Байдачный С. С.

«СОЛОН»

Silverlight 4:

Создание насыщенных Web-приложений

- Создание Web-приложений нового поколения
- Взаимодействие с объектной моделью браузера
- Управление шаблонами и стилями
- Использование графики и анимации
- Создание собственных элементов управления
- Взаимодействие с данными
- Использование медиа-элементов и Deep Zoom
- Отладка и тестирование Silverlight-приложений
- Интеграция с SharePoint 2010

Библиотека Профессионала

УДК 621.396.218
ББК 32.884.1
Б18

Байдачный С. С.
Б18 Silverlight 4: Создание насыщенных Web-приложений — М.: СОЛОН-ПРЕСС, 2010. — 288 с.: ил. — (Серия «Библиотека профессионала»).

ISBN 978-5-91359-079-4

Silverlight 4 — новая технология от Microsoft, предназначенная для разработки насыщенных Web-приложений, или приложений с «богатым» интерфейсом. Основные характеристики Silverlight-приложений — это интенсивное использование графики, анимации, работа с медиа-файлами, а также эффективное взаимодействие с данными и серверными компонентами. При этом разработчик имеет возможность не только использовать управляемые языки программирования (C#, VB.NET) для разработки Silverlight-приложений, но и получить доступ к большинству преимуществ, доступных в .NET Framework. Если взять во внимание, что процесс разработки Silverlight-приложений тесно интегрирован в Visual Studio, то можно утверждать, что использование Silverlight не вызовет затруднений у существующих .NET разработчиков.

Данная книга может быть полезна для всех, кто решил изучить Silverlight 4 и уже имеет общие познания в разработке приложений на платформе .NET.

УДК 621.396.218
ББК 32.884.1

По вопросам приобретения обращаться:
Тел: (495) 254-44-10, (499) 795-73-26

Сайт издательства СОЛОН-ПРЕСС: **www.solon-press.ru**
E-mail: **kniga@coba.ru**

ISBN 978-5-91359-079-4

Глава 1

ВВЕДЕНИЕ В SILVERLIGHT 4

Наверняка, если Вы уже сталкивались с Silverlight, то интересуетесь новыми возможностями, которые появились в четвертой версии. Поэтому, первая глава посвящена обзору Silverlight 4. Тут описаны практически все новые возможности технологии, с которыми мы будем встречаться по ходу книги.

Если Вы еще не сталкивались с Silverlight, то переходите сразу ко второй главе.

Поддержка Drag&Drop

Обзор новых возможностей начнем с простого, но приятного нововведения в Silverlight 4 — поддержки функциональности Drag&Drop. Естественно, речь идет не о том, как перемещать элементы внутри самого Silverlight-приложения (это легко сделать и в первой версии), а как передать приложению внешний объект, находящийся на машине пользователя.

В качестве примера использования Drag&Drop в вебе можно рассмотреть сценарий, когда с помощью Silverlight-приложения пользователь выполняет загрузку файла на сервер: добавляет фотографию в свой профиль; загружает файл в свое хранилище на Skydrive и т. д. Все эти сценарии можно реализовать с помощью дополнительного диалогового окна, но Drag&Drop позволит добавить некоторую «изюминку» вашему интерфейсу.

Чтобы продемонстрировать использование Drag&Drop, разработаем приложение, в окно которого можно перетянуть несколько изображений (в формате, поддерживаемом Silverlight) и отобразить их.

Для реализации Drag&Drop механизма в Silverlight 4 присутствует очень полезное свойство **AllowDrop**, которое доступно у любого из наследников **UIElement**, то есть у любого из визуальных элементов, включая элементы компоновки и графические примитивы. Это означает, что можно реализовать механизм «перетаскивания» внешнего объекта на любой из визуальных элементов.

Устанавливая свойство **AllowDrop** в значение True, разработчик инициирует работу четырех событий, которые можно использовать для реализации Drag&Drop функциональности: **Drop**, **DragEnter**, **DragLeave**, **DragOver**. Основным событием является **Drop**, именно оно генерируется при завершении операции перетаскивания объекта. Остальные три события генерируются при перемещении курсора мыши, захватившего объект, но до завершения операции

перетаскивания, то есть эти события являются вспомогательными и могут быть использованы для дополнительной визуализации процесса перетаскивания в интерфейсе (например, выделение цветом областей, доступных для освобождения объекта).

Реализуем простой интерфейс, содержащий кнопку и панель типа **Canvas**. Объект на основе **Canvas** подходит для наших целей лучше всего, так как позволяет привязать отображаемое изображение к координатам курсора мыши.

```
<UserControl x:Class="DragAndDrop_Chapter0.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.RowDefinitions>
            <RowDefinition Height="50"></RowDefinition>
            <RowDefinition></RowDefinition>
        </Grid.RowDefinitions>
        <Button x:Name="printButton" Content="Print" Width="100">
        </Button>
        <Border Grid.Row="1" x:Name="photoBorder"
            BorderBrush="Blue" BorderThickness="4">
            <Canvas x:Name="photoPanel" Grid.Row="1"
                    Background="Gray" AllowDrop="True"
                    Drop="photoPanel_Drop"
                    DragEnter="photoPanel_DragEnter"
                    DragLeave="photoPanel_DragLeave">
            </Canvas>
        </Border>
    </Grid>
</UserControl>
```

В приведенном выше коде панель photoPanel будет использоваться для отображения изображений, перетянутых с компьютера пользователя. Тут же мы определили свойство **AllowDrop**, установленное в True, и указали обработчики событий. Кнопка printButton нам понадобится в следующем разделе.

Ниже показана реализация описанных выше обработчиков событий **Drop**, **DragEnter**, **DragLeave**.

```
private void photoPanel_Drop(object sender, DragEventArgs e)
{
    photoBorder.BorderBrush = new SolidColorBrush(Colors.Blue);

    IDataObject data = e.Data;
    FileInfo[] files = (FileInfo [])data.GetData
        (DataFormats.FileDrop);

    int i=0;
    foreach (FileInfo f in files)
    {
        Stream s=f.OpenRead();
        BitmapImage bitmap=new BitmapImage();
        bitmap.SetSource(s);

        Image img = new Image();
```

```
            img.Source = bitmap;

            img.Width = 100;
            img.Height = 80;

            Point p = e.GetPosition(photoPanel);
            img.Margin = new Thickness(p.X+10*i, p.Y+8*i, 0, 0);
            i++;

            photoPanel.Children.Add(img);
        }
    }

    private void photoPanel_DragEnter(object sender, DragEventArgs e)
    {
        photoBorder.BorderBrush = new SolidColorBrush(Colors.Red);
    }

    private void photoPanel_DragLeave(object sender, DragEventArgs e)
    {
        photoBorder.BorderBrush = new SolidColorBrush(Colors.Blue);
    }
```

Последние два события мы обрабатываем лишь с целью выделить область, доступную для размещения перетаскиваемого объекта (тут нужно помнить, что при генерации события **Drop**, событие **DragLeave** уже не генерируется, поэтому нужно снять выделение). А вот основную работу делает обработчик события **Drop**: получение информации о передаваемых файлах, открытие

Рис. 1.1. Результат работы приложения

потока, загрузка и отображение изображения с помощью Pixel API (одно из новшеств Silverlight 3).

Чтобы сделать код более читабельным, мы не обрабатываем ошибки, связанные с типом файлов, но в реальной задаче это необходимо сделать обязательно (попробуйте в этом примере перетащить текстовый файл или офисный документ и Вы получите ошибку).

Чтобы продемонстрировать работу нашего приложения, выберите одно или несколько изображений на вашем компьютере и попробуйте перетащить его в окно приложения. Уменьшенное изображение отобразится по текущим координатам (рис. 1.1).

Печать из Silverlight-приложений

Одна из важных и фундаментальных возможностей, которая должна значительно расширить возможности Silverlight-приложений, это печать.

Для демонстрации возможности печати будем использовать предыдущий пример, определив обработчик события **Click** для кнопки printButton, объект типа **PrintDocument** и обработчик события **PrintPage**. Вот как будет выглядеть наш код:

```
private PrintDocument printDoc=new PrintDocument();

public MainPage()
{
    InitializeComponent();

    printDoc.PrintPage += new
       EventHandler<PrintPageEventArgs>(printDoc_PrintPage);
}

void printDoc_PrintPage(object sender, PrintPageEventArgs e)
{
    e.PageVisual = photoPanel;
    e.HasMorePages = false;
}

private void printButton_Click(object sender, RoutedEventArgs e)
{
    printDoc.DocumentName = "Images";
    printDoc.Print();
}
```

Как видно из этого примера, за печать в Silverlight отвечает класс **PrintDocument**. Объект этого класса способен генерировать три события: **StartPrint**, **EndPrint** и **PrintPage**. Первые два события позволяют провести предварительную подготовку к печати и получить информацию об успешном завершении печати соответственно. А вот обработчик события **PrintPage**, как раз и выполняет всю черновую работу. Задача обработчика события **PrintPage** — получить параметры страницы (тут только длина и ширина отображаемой об-

ласти), используя параметр типа **PrintPageEventArgs**, сформировать содержимое страницы и отправить ее на печать.

Фактически, объект типа **PrintDocument** способен печатать любой элемент, порожденный от **UIElement**, то есть, для печати страницы Вы можете выбрать любой из контейнеров, расположить в нем свои элементы и передать ссылку на этот контейнер с помощью свойства **PageVisual**. После этого **UIElement** преобразуется в картинку, и происходит его печать.

В примере выше мы не формировали специальную страницу для печати, а просто использовали существующий контейнер, отображающий наши картинки.

В завершение хочу отметить, что при реализации печати необходимо помнить о важном свойстве **HasMorePages**. Именно благодаря этому свойству у программиста появляется возможность печатать не одну, а несколько страниц. Как только печать закончена, свойство нужно установить в **false**.

Обработка нажатия правой кнопки мыши

Еще с выходом Silverlight 1, многие разработчики начали жаловаться на то, что не могут реализовать собственное контекстное меню при нажатии правой кнопки мыши в своем приложении. Действительно, правая кнопка полностью принадлежала Silverlight, а пользователь мог вызвать только контекстное меню, позволяющее получить доступ к настройкам встраиваемого компонента.

Silverlight 4 позволяет полностью переопределить поведение при нажатии правой кнопки мыши. При этом программист вовсе не обязан отображать меню. Действие может полностью зависеть от самого приложения.

Чтобы переопределить работу правой кнопки мыши, достаточно выполнить два действия:

1. Отключить существующее меню. Для этого нужно определить обработчик с события **MouseRightButtonDown** и установить свойство **Handled** в true. Это сигнализирует о том, что мы берем обработку события на себя и стандартное меню отображать не нужно;

2. Определить вызов собственного меню (или любые другие действия) в обработчике события **MouseRightButtonUp**.

Расширим функционал предыдущего приложения, добавив контекстное меню к изображениям на панели. Для этого расширим метод photoPanel_Drop, добавив следующий код:

```
img.MouseRightButtonDown += new
   MouseButtonEventHandler(img_MouseRightButtonDown);
img.MouseRightButtonUp += new
   MouseButtonEventHandler(img_MouseRightButtonUp);
```

Теперь реализуем сами обработчики событий. Начнем с **MouseRightButtonDown**:

```
void img_MouseRightButtonDown(object sender, MouseButtonEventArgs e)
{
    e.Handled = true;
}
```

Как видно, тут нет ничего сложного. Мы просто нотифицируем Silverlight о перехвате события правой кнопки и подавляем отображение стандартного контекстного меню.

После этого реализуем **MouseRightButtonUp**. Тут есть проблема, которая состоит в том, что если Вы хотите отобразить меню, то его нужно реализовать самостоятельно, или обратиться к сторонним разработчикам за дополнительными компонентами. В списке стандартных компонент и в SDK контекстное меню пока не реализовано. Я не стал разрабатывать меню сам, а установил тестовую версию с сайта: http://www.telerik.com. Тут Вы можете найти множество элементов управления для Silverlight 4.

Замечание. Никогда не используйте контекстное меню от Telerik так, как показано ниже. Компонент достаточно «умный», чтобы отобразить меню в качестве реакции на какое-то событие. Кроме того, этот элемент можно ассоциировать с любым интерфейсным элементом в момент его создания в XAML. Но, если сделать все правильно, то обработчик события для правой кнопки мыши не понадобится ☺. Поэтому будем создавать меню именно при нажатии правой кнопки мыши и уничтожать его в «ручном» режиме.

```
private Image menuImage;
private RadContextMenu contextMenu;

void img_MouseRightButtonUp(object sender, MouseButtonEventArgs e)
{
    menuImage = (Image)sender;
    if (contextMenu != null)
    {
        photoPanel.Children.Remove(contextMenu);
    }

    contextMenu = new RadContextMenu();
    contextMenu.Items.Add("Delete");
    contextMenu.Items.Add("Close Menu");

    Point p=e.GetPosition(photoPanel);

    contextMenu.Margin = new Thickness(p.X, p.Y, 0, 0);
    photoPanel.Children.Add(contextMenu);

    contextMenu.ItemClick += new
  Telerik.Windows.RadRoutedEventHandler(menu_ItemClick);
}

void menu_ItemClick(object sender,
  Telerik.Windows.RadRoutedEventArgs e)
{
    RadRoutedEventArgs args = e as RadRoutedEventArgs;
    RadMenuItem menuItem = args.OriginalSource as RadMenuItem;
    string tag = Convert.ToString(menuItem.Header);
    switch (tag)
    {
        case "Delete":
            photoPanel.Children.Remove(menuImage);
            photoPanel.Children.Remove(contextMenu);
```

```
                break;
            case "Close Menu":
                photoPanel.Children.Remove(contextMenu);
                break;
        }

        contextMenu = null;
    }
```

В коде выше я реализовал механизм создания нового контекстного меню, а также механизм обработки события, связанного с выбором пункта меню. Частично продублировал работу, уже реализованную в компоненте.

Ниже показан снимок экрана во время работы приложения.

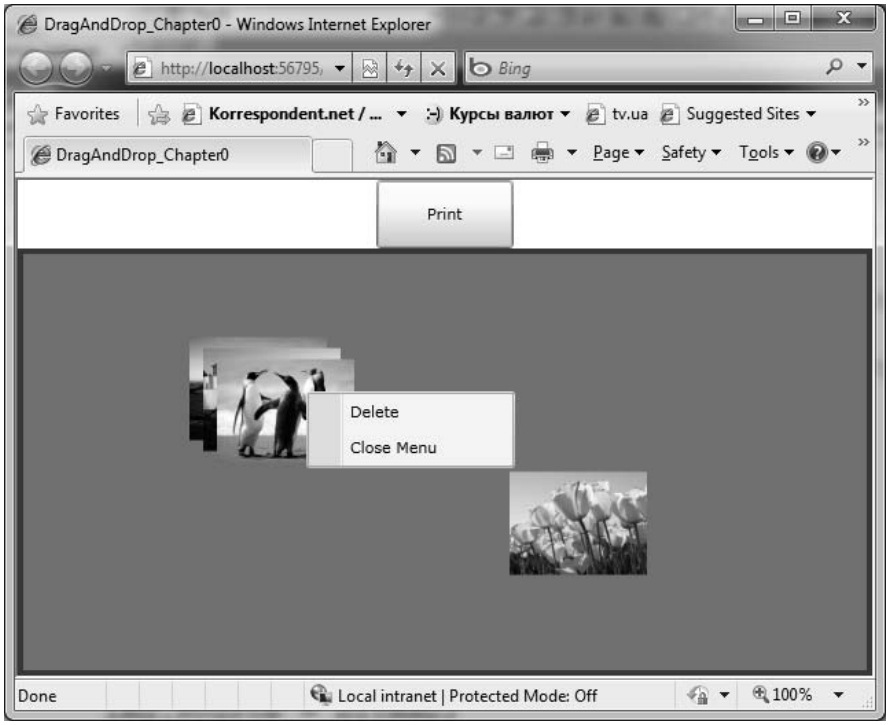

Рис. 1.2. Результат работы приложения

Работа с буфером обмена

Следующая возможность позволяет получить доступ к буферу обмена. Пока речь идет только о работе с текстом, но в будущем (я пока использую бета-версию) эта функциональность будет расширена, и пользователь сможет вставлять и копировать изображения.

Механизм работы с буфером реализован в двух вариантах:

- копирование и вставка текста с помощью горячих клавиш (Ctrl+C, Ctrl+V) при работе с такими элементами как **TextBox**, **RichTextBox** и т. д. Как реализован этот механизм, и как воссоздать его в своем эле-

менте управления — пока остается загадкой. Ведь Ctrl+C, Ctrl+V работают тут без каких-либо разрешений от пользователя, то есть работают всегда, в отличии от второго механизма;

- доступ к буферу обмена с помощью статического класса **Clipboard**, содержащего три статических метода:
- **ContainsText** — проверяет наличие текста в буфере обмена;
- **GetText** — позволяет получить текст из буфера обмена;
- **SetText** — сохраняет текст в буфере обмена.

Рассмотрим второй механизм более детально, так как только он предоставляет возможности для разработчика.

Итак, первое, с чем Вы встретитесь при использовании класса **Clipboard**, — это доступность его методов только в ответ на событие инициированное пользователем, например, нажатие кнопки. При этом пользователь получит сообщение о том, что приложение пытается получить доступ к буферу обмена:

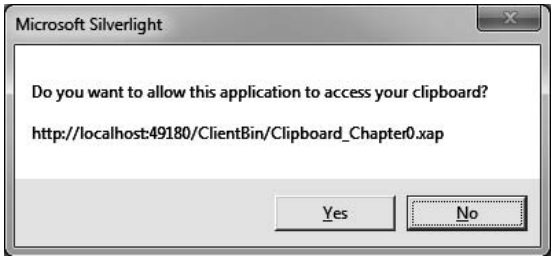

Рис. 1.3. Сообщение пользователю

Пользователь может заблокировать или предоставить доступ приложения к буферу обмена. Если пользователь блокирует доступ к буферу, то генерируется исключение **SecurityException**. Что интересно, если пользователь блокирует доступ при первой попытке, то исключение будет генерироваться всякий раз при попытке вызвать методы класса **Clipboard**, но сообщение пользователь видеть уже не будет. Механизма отобразить пользователю сообщение еще раз я не нашел. Еще одна загадка данной функциональности.

Чтобы продемонстрировать работу с буфером обмена, реализуем простой пример, содержащий поле редактирования и две кнопки:

MainPage.xaml

```
<UserControl x:Class="Clipboard_Chapter0.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
    <StackPanel x:Name="LayoutRoot" Background="White">
        <TextBox Height="92" Name="textBox1" Width="339"
         AcceptsReturn="True"
         HorizontalScrollBarVisibility="Auto" />
        <Button Content="Copy" Height="23"
         Name="copyButton" Width="75"
         Click="copy_Click" />
```

```
        <Button Content="Past" Height="23"
         Name="pasteButton" Width="75" Click="past_Click" />
    </StackPanel>
</UserControl>
```

MainPage.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Security;

namespace Clipboard_Chapter0
{
    public partial class MainPage : UserControl
    {
        public MainPage()
        {
            InitializeComponent();
        }

        private void past_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (Clipboard.ContainsText())
                    textBox1.Text = Clipboard.GetText();
            }
            catch (SecurityException ex)
            {
                textBox1.Text = "You didn't grant permissions!";
            }
        }

        private void copy_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                Clipboard.SetText(textBox1.Text);
            }
            catch (SecurityException ex)
            {
                textBox1.Text = "You didn't grant permissions!";
            }
        }
    }
}
```

Если Вы запустите этот пример, то на экране появится следующее окно, позволяющее работать с текстом, копируя и вставляя содержимое из буфера обмена (рис. 1.4).

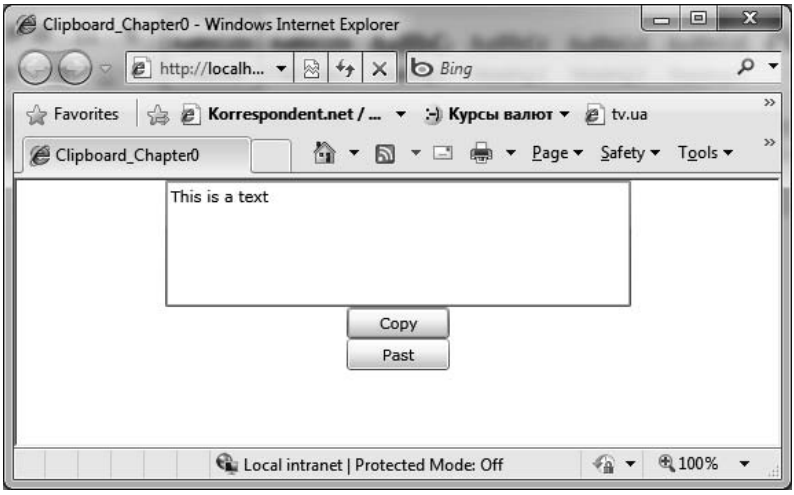

Рис. 1.4. Результат работы приложения

Элементы управления WebBrowser и HtmlBrush

Когда я услышал об очередной возможности Silverlight — отображение HTML, то, конечно же, сильно обрадовался. Ведь это позволило бы разработчикам использовать уже наработанные XML, вместе с XSLT-преобразованием, для отображения отформатированного содержимого (говорят, что слово «контент» — это уже русское слово, поэтому дальше буду использовать его). Однако, все оказалось немного более печально, чем в моих ожиданиях.

Возможность отображения HTML действительно появилась, но работает только для **приложений вне браузера** (Out Of Browser). И это очень странно. Ведь если речь идет об Internet-приложениях, то вряд ли конечный пользователь будет что-то устанавливать себе на машину, не поработав предварительно с приложением в окне браузера. Корпоративная сеть — другое дело, но тут можно использовать и технологию WPF (а то и Win Forms), которая к тому же и быстрее. А учитывая способ инсталляции приложений вне браузера (ведь их нельзя отправить по электронной почте или указать ссылку на инсталляцию), разработчики получают дополнительную головную боль при разработке «двух» версий одного приложения (в браузере и вне браузера). Фактически речь будет идти об одной версии, но с различным поведением, в зависимости от способа запуска.

Из сказанного выше, я делаю вывод, что с помощью этой функциональности Silverlight 4 пытаются позиционировать, как технологию, обладающую рядом преимуществ при создании приложений, работающих в корпоративной сети. Ниже мы рассмотрим еще несколько аналогичных возможностей, которые ставят Silverlight 4 на одну ступень с WPF, но только для приложений, работающих вне браузера.

Как бы то ни было, но если Вы разрабатываете приложение, работающее вне браузера, то можете использовать элемент управления **WebBrowser**, распо-

ложенный в стандартной сборке **System.Windows.dll**. Удивительно, но этот элемент входит в стандартный набор и не требует дополнительных сборок.

При создании элемента **WebBrowser** необходимо явно установить свойства **Height** и **Width**, так как по умолчанию они установлены в 0. Также важным свойством является **Source**, позволяющее установить адрес страницы. Чаще всего это свойство используется для установки начального значения в XAML-файле, но может быть полезно и для получения текущего адреса.

Особое внимание нужно обратить на то, что страница, отображаемая в **WebBrowser**, **должна находиться в домене** Silverlight-приложения (даже после установки на компьютер пользователя ощущается боязнь DOS-атак у разработчиков этой функциональности). Для отображения HTML-контента в процессе работы приложения, можно использовать один из двух методов:

- **Navigate** — действие этого метода аналогично установке свойства **Source**, то есть элемент **WebBrowser** отобразит страницу, если она находится в домене приложения. Если Вы хотите отобразить внешний контент, то необходимо разместить ссылку на него внутри элемента **iframe**;
- **NavigateToString** — этот метод более универсален и позволяет отобразить любой HTML-контент, полученный в качестве параметра. Это означает, что если Вы хотите отобразить внешний контент, то просто формируете строку, содержащую **iframe**, и указываете ссылку на необходимую страницу (независимо от домена).

Ниже показан пример приложения, которое позволяет осуществлять навигацию на любую страницу из Silverlight-приложения:

```
<UserControl x:Class="HTML_Chapter0.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
>
    <StackPanel x:Name="LayoutRoot" Background="White">
        <StackPanel Orientation="Horizontal"
            HorizontalAlignment="Center">
            <TextBox Name="urlText"
              Text="http://www.microsoft.com" Width="300">
            </TextBox>
            <Button Content="Navigate"
Click="Button_Click"></Button>
        </StackPanel>
        <WebBrowser Height="800" Width="1024" Name="webBrowser">
        </WebBrowser>
    </StackPanel>
</UserControl>
```

А вот и код, осуществляющий навигацию:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    webBrowser.NavigateToString(
        String.Format(
        "<iframe height=800 width=1024 src={0}></iframe>",
        urlText.Text));
}
```

При этом не забудьте сконфигурировать приложение как работающее вне браузера (и установить его после запуска на локальную машину).

Результат работы приложения Вы можете увидеть ниже:

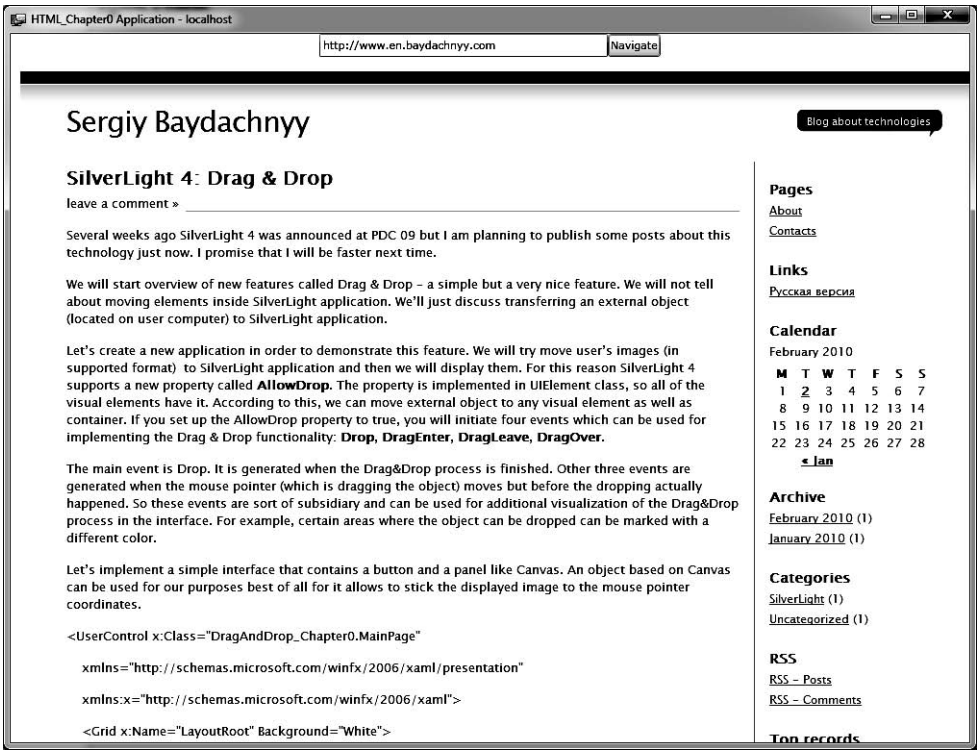

Рис. 1.5. Результат работы приложения

Использование **iframe** позволяет не только отобразить HTML-страницу, которая находится вне домена приложения, но и внедрить элемент **<object>**. Это автоматически означает, что Silverlight-приложения с легкостью могут содержать в себе Flash-приложения (например, отображать ролики с youtube).

Нужно также отметить и возможность принудительно выполнять Java-скрипты, содержащиеся в **WebBrowser** элементе. Для этого используется метод **InvokeScript**.

В завершении раздела отмечу, что HTML можно использовать в качестве заливки для любого элемента, поддерживающего работу с **Brush**. Для этих целей существует специальный класс **HtmlBrush**, который в качестве источника принимает объект **WebBrowser**.

RichTextArea элемент управления

Описывая очередную возможность Silverlight 4, почему-то вспомнил лозунг: «Перед использованием — доработать напильником». Действительно, если смотреть на предыдущие разделы, то вроде бы новый функционал и есть,

но чего-то все время не хватает. Есть возможность определить контекстное меню, но инфраструктуры для такого меню нет (хотя инфраструктура есть даже для подсказок), есть возможность работать с буфером обмена, но с жесткими ограничениями, есть возможность отображать HTML, но только для приложений, работающих вне браузера. Не исключением является и новый элемент управления, который сразу попал в стандартную поставку Silverlight 4, — это **RichTextArea**.

Уже пару лет разработчики пытаются добиться элемента управления, который бы позволил отображать форматированный текст. Пусть это будет DOCX, RTF, XPS или PDF, что не столь принципиально. Большинство текстовых редакторов спокойно преобразуют текст из одного формата в другой. Вместо ожидаемого элемента, разработчикам предоставили **RichTextArea**, который действительно способен отображать текст с минимальным форматированием, но в своем, только ему понятном, формате. Давайте посмотрим в каком именно.

Ниже пример элемента RichTextArea, отображающего текст с минимальный форматированием:

```
<RichTextArea HorizontalAlignment="Left"
    Name="rArea" VerticalAlignment="Top"
    Height="300" Width="400" >
    <Paragraph>
        <Bold>This is a bold text</Bold>
        <LineBreak></LineBreak>
        <Underline>This is an underline text</Underline>
        <LineBreak></LineBreak>
        <Italic>This is an italic text</Italic>
        <LineBreak></LineBreak>
        This is a button:
        <InlineUIContainer>
            <Button Content="Button"></Button>
        </InlineUIContainer>
    </Paragraph>
</RichTextArea>
```

В результате на экране отобразится следующий контент (рис. 1.6).

Рис. 1.6

Как видно, **RichTextArea** является контейнером для других элементов. Так, основным наполнением элемента **RichTextArea** выступает набор элементов **Paragraph**. Эти элементы описывают параграфы текстового документа. В свою очередь, параграф может включать набор из следующих элементов:

- **Run** — задает обычный текст;
- **Span** — служит для группировки других элементов;
- **Bold** — определяет жирное начертание символов;

- **LineBreak** — переход на другую строку;
- **Italic** — определяет рукописное начертание символов;
- **Underline** — текст с подчеркиванием;
- **HyperLink** — создает гиперссылку, которая становится активной только в режиме ReadOnly элемента **RichTextArea**;
- **InlineUIContainer** — позволяет вставить в документ любой из элементов, порожденных от **UIElement**.

Естественно, создавать и заполнять **RichTextArea** можно и программно. Вот небольшой пример кода:

```
Bold b = new Bold();
b.Inlines.Add("This is a bold text");

Italic i = new Italic();
i.Inlines.Add("This is an italic text");

Underline u = new Underline();
u.Inlines.Add("This is an underlined text");

Paragraph myPar = new Paragraph();
myPar.Inlines.Add(b);
myPar.Inlines.Add(new LineBreak());
myPar.Inlines.Add(i);
myPar.Inlines.Add(new LineBreak());
myPar.Inlines.Add(u);

rArea.Blocks.Add(myPar);
```

Таким образом, **RichTextArea** является элементом с минимальным интерфейсом, где всю работу необходимо выполнить программисту. Конечно, Вы можете найти примеры готовых методов, позволяющих сохранить и загрузить содержимое **RichTextArea**, или реализацию удобного интерфейса пользователя (вот один из примеров: http://channel9.msdn.com/learn/courses/Silverlight4/RichTextEditor), но много нужно будет сделать самостоятельно.

Управление окном приложения

Еще одна возможность приложений, работающих вне браузера, — это поддержка объекта типа **Window**. С помощью класса **Window** разработчик способен управлять окном во время работы приложения.

Естественно, что объект типа **Window** создается «за сценой», а разработчик может получить доступ к нему с помощью свойства **MainWindow** объекта **Application**.

Среди основных свойств класса **Window** можно выделить следующие:

- **Height** — определяет высоту окна;
- **Width** — определяет длину окна;
- **Left** — задает отступ от левой границы экрана;
- **Top** — задает отступ от верхней границы экрана;
- **IsActive** — возвращает **true**, если окно активно в данный момент и находится в фокусе. В противном случае возвращает **false**;

- **TopMost** — если это свойство установлено в **true**, то окно всегда располагается поверх других окон (всегда находится на экране);
- **WindowState** — определяет состояние окна (Normal, Minimize, Maximize).

Среди методов можно выделить лишь **Activate**, который активирует окно приложения, выводя его на передний план и передавая ему фокус.

Стоит также отметить, что устанавливать свойства объекта, порожденного от **Window,** можно лишь в ответ на действия пользователя (нажатие кнопки и др.) либо в обработчике события **Startup** объекта **Application** (либо до Startup).

Silverlight 4 поддерживает специальный раздел в конфигурационном файле, позволяющий установить начальные параметры окна. Вот как может выглядеть часть конфигурационного файла с установленными параметрами:

```
<OutOfBrowserSettings>
  <OutOfBrowserSettings.WindowSettings>
    <WindowSettings Title="My Window"
      Left="100" Top="100"
      Height="300" Width="300" />
  </OutOfBrowserSettings.WindowSettings>
</OutOfBrowserSettings>
```

Поддержка уведомлений

Следующая возможность приложений, работающих вне браузера, — это способность отображать уведомления. В данном случае, под уведомлениями понимаются всплывающие окна, не требующие взаимодействия с пользователем (хотя они способны обрабатывать события от мыши, но не от клавиатуры) и исчезающие через заданный промежуток времени. Примеры таких окон можно найти в приложении Microsoft Outlook, которое отображает всплывающие сообщения при получении нового письма.

Чтобы отобразить уведомление, достаточно создать объект типа **NotificationWindow** и установить свойство **Content** у созданного объекта. Свойство **Content** принимает любой объект, порожденный от **FrameworkElement**, то есть любой визуальный элемент или элемент компоновки. После установки свойств **Content**, **Width**, **Height**, достаточно вызвать метод **Show**, принимающий время показа окна в миллисекундах.

Рассмотрим пример приложения, работающего вне браузера и отображающего примитивное окно в момент запуска. Интерфейс приложения реализуем следующим образом:

```
<UserControl x:Class="Notify_Chapter0.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Grid x:Name="LayoutRoot" Background="White">
        <StackPanel HorizontalAlignment="Center">
            <TextBlock Text="Notification Demo"
               TextAlignment="Center" FontSize="18">
```

```
            </TextBlock>
            <Button Name="btn" Visibility="Collapsed"
               Content="Инсталировать приложение" FontSize="18"
               Click="Button_Click"></Button>
            <TextBlock Name="txt" Visibility="Collapsed"
               TextWrapping="Wrap" TextAlignment="Center"
               Text="Приложение должно быть запущено вне браузера"
               FontSize="18">
            </TextBlock>
        </StackPanel>
    </Grid>
</UserControl>
```

Во время запуска приложения необходимо проверить, действительно ли приложение запущено вне браузера, и только затем приступать к созданию окна. Вот как это можно реализовать:

```
public MainPage()
{
    InitializeComponent();

    if (App.Current.IsRunningOutOfBrowser)
    {
        NotificationWindow notify = new NotificationWindow();

        StackPanel panel = new StackPanel();
        panel.Background = new SolidColorBrush(Colors.Gray);
        panel.Width = 250;
        panel.Height = 50;

        TextBlock header = new TextBlock();
        header.Text = "New message";
        header.FontWeight = FontWeights.Bold;

        TextBlock message = new TextBlock();
        message.Text = "This is a new message";

        panel.Children.Add(header);
        panel.Children.Add(message);

        notify.Content = panel;
        notify.Width = panel.Width;
        notify.Height = panel.Height;
        notify.Show(10000);
    }
    else
    {
        if (App.Current.InstallState == InstallState.Installed)
        {
            txt.Visibility = Visibility.Visible;
        }
        else
        {
            btn.Visibility = Visibility.Visible;
```

```
        }
    }
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    App.Current.Install();
}
```

В результате после запуска приложения на экране можно увидеть следующую картину:

Рис. 1.7. Результат работы приложения

Как видно, уведомление отображается в системной области с небольшим отступом от нее (по 44 единицы).

Данная функциональность может быть полезна при обработке обновлений, коммуникациях, асинхронной обработке данных и др.

Поддержка микрофона и камеры

Еще одна потрясающая возможность Silverlight 4, — это поддержка камеры и микрофона. Чтобы показать эти возможности в действии, рассмотрим небольшой пример. Для этого создадим новый Silverlight-проект и реализуем следующий интерфейс:

```
<UserControl x:Class="WebCam_Chapter0.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <StackPanel x:Name="LayoutRoot" Background="White"
        Loaded="LayoutRoot_Loaded">
        <Rectangle Width="400" Height="300" Fill="Gray"
            Name="videoContainer">
        </Rectangle>
```

Оглавление

От автора 3

Глава 1. ВВЕДЕНИЕ В SILVERLIGHT 4 5
- Поддержка Drag&Drop 5
- Печать из Silverlight-приложений 8
- Обработка нажатия правой кнопки мыши 9
- Работа с буфером обмена 11
- Элементы управления WebBrowser и HtmlBrush 14
- RichTextArea элемент управления 16
- Управление окном приложения 18
- Поддержка уведомлений 19
- Поддержка микрофона и камеры 21
- Поддержка колесика мыши 27
- Элемент управления ViewBox 28
- Повышение доверия 29
  - Расширенные возможности работы в полноэкранном режиме 30
  - Отсутствие сообщений о доступе к ресурсам 30
  - Запросы между доменами 31
  - Доступ к некоторым папкам 31
  - Взаимодействие с COM 31
- Неявные стили 32
- Заключение 32

Глава 2. НАЧИНАЕМ РАБОТУ С SILVERLIGHT 33
- Что такое Silverlight? 33
- Инструменты для создания Silverlight-приложений. 35
- Первое приложение в Expression Blend 4 37
- Создание приложения в Visual Studio 2010 42

Обзор технологии ... 45
XAML ... 45
Элементы компоновки ... 45
Элементы управления ... 46
Графические примитивы ... 46
Управление видео ... 46
Работа с данными ... 46
Работа со службами ... 47
Работа вне браузера ... 47
Базовые классы ... 47
Заключение ... 47

Глава 3. АРХИТЕКТУРА SILVERLIGHT ... 48
Структура приложения ... 48
Развертывание приложения ... 53
Кэширование сборок и загрузка по требованию ... 54
Загрузка сборки по требованию ... 54
Кэширование сборки ... 58
Размещение Silverlight-элемента на странице ... 60
Использование элемента <object> ... 60
Немного о классах в JavaScript ... 62
Использование Silverlight.js ... 64
Анимация во время загрузки ... 66
Взаимодействие со встраиваемым элементом ... 69
Использование JavaScript ... 69
Переход в полноэкранный режим ... 70
Взаимодействие Silverlight и JavaScript ... 72
Вызов управляемых методов из JavaScript ... 72
Вызов JavaScript методов из управляемого кода ... 74
Взаимодействие между Silverlight-приложениями ... 75
Заключение ... 78

Глава 4. ИСПОЛЬЗОВАНИЕ XAML ... 79
Введение в XAML ... 79
Основные конструкции ... 80
Пространства имен в XAML ... 84
Подключение кода и обработчиков событий ... 85
Расширение разметки ... 88

Зависимые свойства ... 89
Динамическая загрузка XAML ... 90
Заключение ... 91

Глава 5. ЭЛЕМЕНТЫ УПРАВЛЕНИЯ И СОБЫТИЯ ... 92
Немного об элементах управления ... 92
Элементы компоновки ... 94
Элемент управления Canvas ... 95
Элемент управления StackPanel ... 96
Элемент управления Grid ... 99
Базовые элементы управления ... 105
Класс Control ... 105
Кнопки ... 106
Текстовые элементы управления ... 108
Элементы управления списками ... 109
Элементы управления, основанные на диапазоне значений ... 111
Элемент управления ToolTip ... 112
Использование диалоговых окон ... 112
Заключение ... 112

Глава 6. ПРИВЯЗКА К ДАННЫМ ... 113
Привязка к свойству элемента управления ... 113
Привязка к объекту ... 117
Привязка к коллекции ... 122
Конвертеры данных ... 125
Проверка данных при связывании ... 128
ValidatesOnExceptions и NotifyOnValidationError ... 128
ValidatesOnDataErrors ... 130
ValidatesOnNotifyDataErrors ... 133
Заключение ... 133

Глава 7. ВЗАИМОДЕЙСТВИЕ С СЕРВЕРОМ ... 135
Использование WebClient ... 135
Использование HttpWebRequest и HttpWebResponse ... 138
Использование прокси-классов для взаимодействия со службами ... 140
Доступ к службам в других доменах ... 143
Заключение ... 144

Глава 8. ГРАФИКА, ТРАНСФОРМАЦИЯ И АНИМАЦИЯ 145
Графические примитивы 145
Кисти 149
SolidColorBrush 149
Поддержка системных цветов 150
LinearGradientBrush 150
RadialGradientBrush 151
ImageBrush и VideoBrush 153
Использование геометрических объектов 154
Работа с изображениями 155
Работа с эффектами 155
Pixel API 159
Работа с кэшем 161
Трансформация 161
Основные виды трансформаций 161
CompositeTransform в Silverlight 4 164
Трехмерные проекции 165
Введение в анимацию 167
Общие типы анимации 167
Запуск анимации 168
Анимация с помощью ключевых кадров 169
Простая анимация 170
Заключение 171

Глава 9. РАБОТА С АУДИО И ВИДЕО 172
Использование MediaElement 172
Общие сведения 172
Использование маркеров 181
Поддержка GPU 183
Возможности Internet Information Services 7 184
Запуск Web Platform Installer 184
Создание списков 187
Возможности Bit Rate Throttling 187
Использование Smooth Streaming 188
Защита видео с помощью DRM 191
Заключение 191

Глава 10. РЕСУРСЫ И СТИЛИ 193
Ресурсы 193
Ресурсы приложения 193
Ресурсы объектов 194
Выделение ресурсов объектов в отдельные файлы 196
Стили 197
Понятие стилей 197
Динамическая установка стилей 200
BasedOn стили 201
Заключение 201

Глава 11. СОЗДАНИЕ ШАБЛОНОВ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ 202
Понятие шаблона 202
Разбор шаблона для элемента Button 203
Составляющие элемента управления 203
Состояния и переходы 205
Заключение 208

Глава 12. ОТЛАДКА ПРИЛОЖЕНИЙ И ТЕСТИРОВАНИЕ 209
Отладка с помощью Visual Studio 2010 209
Обработка ошибок в Silverlight 211
Обработка ошибок в управляемом коде 211
Обработка ошибок в JavaScript 212
Асинхронный вызов методов 213
Тестирование Silverlight-приложений 214
Заключение 218

Глава 13. СОЗДАНИЕ СЛОЖНЫХ ПРИЛОЖЕНИЙ 219
Разработка приложений, работающих вне браузера 219
Isolated Storage 223
IsolatedStorageSettings 223
IsolatedStorageFile 224
Навигация в Silverlight-приложениях 226
Расширение модели приложения 234
Managed Extensibility Framework 235
Заключение 236

Глава 14. ИСПОЛЬЗОВАНИЕ DEEP ZOOM 237
Что такое Deep Zoom? 237
Использование Deep Zoom Composer 238
Работа с Deep Zoom в Silverlight 242
Заключение 243

Глава 15. ИНТЕГРАЦИЯ С SHAREPOINT 2010 244
Обзор возможностей 244
Работа с Web-частями 246
Развертывание Silverlight-приложения с помощью Visual Studio 2010 247
Использование REST 252
Поддержка Client API 256
Заключение 258

Глава 16. ВВЕДЕНИЕ В MICROSOFT EXPRESSION STUDIO 259
Обзор продуктов 259
Работа с Expression Encoder 260
Преобразование видео 260
Использование встроенных шаблонов 262
Использование Expression Encoder для трансляции живого видео . 263
Захват изображения и звука 264
Работаем с Expression Blend 266
Общий обзор 266
Работа с анимацией 269
Создание шаблонов для элементов управления 269
Заключение 272