Использование визуализатора WPF

Начиная разработку нового клиентского приложения для системы Windows с помощью интегрированной среды Visual Studio, пользователь может выбрать одну из двух технологий: Windows Forms или Windows Presentation Foundation (WPF). Это совершенно разные интерфейсы прикладного программирования для управления слоем презентации пользовательского приложения. Технология WPF чрезвычайно мощная и гибкая. Она была разработана для преодоления недостатков и ограничений, присущих технологии Windows Forms. Во многих аспектах технология WPF является преемником Windows Forms. Однако овладение этой мощной и гибкой подсистемой требует от разработчика больших усилий, поскольку она заметно отличается от своего предшественника.

В главе описывается процесс создания простого WPF-приложения с помощью интегрированной среды Visual Studio 2010. Подробное описание технологии WPF выходит за рамки рассмотрения данной книги, так как для этого потребовалось бы слишком много места. Вместо этого мы приводим обзор возможностей системы Visual Studio 2010, помогающих быстро создавать пользовательские интерфейсы с помощью языка XAML.

Что такое WPF

Windows Presentation Foundation — это презентационный функциональный слой для системы Windows. Что же делает его уникальным и почему его следует считать достойной альтернативой подсистеме Windows Forms? В то время, как подсистема Windows Forms для рендеринга использует растровые механизмы GDI/GDI+, подсистема WPF имеет свой собственный векторный механизм рендеринга. По этой причине она не создает окна и элементы управления стандартным способом и в стандартном виде, присущем системе Windows. Технология WPF радикально отличается от технологии Windows Forms. В графической подсистеме Windows Forms разработчик обычно определял пользовательский интерфейс с помощью визуального конструктора, автоматически генерируя код (на языке проекта) в файле с расширением `.designer`. Таким образом, по существу, пользовательский интерфейс определялся и управлялся кодом на языке C# или VB. В то же время пользовательский интерфейс, созданный по технологии WPF, фактически определяется языком разметки Extensible Application Markup Language, который обычно называют XAML (произносится как "zammel"). Этот язык основан на языке XML и специально разработан компанией Microsoft для использования в системе WPF. Именно язык XAML, лежащий в основе технологии WPF, обеспечивает ее мощь и гибкость, позволяя разрабатывать намного более богатые и особенные пользовательские интерфейсы по сравнению с технологией Windows Forms. Определение пользовательского интерфейса с помощью языка XAML является одинаковым для любого языка проекта. По этой причине наряду с новыми возможностями управления пользовательским интерфейсом появилось большое количество новых вспомогательных концепций, влияющих на код. Например, возникли свойства зависимостей, значениями которых являются выражения, что часто требуется во многих сценариях связывания для поддержки сложных возможностей связывания, предоставляемых языком XAML. Тем не менее код в WPF-приложении почти не отличается от кода стандартного приложения, созданного по технологии Windows Forms, — основное внимание разработчику следует уделить языку XAML.

При разработке WPF-приложений необходимо думать совершенно иначе, чем при разработке приложений по технологии Windows Forms. Разработчик должен сконцентрировать свое внимание на преимуществах новых возможностей связывания, предоставляемых языком XAML, и рассматривать свой код не как контроллер пользовательского интерфейса, а как обслуживающий механизм. Теперь код не должен "проталкивать" данные в пользовательский интерфейс и говорить ему, что делать. Вместо этого пользовательский интерфейс должен спрашивать у кода, что делать, и посылать ему запросы на данные (т.е. "выталкивать" их из него). Разница почти незаметна, но способ определения презентационного слоя приложения изменяется радикально. Представьте себе, что пользовательский интерфейс теперь начальник. Код может (и должен) принимать решения, но больше не должен инициировать действия.

Это "новое мышление" порождает новые шаблоны проектирования, описывающие взаимодействие кода и элементов пользовательского интерфейса, такие как популярный шаблон Model-View-ViewModel (MVVM), позволяющий намного лучше проводить модульное тестирование кода, обслуживающего пользовательский интерфейс и поддерживающего явное разделение между элементами проектировщика и разработчика в рамках проекта. В результате изменяется способ написания кода, и в итоге — способ проектирования приложения. Такое явное разделение упрощает работу проектировщика и разработчика, позволяя проектировщику работать с помощью

программы Expression Blend над той же частью проекта, над которой работает разработчик (с помощью системы Visual Studio), и при этом избегать столкновения.

Благодаря гибкости языка XAML система WPF позволяет разрабатывать уникальные пользовательские интерфейсы и способы *интерактивного взаимодействия* (user experiences). В их основе лежат стили и шаблонные функциональные возможности системы WPF, которая отделяет внешний вид элементов управления от их поведения. Это позволяет легко изменять внешний вид элементов управления без изменения самого элемента.

Хорошая новость заключается в том, что благодаря применению языка XAML технология WPF реализует лучший способ определения пользовательских интерфейсов по сравнению с Windows Forms, а также порождает большое количество дополнительных понятий. Плохая новость состоит в том, что эта гибкость и мощь языка XAML требует значительного времени для обучения даже опытных разработчиков. У разработчиков, достигших приемлемого уровня производительности труда с помощью технологии Windows Forms, система WPF несомненно вызовет чувство неудовлетворения, пока они не освоят новые концепции и не испытают реальной потребности изменить свой образ мышления. Многие простые задачи вначале выглядят намного более сложными, чем они могли бы быть, если их реализовать с помощью технологии Windows Forms. Однако со временем разработчики оценят преимущества новых возможностей, которые им предоставляют технология WPF и язык XAML. Поскольку графическая система Silverlight имеет много общего с системой WPF (обе основаны на языке XAML, так что Silverlight, по существу, — это подсистема системы WPF), изучение технологии WPF позволит разработчиками одновременно научиться создавать приложения с помощью системы Silverlight.

Если читатели уже знакомы с более ранними версиями системы WPF (поставляемыми с платформами .NET Framework 3.0 и 3.5), то они могли заметить, что текст, прорисованный в системе WPF, часто выглядел слегка расплывчатым, а не четким и контрастным. Это приводило к многочисленным жалобам разработчиков. К счастью, в версии .NET Framework 4.0 прорисовка текста была значительно улучшена, и теперь разочарованные разработчики могут вернуться к использованию этой технологии. Компания Microsoft продемонстрировала свою приверженность технологии WPF, переписав с ее помощью редактор кода в среде Visual Studio 2010, чтобы показать ее мощь и гибкость.

Начало работы с системой WPF

Открыв диалоговое окно New Project, разработчик видит множество встроенных шаблонов проектов для технологии WPF, поставляемых вместе с системой Visual Studio 2010: WPF Application, WPF Browser Application, WPF Custom Control Library и WPF User Control Library (рис. 18.1).

Обратите внимание на то, что эти проекты большей частью являются эквивалентами проектов для графической системы Windows Forms. Исключением является шаблон WPF Browser Application, генерирующий XBAP-файл, использующий браузер как контейнер для пользовательского приложения “толстого” клиента (это похоже на то, что делает программа Silverlight, за исключением того, что XBAP-приложение предназначено для полноценной платформы .NET Framework, которая должна быть инсталлирована на компьютере клиента).

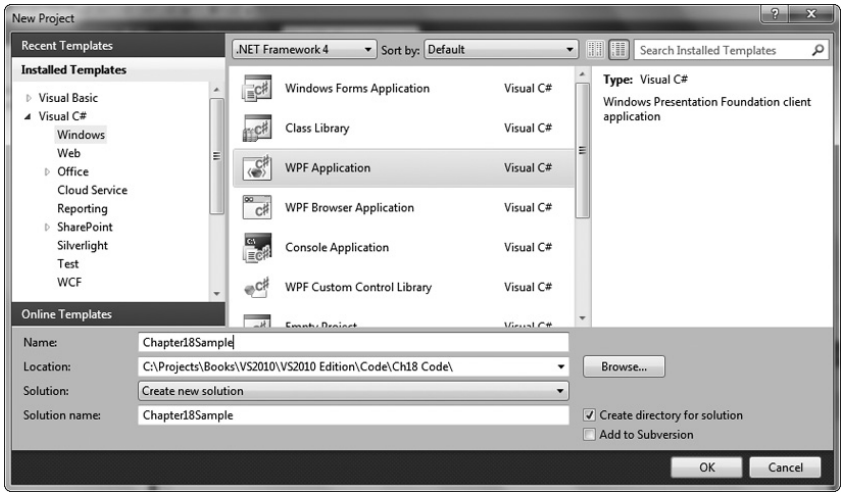

Рис. 18.1

В качестве примера создадим проект, используя шаблон WPF Application, хотя большинство функциональных возможностей системы Visual Studio 2010, обсуждаемых здесь, в одинаковой степени относится и к другим типам проектов. Генерируемая структура проекта выглядит так, как показано на рис. 18.2.

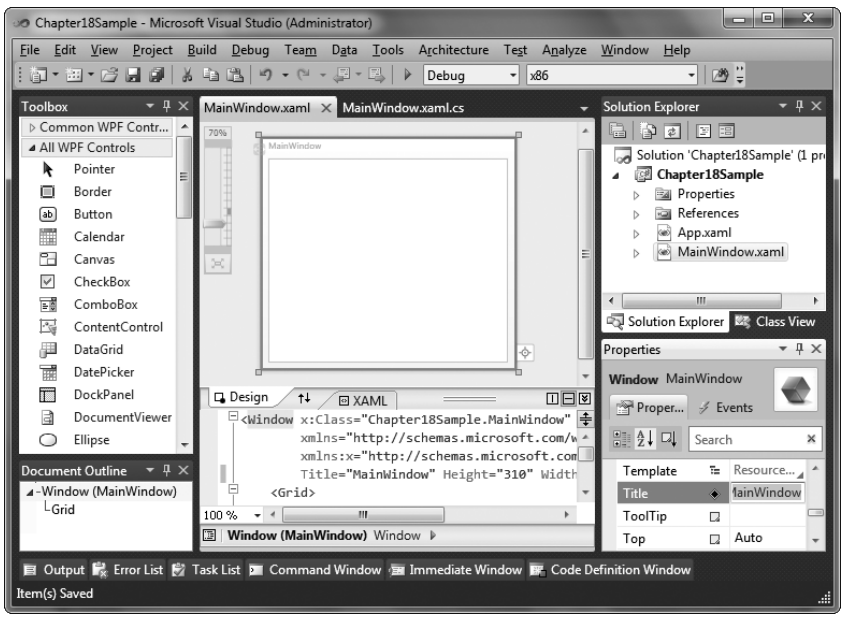

Рис. 18.2

Здесь видно, что структура проекта состоит из файлов `App.xaml` и `MainWindow.xaml`, каждому из которых соответствует свой исходный файл (`.cs` или `.vb`), которые можно увидеть, развернув соответствующие узлы проекта. На этом этапе файл `App.xaml`

содержит элемент Application XAML, имеющий атрибут `StartupUri`, который используется для определения первоначального загружаемого XAML-файла (по умолчанию — `MainWindow.xaml`). В технологии Windows Forms он соответствует начальной форме. Итак, если мы захотим изменить имена файла `MainWindow.xaml` и его соответствующего класса на более информативные, то должны будем выполнить следующие действия.

- Изменить имя файла с расширением `.xaml`. Исходный файл будет переименован автоматически и согласованно.
- Изменить имя класса в исходном файле вместе с именем конструктора, а также значение атрибута `x:Class` элемента Window в файле с расширением `.xaml`, для того, чтобы сослаться на новое имя класса (полностью квалифицированное своим пространством имен). Обратите внимание на то, что два последних этапа выполняются автоматически, если сначала изменить имя класса в исходном файле и использовать интеллектуальные дескрипторы, которые появляются после этого, для того, чтобы переименовать объект во всех местах, где на него есть ссылки.
- Изменить атрибут `StartupUri` элемента Application в файле `App.xaml` и указать новое имя файла с расширением `.xaml` (поскольку он является объектом запуска).

Как видим, для переименования файла в WPF-проекте требуется внести больше изменений, чем в стандартном проекте Windows Forms, однако эти изменения вполне очевидны, если вы знаете, что делаете (и используете интеллектуальные дескрипторы, чтобы уменьшить количество требуемых действий).

Изучая окна системы Visual Studio на рис. 18.2, можно увидеть, что уже известное нам инструментальное окно Toolbox, расположенное вдоль левого края экрана, содержит элементы управления WPF, похожие на элементы управления, которые используются при создании приложений по технологии Windows Forms. Под этим окном, все там же слева, расположено инструментальное окно Document Outline. И в проектах Windows Forms, и в проектах Web Applications оно содержит иерархическое представление элементов текущего окна. Выбрав один из этих узлов, можно выделить соответствующий элемент управления в основном окне редактора. Это упрощает навигацию по более сложным документам. Интересной особенностью окна Document Outline при работе с системой WPF является тот факт, что если курсор мыши "зависает" над элементом, то пользователь может увидеть его миниатюрное изображение. Это помогает идентифицировать выбираемый элемент управления.

Инструментальное окно Document Outline можно свернуть у одного из краев рабочей области Visual Studio. И наоборот, его можно открыть, выбрав команду View⇨Other Windows.

На рис. 18.2, *справа*, показано инструментальное окно Properties. И внешним видом, и функционированием оно очень похоже на инструментальное окно Properties в конструкторе форм в системе Windows Forms. Однако на самом деле это окно относится к конструктору системы WPF и имеет дополнительные возможности для редактирования окон и элементов управления в этой графической системе. Посередине экрана расположена основная область редактирования и предварительного просмотра, которая в данный момент разделена, чтобы продемонстрировать как визуальный макет окна (в верхней части), так и код на языке XAML, который этот макет определяет (в нижней части).

Основы языка XAML

Если читатели знакомы с языком XML (или хотя бы с языком HTML), то они заметят, что синтаксис языка XAML относительно прост, поскольку он основан на языке XML. Язык XAML имеет только один корневой узел, и элементы вкладываются один в другой, определяя внешний вид и содержимое пользовательского интерфейса. Каждому элементу языка XAML соответствует класс платформы .NET, а именам атрибутов — свойства/события этого класса. Отметим, что имена элементов и атрибутов чувствительны к регистру.

Посмотрим на XAML-файл, по умолчанию созданный для класса `MainWindow`.

```
<Window x:Class="Chapter18Sample.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="MainWindow" Height="300" Width="300">
    <Grid>

        </Grid>
</Window>
```

Здесь `Window` — корневой узел, а `Grid` — элемент, который в нем содержится. Для того чтобы понять их смысл, следует поразмышлять о них в терминах "окна, содержащего сетку". Корневому узлу соответствует класс, определенный атрибутом `x:Class` и содержащий объявления префиксов пространств имен (вскоре мы вернемся к этой теме), а также некоторые атрибуты, используемые для задания значений свойств (`Title`, `Height` и `Width`) класса `Window`. Значения всех атрибутов (независимо от типа) должны быть заключены в кавычки.

В корневом узле определены два префикса пространств имен. Оба эти префикса объявлены с помощью атрибута `xmlns` (XML-атрибута, используемого для объявления пространств имен). Объявления префиксов пространств имен в языке XAML можно рассматривать как аналоги инструкций `using`/`Imports` в начале классов в языках C#/VB, но без кавычек.

Эти объявления присваивают пространствам имен уникальные префиксы, используемые в XAML-файле, а также префиксы, используемые для квалификации пространств имен при ссылке на классы, находящиеся в них (т.е. для задания местоположения классов). Префиксы сокращают многословность языка XAML, позволяя при ссылке на класс в файле XAML указывать только префикс, а не все пространство имен. Префикс определяется сразу после двоеточия в выражении `xmlns`. Первое определение на самом деле не задает префикс, потому что оно определяет пространство имен, заданное по умолчанию (пространство имен системы WPF). В то же время второе объявление задает `x` как префикс пространства имен языка XAML. Оба определения отображаются в унифицированные идентификаторы ресурса URI, а не в конкретные пространства имен, поскольку они представляют собой консолидированные пространства имен (т.е. охватывают несколько пространств имен) и поэтому ссылаются на единый идентификатор URI, используемый для определения данной консолидации. Однако разработчику не следует беспокоиться об этом — нужно просто оставить эти определения как есть и добавлять свои определения после них. Собственные определения пространств имен, добавляемые разработчиком, почти всегда начинаются с ключевого слова `clr-namespace`, которое ссылается на пространство имен `CLR` и содержащую его сборку. Рассмотрим пример.

```
xmlns:wpf="clr-namespace:Microsoft.Windows.Controls;assemblty=WPFToolkit"
```

Префиксы могут быть любыми, но лучше всего делать их по возможности короткими. Как правило, пространства имен определяются в корневом узле XAML-файла. Это не обязательно, поскольку префикс пространства имен может быть определен на любом уровне в XAML-файле. И все же стандартной практикой стало собирать префиксы вместе в корневом узле, чтобы за ними было легче следить.

При необходимости сослаться на элемент управления в коде или связать его с другим элементом управления в XAML-файле (например, при связывании элемента ElementName) нужно приписать этому элементу управления какое-то имя. Многие элементы управления используют для этой цели свойство Name, но присвоить имя элементу управления можно с помощью атрибута `x:Name`. Он определен в пространстве имен XAML (поэтому имеет префикс `x:`) и может применяться к любому элементу управления. Если свойство Name реализовано (чаще всего так и есть, потому что это свойство определено в базовых классах, которые наследуются большинством элементов управления), этот атрибут просто отображается в него и служит той же цели. Рассмотрим пример. Инструкция

```
<Button x:Name="OKButton" Content="OK" />
```

эквивалентна инструкции

```
<Button Name="OKButton" Content="OK" />
```

Оба варианта являются правильными с формальной точки зрения (хотя в приложении Silverlight большинство элементов управления не поддерживают атрибут `Name` и должны использовать вместо него атрибут `x:Name`). Как только эти свойства заданы, генерируется поле (в автоматически генерируемом коде, который разработчику недоступен), которое можно использовать для ссылки на элемент управления.

Элементы управления системы WPF

Система WPF содержит множество элементов управления, предназначенных для использования в пользовательском интерфейсе. Эти элементы управления примерно соответствуют стандартным элементам управления в системе Windows Forms. Если взглянуть на предыдущие версии системы WPF, то можно заметить много элементов управления (таких как Calendar, DatePicker, DataGrid и др.), которые были включены в стандартный набор для системы Windows Forms, но не вошли в стандартный набор для системы WPF. Для того чтобы воспользоваться этими элементами управления, пользователь должен подключить свободно распространяемый пакет WPF Toolkit, размещенный на сайте CodePlex. Этот пакет был разработан компанией Microsoft, чтобы компенсировать указанный недостаток первой версии системы WPF за счет недостающих компонентов. Однако с появлением системы WPF 4.0 многие элементы управления из пакета WPF Toolkit уже вошли в стандартный набор. Разумеется, пользователь может использовать пакеты элементов управления, созданные другими разработчиками в тех случаях, когда стандартных средств недостаточно, но в настоящее время для этого нужны веские основания.

Несмотря на то что набор элементов управления в системе WPF сопоставим с набором элементов управления в системе Windows Forms, по своим свойствам они сильно отличаются от своих аналогов. Например, у многих элементов управления больше нет свойства Text, а вместо него появилось свойство Content, которое используется для присвоения элементу управления определенного содержимого (а следовательно, и имени). В большинстве случаев это свойство можно интерпретировать как прямой аналог свойства Text в элементах управления системы Windows Forms и просто присваивать ему какую-то строку, которая должна прорисовываться на экране. Однако на самом деле свойство Content может принимать не только текстовое значение, но и

любой элемент системы WPF. Это открывает практически безграничные возможности для настройки существующих элементов управления и освобождает пользователя от необходимости создавать свои собственные элементы управления. При разработке сложных пользовательских интерфейсов это очень полезно. Кроме того, многие элементы управления больше не имеют удобных свойств, которые существовали в системе Windows Forms. Это может показаться несколько странным. Например, у элемента управления Button в системе WPF нет свойства Image, позволявшего присваивать кнопке определенное изображение, как это было в системе Windows Forms. На первый взгляд, это сильно ограничивает возможности системы WPF, но это впечатление ошибочно, поскольку теперь у кнопки есть свойство Content. Так как свойство Content позволяет присваивать элементу управления системы WPF определенное содержимое, пользователь может присвоить ему элемент StackPanel (который будет рассмотрен в следующем разделе), содержащий как элемент управления Image, так и элемент управления TextBlock, обеспечивающие тот же самый эффект. Это требует от пользователя чуть больше усилий, чем при работе с системой Windows Forms, но позволяет ему проще моделировать содержимое кнопки на любой форме (а не подбирать элементы управления, которые он может реализовать) и обеспечивает невероятную гибкость системы WPF и языка XAML. Код на языке XAML для кнопки, показанной на рис. 18.3, имеет следующий вид.

```
<Button HorizontalAlignment="Left" VerticalAlignment="Top" Width="100"
 Height="30">
    <Button.Content>
         <StackPanel Orientation="Horizontal">
              <Image Source="/Chapter18Sample;component/Images/save.png"
               Width="16"
               Height="16" />
              <TextBlock Margin="5,0,0,0" Text="Save"
VerticalAlignment="Center" />
         </StackPanel>
    </Button.Content>
</Button>
```

Рис. 18.3

Другими замечательными свойствами, которые отличаются от свойств системы Windows Forms, являются свойства IsEnabled (которое в системе Windows Forms называлось Enabled) и Visibility (которое в системе Windows Forms называлось Visible). На примере свойства IsEnabled видно, что имена большинства булевых свойств имеют префикс Is (например, IsTabStop, IsHitTestVisible и т.д.), что соответствует стандартной схеме именования. Однако свойство Visibility больше не имеет булевого значения — теперь оно представляет собой перечисление, принимающее значение Visible, Hidden или Collapsed.

Не забывайте о пакете WPF Toolkit, доступном на веб-сайте `http://wpf.codeplex.com`, поскольку в него постоянно включаются новые элементы управления для системы WPF, которые могут оказаться полезными.

Компоновочные элементы управления в системе WPF

В системе Windows Forms для размещения элементов управления на рабочей поверхности использовались абсолютные координаты (т.е. каждый элемент управления имел явно заданные координаты *x* и *y*). Со временем появились элементы управления TableLayoutPanel и FlowLayoutPanel, которые могут содержать другие элементы управления. Это позволило создавать более сложные схемы размещения элементов

управления на форме. Однако концепция позиционирования элементов управления в системе WPF немного отличается от системы Windows Forms. Наряду с элементами управления, имеющими специальное предназначение (например, кнопками, полями ввода и т.п.), в системе WPF есть множество элементов управления, используемых специально для определения компоновки пользовательского интерфейса.

Компоновочные элементы управления являются невидимыми и предназначены для позиционирования других элементов управления на своей поверхности. В системе WPF нет поверхности, по умолчанию предназначенной для позиционирования элементов управления, — поверхность, на которой работает пользователь, определяется компоновочными элементами управления, образующими иерархию. Компоновочные элементы управления, как правило, находятся в этой иерархии непосредственно под корневым узлом XAML-файла и определяют метод компоновки, принятый по умолчанию для данного XAML-файла. Наиболее важными компоновочными элементами управления в системе WPF являются Grid, Canvas и StackPanel, поэтому мы рассмотрим каждый из них. Например, в XAML-файле, созданном по умолчанию для класса MainWindow, рассмотренного ранее, элемент Grid располагался непосредственно под корневым узлом Window, а значит, должен был по умолчанию служить для этого окна рабочей поверхностью. Разумеется, пользователь может заменить любой компоновочный элемент управления, а также использовать любые дополнительные элементы управления для создания дополнительных поверхностей и иного размещения элементов управления.

В следующем разделе мы рассмотрим, как скомпоновать форму с помощью рабочей поверхности конструктора, но сначала заглянем в код XAML, чтобы увидеть, как используются элементы управления.

Если пользователь хочет размещать элементы управления, используя абсолютную систему координат (как в системе Windows Forms), то он может в качестве поверхности выбрать элемент управления Canvas, предусмотренный в системе WPF.

Определение элемента Canvas в языке XAML выглядит очень просто.

```
<Canvas>

    </Canvas>
```

Для того чтобы разместить элемент управления (например, TextBox) на этой поверхности, используя координаты *x* и *y* (по отношению к левому верхнему углу элемента Canvas), необходимо использовать концепцию *присоединенных свойств* (attached properties), принятую в языке XAML. Элемент управления TextBox на самом деле не имеет свойств, определяющих его местоположение, поскольку его позиция в компоновочном элементе управления полностью зависит от типа этого элемента. Следовательно, свойства, которых недостает элементу управления TextBox, чтобы определить его положение в компоновочном элементе, должны быть позаимствованы у самого компоновочного элемента, поскольку именно он определяет, где в нем должен находиться элемент TextBox. Именно в этот момент на первый план выходят присоединенные свойства. Коротко говоря, присоединенными называют свойства, которые присваивают элементу управления какое-то значение, но определяются и принадлежат другому элементу управления, стоящему выше в иерархии элементов. При использовании такого свойства его имя уточняется именем элемента управления, в котором оно на самом деле определено, за которым следует точка, а затем имя элемента управления, в котором оно используется (например, Canvas.Left). Присвоив это значение другому элементу управления, который содержится внутри (как, например, наше поле ввода TextBox), элемент Canvas хранит это значение в себе и с его помощью управляет позицией поля ввода. Например, следующий фрагмент кода на языке тре-

бует разместить поле ввода в точке 15, 10, используя свойства Left и Top, определенные в элементе управления Canvas.

```
<Canvas>
     <TextBox Text="Hello" Canvas.Left="15" Canvas.Top="10" />
</Canvas>
```

В то время как абсолютное позиционирование в системе Windows Forms принято по умолчанию, в системе WPF для компоновки лучше всего использовать элемент управления Grid. Элемент управления Canvas следует использовать редко и только при крайней необходимости, поскольку элемент управления Grid представляет собой намного более мощное средство для определения компоновки форм и лучше работает в большинстве сценариев. Одним из самых заметных преимуществ элемента управления Grid является возможность автоматически изменять размер его содержимого при изменении его собственного размера. Поэтому пользователь может легко проектировать формы, автоматически изменяющие свои размеры, чтобы заполнить всю доступную область,— иначе говоря, размер и положение элементов управления в этой форме определяются динамически. Элемент управления Grid позволяет разделять свою область на подобласти (ячейки), в которые можно поместить другие элементы управления. Эти ячейки определяются путем задания строк и столбцов сетки с помощью значений свойств RowDefinitions и ColumnDefinitions. Пересечения строк и столбцов становятся ячейками, в которые можно помещать элементы управления.

Для определения строк и столбцов пользователь должен знать только, как определять сложные значения в языке XAML. До сих пор мы могли присваивать элементам управления лишь простые значения, которые отображались либо в элементарные типы данных платформы .NET, либо в имя элемента перечисления. Кроме того, мы могли конвертировать строку в соответствующий объект. Значения этих простых свойств использовались как атрибуты при определении элемента управления. Однако сложные значения нельзя присваивать таким способом, потому что они отображаются в объекты (т.е. необходимо выполнить присваивание нескольких свойство объекта) и должны определяться с помощью *синтаксиса "свойство–элемент"* (property element syntax). Поскольку свойства RowDefinitions и ColumnDefinitions элемента Grid представляют собой коллекции, они принимают сложные значения, которые должны быть определены с помощью синтаксиса "свойство–элемент". Например, рассмотрим сетку, состоящую из двух строк и трех столбцов, определенных с помощью синтаксиса "свойство–элемент".

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100" />
        <ColumnDefinition Width="150" />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
</Grid>
```

Для того чтобы задать свойство RowDefinitions с помощью синтаксиса "свойство–элемент", необходимо создать и определить дочерний элемент компонента Grid. Ставя имя Grid перед именем свойства, мы указываем, что оно принадлежит элементу управления, который относится к более высокому уровню иерархии (как и в случае присоединенных свойств), а задание свойства в виде элемента XAML-файла означает, что мы присваиваем сложное значение указанному свойству элемента управления Grid.

Свойство RowDefinitions получает коллекцию объектов RowDefinitions. Таким образом мы создаем несколько экземпляров объектов RowDefinition, образующих эту

коллекцию. Соответственно свойству ColumnDefinitions присваивается коллекция объектов ColumnDefinition. Для того чтобы продемонстрировать, что ColumnDefinition (как и RowDefinition) — это действительно объект, свойство Width объекта ColumnDefinition было задано в двух первых строках определения столбца.

Для того чтобы поместить элемент управления в заданную ячейку, необходимо снова использовать присоединенные свойства, на этот раз сообщив контейнеру, какие столбец и строку он должен разместить внутри себя.

```
<CheckBox Grid.Column="0" Grid.Row="1" Content="A check box" IsChecked="True" />
```

Еще одним важным контейнерным элементом управления, используемым для компоновки, является StackPanel. Он расставляет элементы управления, содержащиеся в нем, в горизонтальном или вертикальном направлении (в зависимости от значения своего свойства Orientation). Например, если в одной и той же ячейке сетки определены две кнопки (без элемента StackPanel), сетка может разместить вторую кнопку прямо поверх первой. Если же поместить эти кнопки в элемент управления StackPanel, то он выровняет их и разместит одну после другой.

```
<StackPanel Orientation="Horizontal">
    <Button Content="OK" Height="23" Width="75" />
    <Button Content="Cancel" Height="23" Width="75" Margin="10,0,0,0" />
</StackPanel>
```

Конструктор WPF и редактор XAML

По сравнению с версией Visual Studio 2008 конструктор WPF и редактор XAML приобрели множество усовершенствований, включая повышенную надежность (конструктор WPF в системе Visual Studio 2008 очень неустойчив), и, что самое важное, теперь конструктор поддерживает связывание с помощью метода “перетащить и опустить”.

Конструктор WPF по внешнему виду напоминает конструктор Windows Form, но поддерживает больше уникальных свойств. Для того чтобы ближе ознакомить с ними читателей, на рис. 18.4 мы выделили это окно, чтобы рассмотреть его компоненты более подробно.

Во-первых, окно разделено на область визуального конструктора вверху и окно для редактирования кода внизу. Если пользователь предпочитает другое представление, он может щелкнуть на пиктограмме со стрелками, направленными вверх и вниз, которая расположена между корешками вкладок Design и XAML. Вторая пиктограмма на рис. 18.4, *справа*, подсвечена. Это означает, что окно разделено горизонтально. Выбрав пиктограмму, расположенную слева от нее, можно разделить окно по вертикали.

Работать с конструктором WPF в режиме разделения экрана очень удобно, потому что у пользователя есть возможность непосредственно и регулярно модифицировать код XAML, без проблем используя конструктор для решения общих задач.

Если же пользователь не хочет работать в режиме разделения экрана, то он может дважды щелкнуть на корешках вкладок Design или XAML. В этом случае соответствующая вкладка полностью заполнит окно редактирования, как показано на рис. 18.5, и для переключения между ними пользователь сможет поочередно щелкать на корешках вкладок. Для того чтобы вернуться в режим разделения экрана, нужно щелкнуть на пиктограмме Expand Pane, которая расположена на панели разделения правее всех.

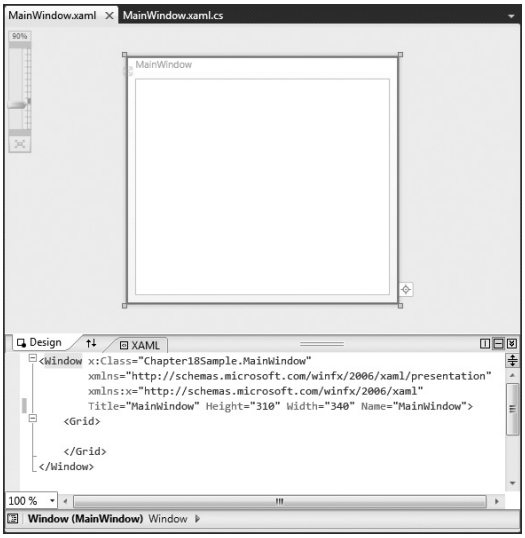

Рис. 18.4

Рис. 18.5

В области конструктора WPF можно заметить элемент управления для изменения масштаба изображения. Он позволяет легко увеличивать или уменьшать масштаб окна или редактируемого элемента управления, что очень удобно при кропотливой настройке мелких деталей, а также для получения общего представления о всем XAML-макете. В данном случае масштаб экрана уменьшен до 90%. Кроме того, на шкале масштабирования существует метка, соответствующая 100%-ному представлению, а также кнопка внизу экрана, позволяющая легко изменять размер XAML-макета, чтобы разворачивать его на всю область конструктора или, наоборот, сворачивать.

Последняя деталь, заслуживающая упоминания, — крошечный трекер, расположенный внизу окна визуального конструктора справа от корешков вкладок Design и XAML. В данном случае форма содержит только один элемент Window, но при добавлении новых элементов в окно этот трекер станет очень полезным для поиска требуемого элемента и перемещения по иерархии элементов управления.

Работа с редактором XAML

Работа с редактором XAML чем-то похожа на работу с редактором HTML в системе Visual Studio. По сравнению с версией Visual Studio 2008 в систему внесено много усовершенствований технологии IntelliSense. Благодаря этому работа с редактором XAML стала очень быстрой и удобной.

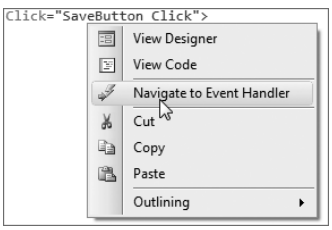

Рис. 18.6

Одной из важнейших особенностей редактора XAML является возможность легко переходить к обработчику события, как только он был назначен для элемента управления. Для этого достаточно щелкнуть правой кнопкой мыши на инструкции присваивания обработчика события в коде XAML и выбрать команду Navigate to Event Handler из раскрывшегося меню, как показано на рис. 18.6.

Работа с конструктором WPF

Несмотря на то что разработчик все же должен владеть языком XAML и уметь работать с редактором XAML, система VS2010 имеет очень хороший конструктор WPF, сравнимый с конструктором Windows Forms, а в некоторых отношениях даже превосходящий его. В этом разделе мы рассмотрим некоторые свойства конструктора WPF.

На рис. 18.7 показаны линии разметки, ориентиры и глифы, которые появляются на экране, когда пользователь выбирает, перемещает или изменяет размер элемента управления. Обратите внимание на глиф, расположенный в правом нижнем углу первого изображения, из представленных на рис. 18.7. Щелкнув на нем, можно легко переключаться из режима, в котором окна имеют фиксированную ширину и высоту, в режим, в котором окна автоматически изменяют размеры, чтобы разместить все свое содержимое. После щелчка глиф изменяется (это означает, что включился режим изменения размеров), а для свойства SizeToContent устанавливается соответствующее значение.

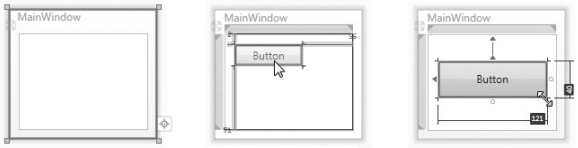

Рис. 18.7

Щелчок на глифе возвращает окно в режим фиксированных размеров. Эта возможность существует только в корневом узле.

Когда пользователь щелкает на глифе, чтобы окно изменило свой размер и вместило содержимое, размер окна не изменяется, потому что его свойства Height и Width заменяются соответствующими свойствами конструктора, так что свойство SizeToContent на проектирование формы не влияет. При возврате в режим фиксированных размеров эти свойства заменяются стандартными свойствами Height и Width.

Второе изображение на рис. 18.7 демонстрирует линии разметки, появляющиеся, когда пользователь перемещает элемент управления по форме (или изменяет его размеры). Аналогичные линии разметки есть в конструкторе Windows Forms. Они помогают выравнивать элементы управления относительно друг друга, а также относительно границ контейнерного элемента управления. Если пользователь не хочет, чтобы отображались линии разметки, но в то же время желает сохранить возможность выравнивания элементов, то при перемещении элементов управления следует удерживать нажатой клавишу <Alt>.

Третье изображение на рис. 18.7 демонстрирует линейки, появляющиеся при изменения размера элемента управления. Эта функциональная возможность позволяет легко увидеть и настроить новые размеры изменяемого элемента управления.

На рис. 18.7 можно заметить некоторые точки привязки (т.е. две стрелки, направленные влево и вверх от кнопки к соответствующим границам контейнерной сетки). Эти стрелки означают, что кнопка будет иметь поля, и задают ее положение в ячейке сетки. В данном случае стрелки означают, что кнопка будет иметь поля слева и сверху, фактически “привязывая” ее верхнюю и левую стороны к левой и верхней сторонам содержащей ее сетки. Однако верхнюю точку привязки легко переключить

так, чтобы она фиксировала кнопку по отношению к нижней границе, а левую точку привязки перенести на правую сторону. Если щелкнуть на стрелке в верхней точке привязки, то она будет перенесена на нижнюю сторону, а если щелкнуть на стрелке в левой точке привязки, то она будет перенесена на правую сторону. Стрелки привязки меняют позицию после каждого щелчка. Кроме того, точки привязки можно устанавливать на обеих сторонах элемента управления сразу (т.е. на левой и правой, а также на верхней и нижней соответственно), так что при изменении размера содержащей его ячейки сетки он будет растягиваться или сжиматься. Например, если левая сторона поля ввода привязана к ячейке сетки, пользователь может также привязать ее к правой стороне, щелкнув на маленьком кружке, расположенном на правой стороне поля ввода. Для того чтобы удалить привязку с какой-то стороны, достаточно щелкнуть на стрелке привязки с этой стороны.

Как указывалось ранее, наиболее важным элементом управления, влияющим на компоновку формы, является Grid. Рассмотрим некоторые специальные возможности конструктора WPF для работы с этим элементом управления. По умолчанию файл `MainWindow.xaml` создается с одним элементом сетки без строк и столбцов. Прежде чем добавлять элементы управления, следует определить некоторые строки и столбцы, с помощью которых можно управлять их компоновкой на форме. Для этого следует начать с выбора сетки, щелкнув на пустой области в середине окна, а затем выбрать соответствующий узел в инструментальном окне Document Outline или установить курсор на соответствующий элемент сетки в самом XAML-файле (в режиме разделения экрана).

При выборе элемента сетки возле левого и верхнего краев сетки появится граница, выделяющая реальную область, занятую сеткой, и относительные размеры каждой строки и столбца, как показано на рис. 18.8. В данном случае на рисунке показана сетка, состоящая из двух строк и двух столбцов.

Добавлять строки и столбцы можно, просто щелкая на границе. После добавления строки или столбца их маркеры можно выделить и перетащить, чтобы задать правильный размер. При первичном размещении маркеров на экране, к сожалению, не высвечивается никакой информации о размерах строки или столбца; однако после создания маркера они появляются.

Когда пользователь перемещает курсор поверх размера строки или столбца, вдоль верхнего края сетки появляются пиктограммы (рис. 18.9).

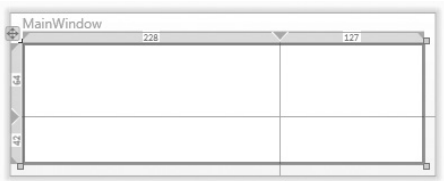

Рис. 18.8

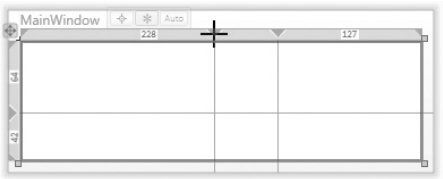

Рис. 18.9

Они позволяют легко указать, что размер строки или столбца должен быть фиксированным (#), пропорциональным (*) или определяться в зависимости от содержимого (Auto).

Для того чтобы удалить строку или столбец, необходимо щелкнуть на них и перетащить за пределы сетки. В этом случае строка или столбец будут удалены, а элементы управления в окружающих ячейках будут изменены соответствующим образом.

Пропорциональный размер (weighted proportion) означает, что пользователь должен задать процент доступного пространства (по сравнению с другими столбцами). После того как строки и столбцы с фиксированными и автоматическими размерами займут свои места, остальное пространство будет разделено между строками и столбцами с пропорциональными размерами. Если пользователь не указал после префикса в виде звездочки числовой коэффициент, то свободное место разделяется между строками поровну. Допустим, например, что пользователь задал сетку с шириной, равной 1000 пикселей, и двумя столбцами. Если ширина каждого из них задана символом `*`, то каждый из них будет иметь ширину, равную 500 пикселям. Однако, если ширина одного из них задана символом `*`, а ширина другого — `3*`, то 1000 пикселей будут разделены на порции размером 250 пикселей, причем одна порция будет выделена первому столбцу (т.е. его ширина будет равна 250 пикселям), а остальные три — второму (т.е. его ширина будет равна 750 пикселям).

При создании элемента управления путем перетаскивания его в ячейку сетки следует "привязать" его к левому и верхнему краям ячейки (т.е. перетаскивать его до тех пор, пока он не будет зафиксирован в этой позиции). В противном случае поля элемента управления будут определяться его фактическим положением в ячейке, а это иногда не соответствует желаниям пользователя.

Инструментальное окно Properties

Помещая элемент управления на форму, необязательно возвращаться к редактору XAML, чтобы задать значения свойств и назначить обработчики событий. Как и в системе Windows Forms, в системе WPF есть окно Properties, которое, впрочем, имеет некоторые особенности (рис. 18.10).

По сравнению с системой Visual Studio 2008 окно Properties приобрело массу новый функциональных возможностей. В системе Visual Studio 2008 его возможности были очень ограничены. В результате во многих случаях пользователь был вынужден непосредственно модифицировать код XAML. Однако в системе Visual Studio 2010 возможности окна Properties были расширены, уменьшив необходимость редактировать XAML-файлы.

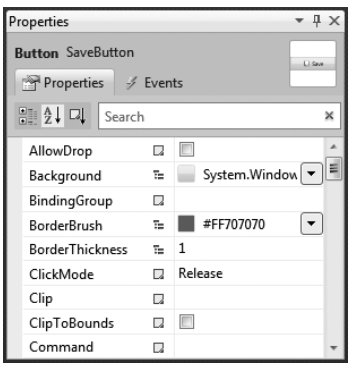

Рис. 18.10

Инструментальное окно Properties для разработки форм по технологии Windows Forms позволяет выбирать элемент управления и задавать его свойства с помощью раскрывающегося списка, расположенного над списком свойств и событий. Однако в окне Properties системы WPF этого списка нет. Теперь пользователь должен выбирать элементы управления в конструкторе, выделяя их в окне Document Outline или устанавливая курсор в область определения элемента управления в XAML-представлении. При этом в левом верхнем углу окна появляется маленький эскиз элемента управления (и всех элементов управления, которые в нем содержатся), за которым следует его квалифицированный тип.

Окно Properties можно использовать, работая с редактором XAML и конструктором одновременно. Однако, если пользователь работает с редактором XAML, конструктор должен быть загружен (если открыть файл непосредственно в редакторе XAML, может возникнуть необходимость переходить в рабочую область конструктора и назад). Если же XAML-файл является некорректным, то сначала придется исправить все ошибки.

Свойство Name элемента управления не входит в список свойств, но для него предусмотрено отдельное поле ввода, расположенное над этим списком. Обратите внимание на то, что если элемент управления еще не имеет имени, то оно присваивается как значение свойства Name (а не x:Name). Однако, если атрибут x:Name для элемента управления уже определен и пользователь изменяет его имя с помощью окна Properties, этот атрибут будет обновлен.

Элементы управления могут иметь много свойств и событий, так что их поиск в списках свойств и событий в системе Windows Forms может оказаться утомительным. Для того чтобы облегчить поиск конкретного свойства, в окне Properties системы WPF предусмотрена функция поиска, которая динамически фильтрует список свойств по тексту, который пользователь вводит в поле ввода. Поисковая строка, которую вводит пользователь, не обязательно должна начинаться с имени свойства или события. Свойство или событие будет найдено, даже если в поисковой строке указана только часть его имени. К сожалению, эта поисковая функция не поддерживает поиск имен, набранных в "верблюжьем" стиле.

Список свойств в конструкторе WPF (аналогично системе Windows Forms) может выводиться как в алфавитном порядке (Alphabetical order), так и по категориям (Categorized order). Отметим, что ни одно свойство, являющееся объектом (например, Margin), не может быть развернуто для демонстрации или редактирования (как это делалось в системе Windows Forms). Если список выводится по категориям, то пользователь может использовать новую и уникальную функциональную возможность окна свойств системы WPF: редакторы категорий. Например, если выбрать элемент управления Button и перейти к категории Text, можно найти специальный редактор для свойств, относящихся к категории Text, и задать их новые значения, как показано на рис. 18.11

В списке свойств также выводятся присоединенные свойства элементов управления, как показано на рис. 18.12.

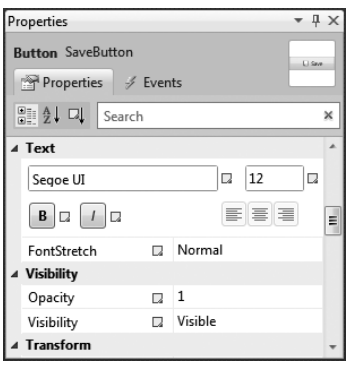

Рис. 18.11

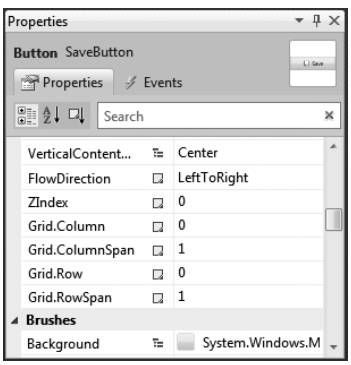

Рис. 18.12

Легко заметить, что справа от имени каждого свойства располагается маленькая пиктограмма. Эта новая функциональная возможность системы Visual Studio 2010 называется *маркерами свойств* (property markers). Эти маркеры указывают, что является источником значения свойства. Когда курсор мыши находится над маркером, на экране появляется подсказка, описывающая смысл свойства. Если источник значения изменяется, то маркер изменяется вместе с ним. Опишем пиктограммы, показанные на рис. 18.13.

- Светло-серая пиктограмма означает, что для свойства еще не задано никакого значения и поэтому оно будет использовать значение, заданное по умолчанию.
- Черный ромбик означает, что для свойства установлено локальное значение (т.е. конкретное значение).
- Желтый цилиндр означает, что для свойства задано выражение, связывающее данные (это понятие будет рассмотрено далее в этой главе).
- Кисточка (с зеленой краской) означает, что для свойства установлен какой-то ресурс.
- Пурпурная иерархия в виде дерева означает, свойство наследовало свое значение у другого элемента управления, который расположен дальше по иерархии.

Щелчок на маркере свойства открывает меню, в котором есть дополнительные команды для присваивания значений данному свойству (рис. 18.14).

Рис. 18.13

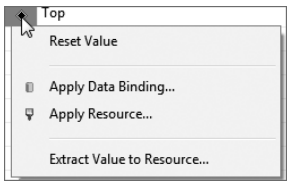

Рис. 18.14

Команда Reset Value просто устанавливает значение свойства равным значению, заданному по умолчанию (удаляя атрибут, присваивающий значения в XAML-файле).

Команда Apply Data Binding вызывает редактор, позволяющий выбирать разные варианты связывания для создания выражений, связывающих данные и используемых в качестве значения свойства. Система WPF поддерживает множество вариантов связывания. Соответствующие окна будут рассмотрены в следующем разделе.

Команда Apply Resource позволяет выбирать ресурс, созданный пользователем или определенный системой WPF, и присваивать его в качестве значения указанному свойству. Ресурсы — это повторно используемые объекты и значения. Эта концепция близка понятию констант в коде. На рис. 18.15 показано всплывающее окно, которое появляется при выборе этой команды.

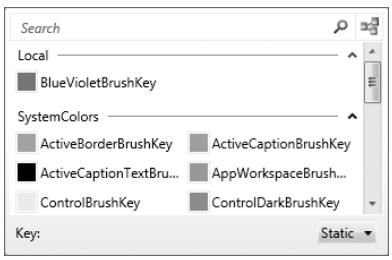

Рис. 18.15

Ресурсами являются все ресурсы, доступные для данного свойства (т.е. находящиеся в его области видимости и имеющие одинаковые типы) и сгруппированные по словарям ресурсов. Обратите внимание на пиктограмму, расположенную в правом верхнем углу всплывающего окна. Щелчок на этой пиктограмме

также группирует ресурсы по XAML-файлам, в которых они определены. Эту опцию можно включать и выключать.

На рис. 18.15 показан ресурс, тип которого совпадает с типом свойства (BlueVioletBrushKey), определенного в текущем XAML-файле (в группе Local). Поскольку это свойство имеет тип SolidColorBrush, в окне продемонстрированы все ресурсы для цвета кисточки, определенные заранее в системе WPF и доступные для выбора.

Вернемся к другим командам из меню, показанного на рис. 18.14. Команда Extract Value to Resource принимает значение данного свойства и возвращает его ресурсу. Этот ресурс создается как ресурс корневого узла в XAML-файле, так что его можно повторно использовать в файле с помощью присвоенного ему уникального ключа. Значение свойства автоматически обновляется так, чтобы использовать этот ресурс. Например, применяя эту команду к свойству Background элемента управления, в котором значение `#FF8888B7` определяет следующий ресурс в элементе `Window.Resources` с именем `BlueVioletBrushKey`.

```
<SolidColorBrush x:Key="BlueVioletBrushKey"> #FF8888B7 </SolidColorBrush>
```

Элемент управления будет ссылаться на этот ресурс следующим образом.

```
Background="{StaticResource BlueVioletBrushKey}"
```

Затем этот ресурс можно применять к другим элементам управления, используя те же самые средства языка XAML или указывая элемент управления и свойство, к которому применяется ресурс, а затем выбирая команду Apply Resource из меню, описанного ранее.

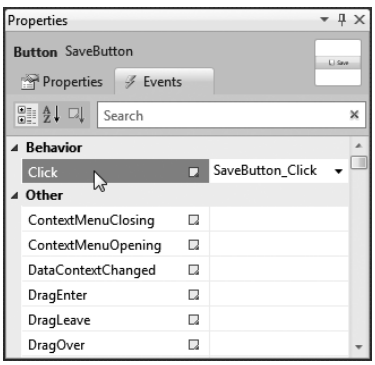

Рис. 18.16

В конструкторе WPF (как и в системе Windows Forms) двойной щелчок на элементе управления автоматически создает обработчик события, заданного для данного элемента по умолчанию в исходном коде. Кроме того, можно создать обработчик события для любых событий, связанных с элементом управления, используя окно Properties, как в системе Windows Forms. Щелкнув на пиктограмме, подсвеченной в окне Properties, можно перейти в вкладку Events, как показано на рис. 18.16. На ней выводится список событий, которые могут происходить с элементом управления. Дважды щелкнув на событии, пользователь может автоматически создать обработчик события в исходном коде.

Разработчики проектов на языке VB.NET, дважды щелкая на элементе управления Button или создавая события с помощью окна Properties, связывают их с помощью синтаксической конструкции Handles. Следовательно, в этом случае обработчик события не присваивается событию как атрибут. Пользователь, применяющий этот метод обработки события, не может видеть определения обработчика события в XAML-файле данного элемента управления, а значит, не может использовать меню Navigate to Event Handler (см. рис. 18.6) в редакторе XAML.

Привязка данных

Привязка данных — одна из очень важных и сильных концепций в системе WPF. Вначале синтаксис привязки данных может отпугнуть, но система Visual Studio 2010 делает создание форм для привязки данных очень легким делом. Система Visual Studio 2010 помогает привязывать данные двумя способами: с помощью команды Apply Data Binding, применяемой к свойству в инструментальном окне Properties, и с помощью перетаскивания средств поддержки привязки данных из окна Data Sources. В этом разделе мы по очереди рассмотрим обе эти возможности.

В системе WPF с элементами управления можно связывать объекты (к которым относятся наборы данных, сущности технологии ADO.NET Entity Framework и т.п.), ресурсы и даже свойства других элементов. Система WPF имеет очень богатые возможности для привязки, позволяющие связывать свойства с чем угодно. Ручное программирование сложных выражений для привязки данных на языке XAML может показаться слишком утомительным, но редактор Apply Data Binding позволяет строить такие выражения с помощью интерфейса "укажи и щелкни".

Для того чтобы связать свойство с элементом управления, сначала необходимо найти требуемый элемент управления в конструкторе и соответствующее свойство в окне Properties. Затем нужно щелкнуть на маркере свойства и выбрать команду Apply Data Binding. Окно, которое откроется после этого, показано на рис. 18.17.

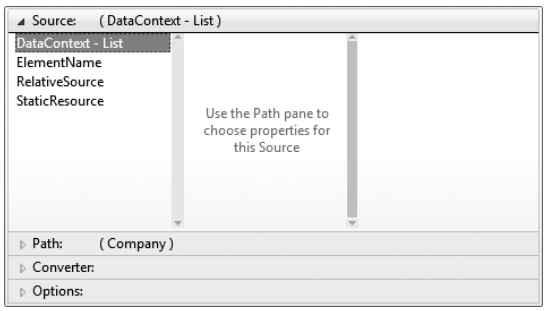

Рис. 18.17

Это окно содержит длинный список этапов (аналогично мастеру), которые необходимо пройти, чтобы осуществить привязку, — Source, Path, Converter и Options, в зависимости от стиля компоновки. Для того чтобы начать новый этап, необходимо щелкнуть на его заголовке.

Обычно, когда пользователь открывает это окно, система предлагает ему выполнить этап Source, позволяющий выбрать источник привязки (иначе говоря, источник данных, с которым необходимо связать элемент управления). Обратите внимание на то, что этот этап можно автоматически пропустить и перейти на этап Path (рис. 18.18), если источник данных уже был связан с элементом управления ранее (или находится выше в иерархии). Если пользовать желает выбрать один из типов привязки (например, ElementName), достаточно выбрать заголовок этапа Source, чтобы изменить ранее заданный источник привязки. Затем необходимо выполнить другие команды привязки (т.е. выбрать команду и перейти к следующему окну).

Рис. 18.18

В примере, показанном на рис. 18.17, на вершине иерархии находится элемент управления Grid, у которого свойству DataContext был присвоен ресурс CollectionViewSource (ссылающийся на объект ViewModel как на источник данных). Значение свойства DataContext наследуется элементами управления, находящимися на более низких уровнях иерархии, поэтому, когда к текстовому полю внутри сетки применяется привязка данных, пользователь может указать, что источником привязки является свойство DataContext текстового поля (в окне это выглядит как присваивание элемента List этому свойству). Выбрав источник привязки, можно переходить на этап Path.

Этап Path позволяет выбрать путь к источнику привязки, содержащему искомое значение. Например, на рис. 18.18 выбрано свойство Company (относящееся к объекту ViewModel, на который ссылается источник связывания).

Если свойство само по себе является объектом, то в нем можно выбрать свойство, к которому следует применить привязку, и т.д. Как показано на рис. 18.18, свойство Company (строка) имеет свойство Length, с которым по желанию можно связать элемент управления.

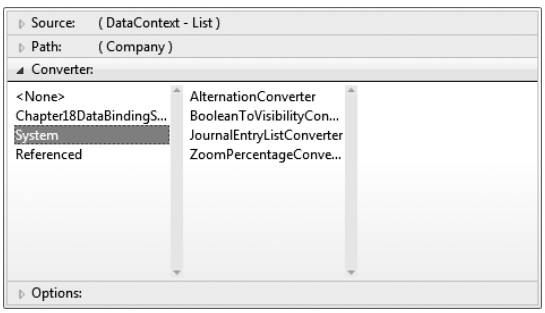

Рис. 18.19

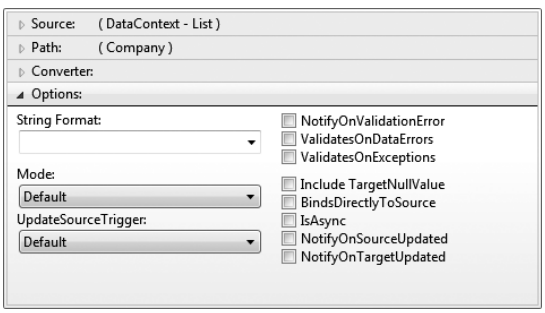

Рис. 18.20

Для того чтобы закрыть редактор, следует дважды щелкнуть на своем последнем выборе. По необходимости можно вызвать конвертер (рис. 18.19), который преобразует связываемое значение перед присваиванием его выбранному значению и перед тем, как значение будет возвращено обратно связываемому свойству (очень мощное средство привязки данных в системе WPF).

Команда Options позволяет также задать другие варианты привязки, показанные на рис. 18.20.

Как видим, этот конструктор намного облегчает процесс создания выражений для привязки, не требуя от пользователя изучения синтаксиса привязки данных. Впрочем, овладеть этим синтаксисом очень полезно, поскольку тогда можно понять выражения, создаваемые в XAML-файле.

Рассмотрим возможности привязки данных с помощью механизма перетаскивания в системе Visual Studio 2010. На первом этапе необходимо создать то, с чем будут связываться данные. Это может быть объект, набор данных или сущность технологии ADO.NET Entity Framework, а также многое другое. Для примера создадим объект. Затем создадим новый класс в проекте с именем `ContactViewModel` и несколько свойств этого класса: FirstName, LastName, Company, Phone, Fax, Mobile и Email (все они являются строками).

Объект назван `ContactViewModel`, потому что он действует как объект `ViewModel`, который имеет отношение к шаблону Model-View-ViewModel (MVVM), упомянутому ранее. Впрочем, этот шаблон проектирования не будет реализован в полном объеме, чтобы не усложнять пример.

Теперь скомпилируем проект (это важно, иначе класс на следующем этапе не появится). Вернитесь к конструктору формы и выберите команду Add New Data Source в меню Data. Затем выберите Object как тип источника данных, щелкните на кнопке Next и выберите класс ContactViewModel из дерева (для того, чтобы найти этот класс, необходимо развернуть узлы иерархии пространства имен). Щелкните на кнопке Finish. Откроется инструментальное окно Data Sources, в котором указан объект ContactViewModel, а ниже перечисляются его свойства, как показано на рис. 18.21.

Перетащите на форму весь объект или его отдельные свойства. В результате возникнет один или несколько элементов управления для демонстрации данных. По умолчанию для демонстрации данных будет создан элемент управления DataGrid, но если пользователь выберет команду ContactViewModel, то появится кнопка, которая в ответ на щелчок откроет меню (как показано на рис. 18.22), предоставляя пользователю возможность выбрать команды DataGrid, List и Details.

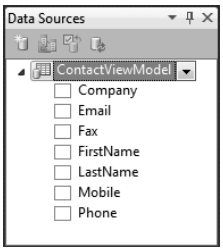

Рис. 18.21

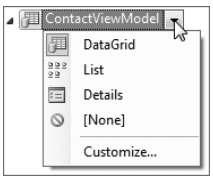

Рис. 18.22

- Команда DataGrid создает элемент управления DataGrid, в котором для каждого свойства объекта предусмотрен отдельный столбец.
- Команда List создает элемент управления List с шаблоном данных, содержащих поля для каждого свойства.
- Команда Details создает элемент управления Grid с двумя столбцами: один для меток, а другой для полей. Для каждого свойства объекта будет создана строка и элемент управления Label, демонстрирующий имя поля (с пробелами, вставленными перед прописными буквами) в первом столбце и само поле (тип которого зависит от типа данных свойства) во втором столбце.

В свойстве Resources элемента управления Window создается ресурс, ссылающийся на объект ContactViewModel, который можно использовать как контекст данных или источник элементов управления, связанных с объектом. Если пользователь планирует задавать источник данных в коде, то на последующих стадиях этот ресурс можно удалить. Элементам управления необходимо выражение для привязки данных. Тип элемента управления, который будет создан на форме для демонстрации данных, зависит от выбора элемента ContactViewModel.

Тип элемента управления, созданного для каждого свойства, по умолчанию является производным от типа данных этого свойства, но, как и в случае элемента Con-

tactViewModel, щелкнув на кнопке, пользователь может выбрать в меню другой тип управляющего элемента (рис. 18.23). Если требуемого типа элемента в списке нет (например, если необходимо использовать управляющий элемент, разработанный сторонним поставщиком), то можно выбрать команду Customize и добавить в список соответствующий тип данных. Если же пользователь не хочет создавать поле для свойства, то необходимо выбрать команду None.

В данном примере мы создадим *форму сведений* (details form), поэтому в окне Data Sources в узле ContactViewModel выберите команду Details. По желанию можно изменить элемент управления, генерируемый для каждого свойства, но пока мы будем генерировать для каждого свойства поле ввода и сгенерируем свойства в форме сведений. Выберите узел ContactViewModel в окне Data Sources и перетащите его на форму. Вместе с полями, соответствующими каждому свойству, будет создана сетка, показанная на рис. 18.24

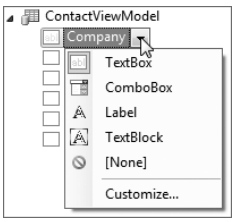

Рис. 18.23

Рис. 18.24

К сожалению, в окне Data Sources нет возможности определить порядок следования полей в форме, поэтому элементы управления придется упорядочивать на сетке вручную (либо с помощью визуального конструктора, либо непосредственно редактируя XAML-файл).

Просмотрев XAML-файл, сгенерированный при осуществлении привязки данных с помощью перетаскивания, легко убедиться, что мы сэкономили много времени и сделали все намного проще и быстрее.

Если в проекте есть элементы управления, имеющие свойства, которым можно присваивать выражения для привязки данных, необходимо создать их свойства зависимости. Свойства зависимости — это особая сущность в технологиях WPF и Silverlight, значением которой может служить вычисляемое выражение (например, выражение для привязки данных). Определение свойств зависимости должно отличаться от определения стандартных свойств. Обсуждение этой концепции выходит за рамки рассмотрения данной главы, но, по существу, только свойствам, определенным как свойства зависимости, можно присваивать выражения для привязки данных.

Выбор стиля приложения

До сих пор наше приложение выглядело слишком простым — проще, чем обычные приложения, создаваемые с помощью системы Windows Forms. Однако огромным преимуществом технологии WPF является легкость, с которой можно изменять внешний вид элементов управления. Это позволяет полностью изменять стиль приложения. Часто используемые изменения можно хранить в виде специальных элементов управления — стилей (коллекции значений свойств элементов управления, хранящейся в виде ресурса, который можно определить один раз и применять ко многим элементам управления) — или полностью переопределять в XAML-файлах, соответствующих элементам управления, создавая новые шаблоны элементов управления. Эти ресурсы можно определить в свойстве Resources любого элемента управления вместе с ключом, который можно применять к любому элементу управления, расположенному на более низких уровнях иерархии, и ссылаться на них с помощью этого ключа. Например, если требуется определить ресурс, доступный для любого элемента управления, существующего в XAML-файле `MainWindow`, то его можно определить в элементе Window.Resources. Если же пользователь хочет иметь возможность использовать его во всем приложении, то он может определить его в свойстве Application.Resources элемента Application в файле `App.xaml`.

Более того, в словаре ресурсов можно определить несколько шаблонов и стилей элементов управления и использовать их как *тему* (theme). Эту тему можно применять ко всему приложению и автоматически настраивать стили пользовательского интерфейса, обеспечивая уникальный и согласованный внешний вид своего приложения. Именно этому посвящен данный раздел. Впрочем, пользователь может не создавать свои собственные темы, а загрузить их на сайте CodePlex со страницы, посвященной проекту WPF Themes (`http://www.codeplex.com/wpfthemes`).

Эти темы изначально были разработаны (*б*ольшая часть — компанией Microsoft) для использования в приложениях технологии Silverlight, но затем они были преобразованы (так, где это было необходимо) для применения к приложениям технологии WPF. Используем одну из этих тем для создания совершенно иного внешнего вида нашего приложения. Создадим новое приложение и добавим на форму несколько разных элементов управления, как показано на рис. 18.25.

Как видите, элементы выглядят очень невыразительно, поэтому мы применим тему и легко полностью изменим внешний вид формы. Загружая тему с веб-страницы проекта WPF Themes, можно заметить, что он содержит решение, состоящее из двух проектов: один реализует темы, а второй демонстрирует результаты использования первого. Однако мы используем темы несколько иначе. Запустим простое приложение и выберем тему, которая нам нравится. Для демонстрации возьмем тему Shiny Blue. В проекте `WPF.Themes`, находящемся в папке `ShinyBlue`, можно найти файл `Theme.xaml`. Скопируйте его в корневой узел нашего проекта (чтобы гарантировать, что он будет включен в наш проект в системе Visual Studio).

Откройте файл `App.xaml` и добавьте следующий код в раздел `Application.Resources`.

```
<ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="Theme.xaml"/>
    </ResourceDictionary.MergedDictionaries>
</ResourceDictionary>
```

Этот код на языке XAML просто объединяет ресурсы из файлов темы с ресурсами приложения. Теперь эти ресурсы будут применяться ко всему приложению, замещая стили элементов управления, заданные по умолчанию, соответствующими стилями из файла тем.

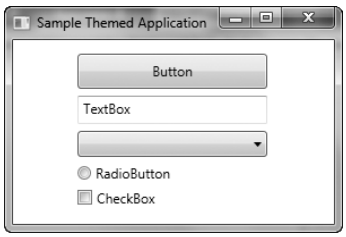

Рис. 18.25

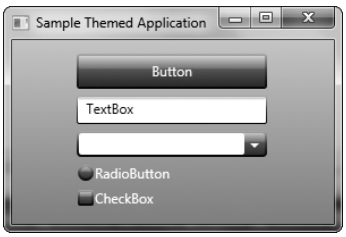

Рис. 18.26

Последнее изменение касается стиля фона наших окон на стиль из файла тем (он не присваивается автоматически). Добавьте в элемент Window атрибут

```
Background="{StaticResource WindowBackgroundBrush}"
```

Запустите проект. Теперь все элементы управления выглядят совершенно иначе, как показано на рис. 18.26.

Для того чтобы изменить тему, достаточно заменить файл `Theme.xaml` другим файлом из проекта WPF.Themes и перекомпилировать проект.

При частом изменении стилей и шаблонов элементов управления в приложениях намного проще использовать программу Expression Blend — инструмент, специально разработанный для разработчиков графических приложений, работающих с языком XAML. Программа Expression Blend намного лучше приспособлена для разработки графики и анимации, чем система Visual Studio. Программа Expression Blend может открывать решения, созданные в системе Visual Studio, а также просматривать, редактировать и компилировать проекты, хотя сама она намного лучше подходит для задач проектирования. Эта интеграция системы Visual Studio и программы Expression Blend позволяет поддерживать технологический процесс проектирования и разработки. Обе программы могут одновременно открывать одно и то же решение или проект (даже на одном и том же компьютере), позволяя пользователю быстро переключаться между ними по мере необходимости. Если файл был открыт одной из этих программ, то при сохранении изменений в файле, выполняемом другой программой, откроется диалоговое окно, в котором пользователю будет задан вопрос о том, не хочет ли он перезагрузить файл. Для того чтобы легко открыть решение, созданное в программе Expression Blend, находясь в системе Visual Studio, пользователь должен щелкнуть правой кнопкой мыши на XAML-файле и выбрать команду Open in Expression Blend.

Взаимодействие с системой Windows Forms

До сих пор мы описывали процесс создания приложения в графической системе WPF, однако вполне вероятно, что пользователь уже имеет большой опыт в разработ-

ке приложений с помощью системы Windows Forms и вряд ли захочет немедленно перейти на новую технологию. У него есть возможность сохранить свои наработки и не переписывать их в погоне за новыми технологиями. Для того чтобы облегчить переход на новую технологию, компания Microsoft сохранила возможность одновременно использовать технологии WPF и Windows Forms при создании одного и того же приложения. Обоюдное взаимодействие поддерживается как приложениями WPF, так и приложениями Windows Forms, причем элементы управления системы WPF могут использоваться в приложениях, созданных по технологии Windows Forms, и наоборот. В настоящем разделе мы покажем, как реализовать эти сценарии.

Использование элементов управления WPF Control в приложениях Windows Forms

Для начала создадим новый проект в своем решении, чтобы внедрить в него элемент управления WPF. Этим элементом (в целях демонстрации) будут поля для ввода имени и пароля пользователя. В диалоговом окне Add New Project (рис. 18.27) выберите шаблон проекта WPF User Control Library. Он уже содержит XAML-файл и исходный файл, необходимые для элемента управления в системе WPF. Если проверить XAML-код, соответствующий данному элементу управления, то можно увидеть, что он, по существу, совпадает с исходным XAML-кодом для окна, рассмотренного в начале главы, за исключением того, что корневым XAML-элементом является элемент UserControl, а не Window.

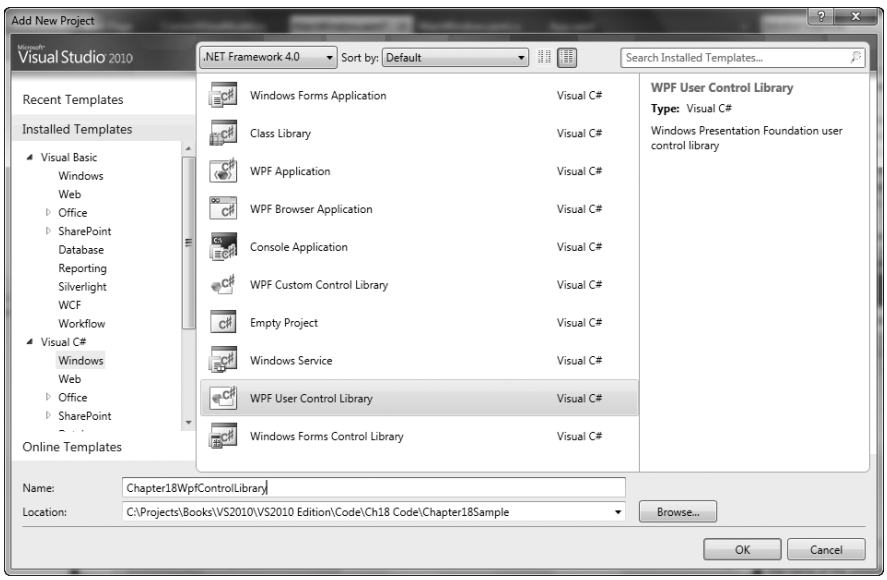

Рис. 18.27

Переименуйте элемент управления в UserLoginControl и добавьте сетку, два текстовых блока и два поля для ввода, как показано на рис. 18.28.

Рис. 18.28

В исходный файл добавьте несколько простых свойств, чтобы открыто обнародовать содержимое текстовых блоков (получатель и отправитель).

VB

```
Public Property UserName As String
    Get
        Return txtUserName.Text
    End Get
    Set(ByVal value As String)
        txtUserName.Text = value
    End Set
End Property

Public Property Password As String
    Get
        Return txtPassword.Text
    End Get
    Set(ByVal value As String)
        txtPassword.Text = value
    End Set
End Property
```

C#

```
public string Username
{
    get { return txtUserName.Text; }
    set { txtUserName.Text = value; }
}

public string Password
{
    get { return txtPassword.Text; }
    set { txtPassword.Text = value; }
}
```

Теперь, когда у вас есть свой собственный элемент управления, созданный по технологии WPF, соберите проект и создайте новый проект типа Windows Forms, который будет содержать ваш элемент управления. Создайте проект и добавьте в него ссылку на WPF-проект, содержащий элемент управления (используйте команду Add Reference из меню, которое открывается после щелчка правой кнопкой мыши на заголовке References).

Откройте форму, которая будет содержать элемент управления WPF. Поскольку библиотека создаваемых пользователем элементов управления WPF находится в том же самом решении, элемент UserLoginControl появится в окне Toolbox и может быть просто перетащен на форму. В результате будет автоматически добавлен элемент управления ElementHost (который может содержать другие элементы управления WPF), содержащий элемент UserLoginControl.

Однако по необходимости это можно сделать вручную. Для этого необходимо выполнить следующие действия. В инструментальном окне Toolbox содержится вкладка WPF Interoperability, в которой находится отдельный элемент ElementHost. Перетащите его на форму, как показано на рис. 18.29, и вы увидите интеллектуальные дескрипторы, предлагающие пользователю выбрать требуемый элемент управления. Если элемент управления не появляется в раскрывающемся списке, то пользователю придется создать свое решение.

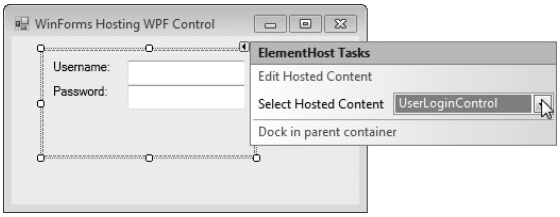

Рис. 18.29

Элемент управления можно загрузить в элемент управления ElementHost. При этом он автоматически получит имя в коде, который будет изменен с помощью свойства HostedContentName.

Использование элементов управления Windows Forms Control в приложениях WPF

Рассмотрим теперь противоположный сценарий — использование элементов управления Windows Forms Control в приложениях WPF. Создайте новый проект `Chapter 18 WinFormsControlLibrary`, используя шаблон проекта Class Library. Удалите класс `Class1` и добавьте в этот проект новый элемент управления User Control с именем `UserLoginControl`.

Откройте этот элемент в визуальном конструкторе и добавьте два текстовых блока и два поля ввода, как показано на рис. 18.30.

Рис. 18.30

В исходный код добавьте несколько простых свойств, чтобы продемонстрировать содержимое текстовых блоков и открыто обнародовать содержимое текстовых блоков (получатель и отправитель).

VB

```
Public Property UserName As String
    Get
        Return txtUserName.Text
    End Get
    Set(ByVal value As String)
        txtUserName.Text = value
    End Set
End Property

Public Property Password As String
    Get
        Return txtPassword.Text
    End Get
    Set(ByVal value As String)
        txtPassword.Text = value
    End Set
End Property
```

C#

```
public string Username
{
    get { return txtUserName.Text; }
    set { txtUserName.Text = value; }
}
```

```
public string Password
{
    get { return txtPassword.Text; }
    set { txtPassword.Text = value; }
}
```

Теперь, когда у вас есть свой собственный элемент управления, созданный по технологии Windows Forms, соберите проект и создайте новый проект типа WPF, который будет содержать наш элемент управления. Создайте проект и добавьте в него ссылку на проект Windows Forms, содержащий элемент управления (используйте команду Add Reference из меню, которое открывается после щелчка правой кнопкой мыши на заголовке References).

Откройте форму, которая будет содержать элемент управления Windows Forms, в визуальном конструкторе. Выберите элемент управления WindowsFormsHost в инструментальном окне Toolbox и перетащите его на форму. К сожалению, при этом визуальный конструктор не может оказать пользователю никакой помощи, и приходится вносить изменения в редакторе XAML editor. Необходимо добавить префикс пространства имен в определение корневого элемента.

```
xmlsn:wfapp="clr-namespace:Chapter18WinFormsControlLibrary;
assembly=Chapter18WinFormsControlLibrary"
```

Модифицируйте элемент WindowsFormsHost, который будет содержать наш элемент управления и перерисовывать его при выполнении приложения, как показано на рис. 18.31.

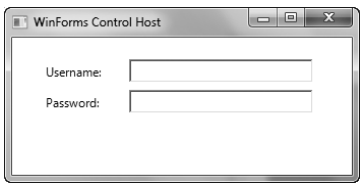

Рис. 18.31

```
<WindowsFormsHost x:Name="windowsFormsHost">
    <wfapp:UserLoginControl x:Name="userLoginDetails" />
</WindowsFormsHost>
```

Отладка с помощью визуализатора WPF

Найти причину ошибки, возникшей на этапе управления, просматривая дерево XAML или визуальное дерево, очень сложно, но, к счастью, в системе VS2010 появилась новая функциональная возможность под названием WPF Visualize, которая помогает отлаживать визуальное дерево WPF-приложения. Например, элемент может оказаться невидимым, хотя должен быть видимым, может появиться не там, где задумано, или иметь неправильный стиль. Визуализатор WPF Visualizer помогает отслеживать такие проблемы, позволяя пользователю просматривать визуальное дерево, просматривать значения свойств выделенного элемента, а также отслеживать источники стилей для свойств.

Для того чтобы открыть визуализатор WPF Visualizer, перейдите в режим прерывания. Затем, используя инструментальное окно Autos, Local или Watch, найдите переменную, которая содержит ссылку на элемент, в XAML-документе для отладки. После этого щелкните на пиктограмме с изображением увеличительного стекла, распо-

ложенной рядом с элементом пользовательского интерфейса WPF, указанным в инструментальном окне, и откройте визуализатор (рис. 18.32). В качестве альтернативы можно поместить курсор на переменную, соответствующую элементу пользовательского интерфейса WPF, чтобы открыть окно DataTip, и щелкнуть на пиктограмме с изображением увеличительного стекла.

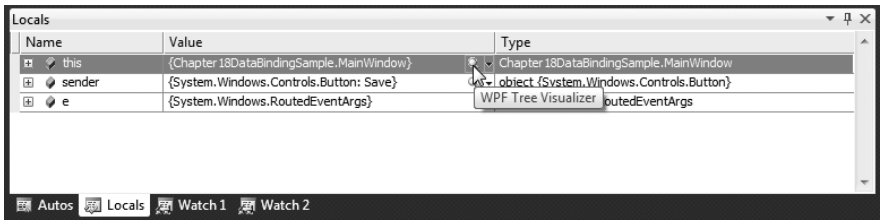

Рис. 18.32

Визуализатор WPF Visualizer показан на рис. 18.33. В левой части окна мы видим визуальное дерево для текущего XAML-документа и прорисовку выбранного элемента, расположенного в дереве под ним. В правой части приведен список всех свойств выбранного элемента дерева, его текущие значения и другая информация о каждом свойстве.

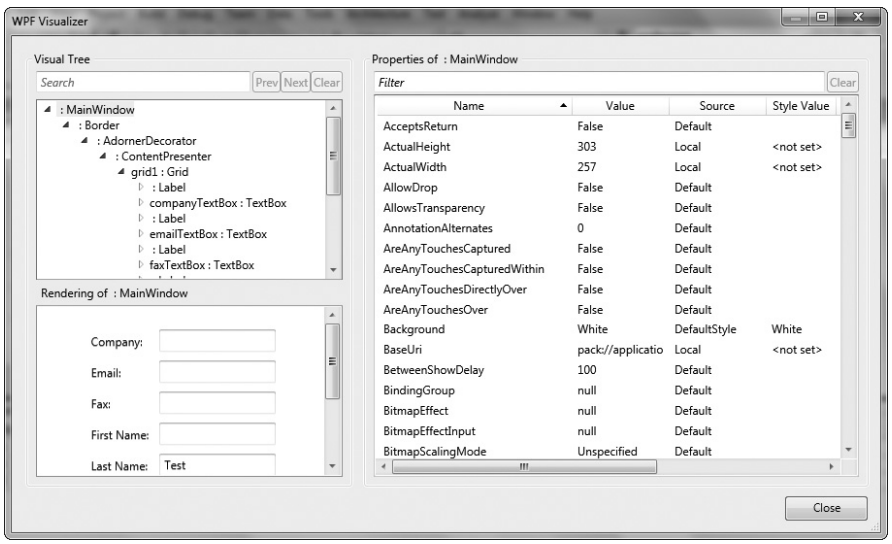

Рис. 18.33

Поскольку визуальное дерево может содержать тысячи элементов, поиск заданного элемента путем перебора может оказаться слишком трудным делом. Если известно имя или тип искомого элемента, то можно ввести его в поле поиска, расположенное над деревом, и пройти по соответствующим узлам, используя кнопки Next и Prev. Можно также отфильтровать список свойств, введя часть имени, значения или типа искомого элемента.

К сожалению, в системе Visual Studio 2010 нет средств для редактирования значения свойства или модификации дерева свойств, но анализ элементов визуального де-

рева и значений их свойств (а также источников этих значений) должен помочь решить проблемы, связанные с XAML, намного проще, чем в предыдущих версиях системы Visual Studio.

Резюме

В этой главе показано, как с помощью системы Visual Studio 2010 создать приложения по технологии WPF. Мы описали несколько важных концепций языка XAML, показали, как использовать уникальные свойства конструктора WPF, а также продемонстрировали способы изменения внешнего вида приложений и возможности взаимодействия систем WPF и Windows Forms.