Начинаем разработку на Silverlight: Часть 2: Определение пользовательского интерфейса и навигации

Это вторая часть цикла статей **Начинаем разработку на Silverlight**. Список статей доступен по ссылке. Полный код проекта на C# можно скачать по ссылке.

Управлением разметкой (layout management) в приложении XAML – важная часть успешной разработки на Silverlight. Большинство возможностей пришло из мира веб и, если вы только не волшебник CSS, работа с ними может вызывать определенные трудности.

Варианты разметки

Silverlight предоставляет гибкую систему для размещения элементов интерфейса. Есть модели разметки, которые поддерживают и динамические и абсолютные стили разметки. Есть достаточно много элементов, предоставляющих управление разметкой, но наиболее используемые, это:

- Canvas
- StackPanel
- Grid

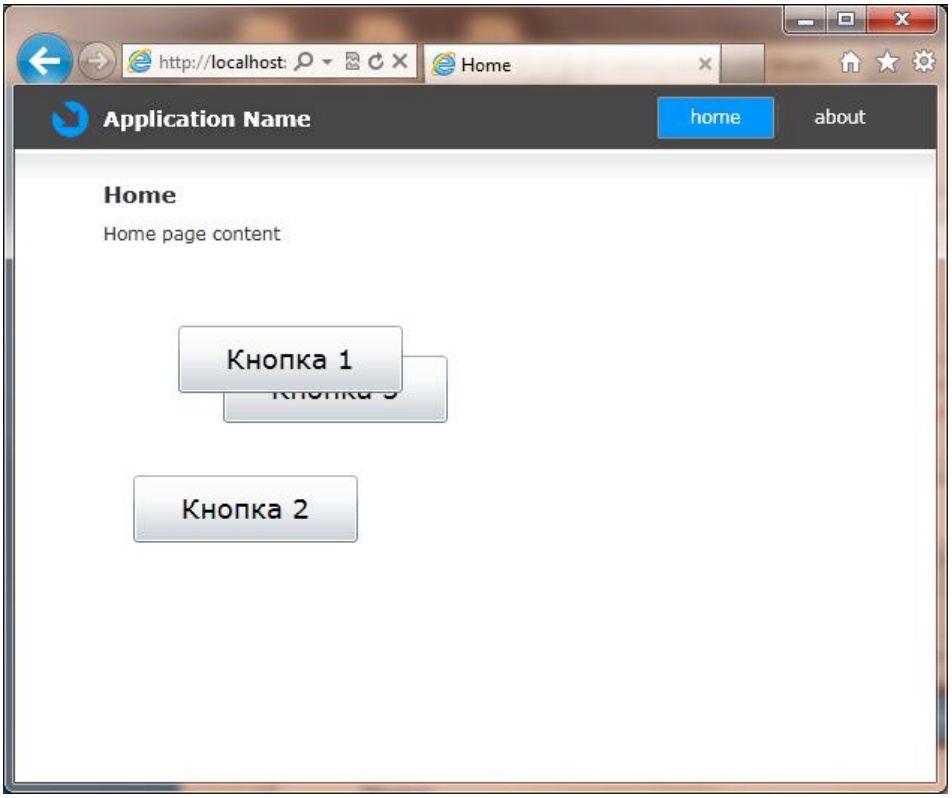
Давайте рассмотрим, как каждый из элементов работает, разметив внутри них другие элементы управления. В целях демонстрации, мы будем использовать простые элементы управления Button. Будет использоваться проект, который мы создали в Части 1, а код, для простоты, будем вставлять в страницу Home.xaml (в процессе исследований часть кода будет удалена, так что сохраните оригинальный проект где-нибудь в стороне).

Canvas

Элемент **Canvas** – предоставляет наиболее простой вариант разметки. Он может быть использован для абсолютного позиционирования элементов с использованием координат. Мы позиционируем элементы на Canvas, используя прикрепленные свойства (*Attached Properties*). Прикрепленные свойства позволяют родительскому элементу расширять свойства размещенного на нем элемента управления (в нашем случае Button). Мы можем разместить несколько кнопок Button на Canvas, например, таким образом:

```
<Canvas>
     <Button Canvas.Top="50" Canvas.Left="50" Content="Кнопка 1" FontSize="18" Width="150" Height="45" />
     <Button Canvas.Top="150" Canvas.Left="20" Content="Кнопка 2" FontSize="18" Width="150" Height="45" />
     <Button Canvas.Top="70" Canvas.Left="80" Canvas.ZIndex="-
1" Content="Кнопка 3" FontSize="18" Width="150" Height="45" />
</Canvas>
```

При запуске приложения (F5) увидим:



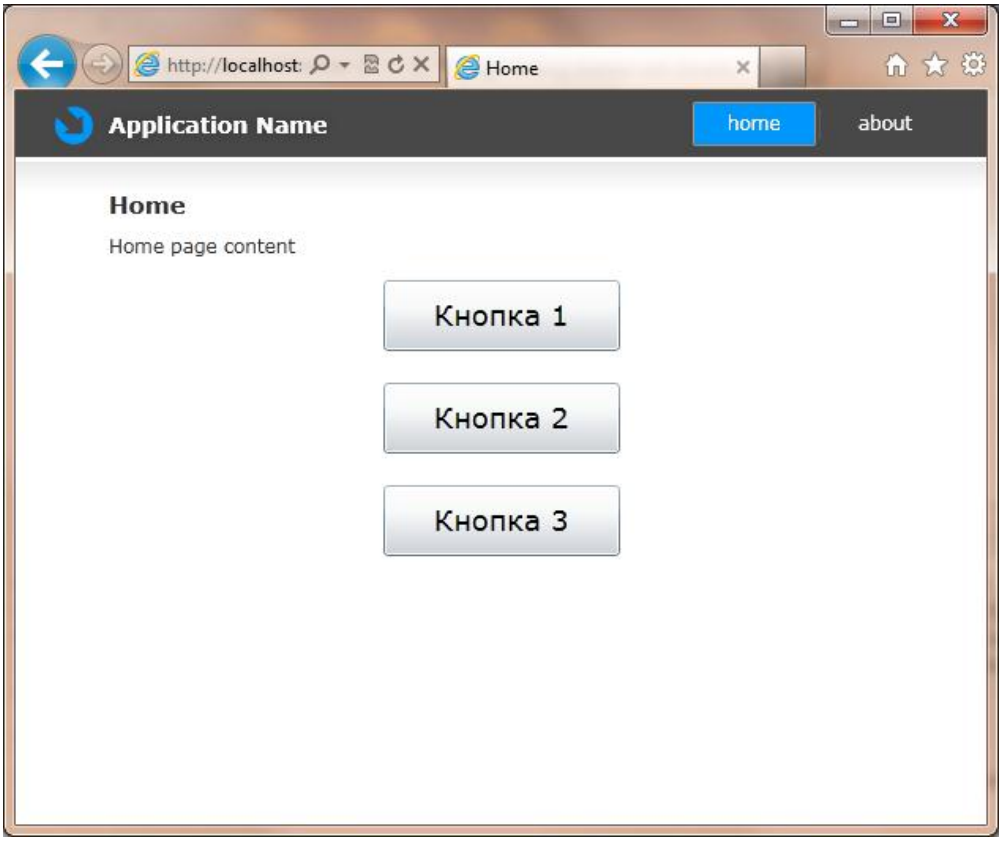
Как видно, элементы позиционированы абсолютным образом. Обратите внимание, что для одной из кнопок указано прикрепленное свойство ZIndex, которое указывает как один элемент перекрывает другой элемент. Элемент Canvas полезен, когда элементы внутри не должны особо перемещаться, а окно приложения изменяться в размере. В противном случае, работать с Canvas может быть сложнее, чем с такими элементами, как StackPanel или Grid.

StackPanel

Элемент **StackPanel** – предоставляет вариант разметки, элементы, помещенный в который, располагаются в стек горизонтально или вертикально (по умолчанию – вертикально). Сделаем пример из трех кнопок Button:

```
            <TextBlock x:Name="ContentText" Style="{StaticResource ContentTextStyle}"
                       Text="Home page content"/>

            <StackPanel>
                <Button Margin="10" Content="Кнопка 1" FontSize="18" Width="150" Height="45" />
                <Button Margin="10" Content="Кнопка 2" FontSize="18" Width="150" Height="45" />
                <Button Margin="10" Content="Кнопка 3" FontSize="18" Width="150" Height="45" />
            </StackPanel>
```

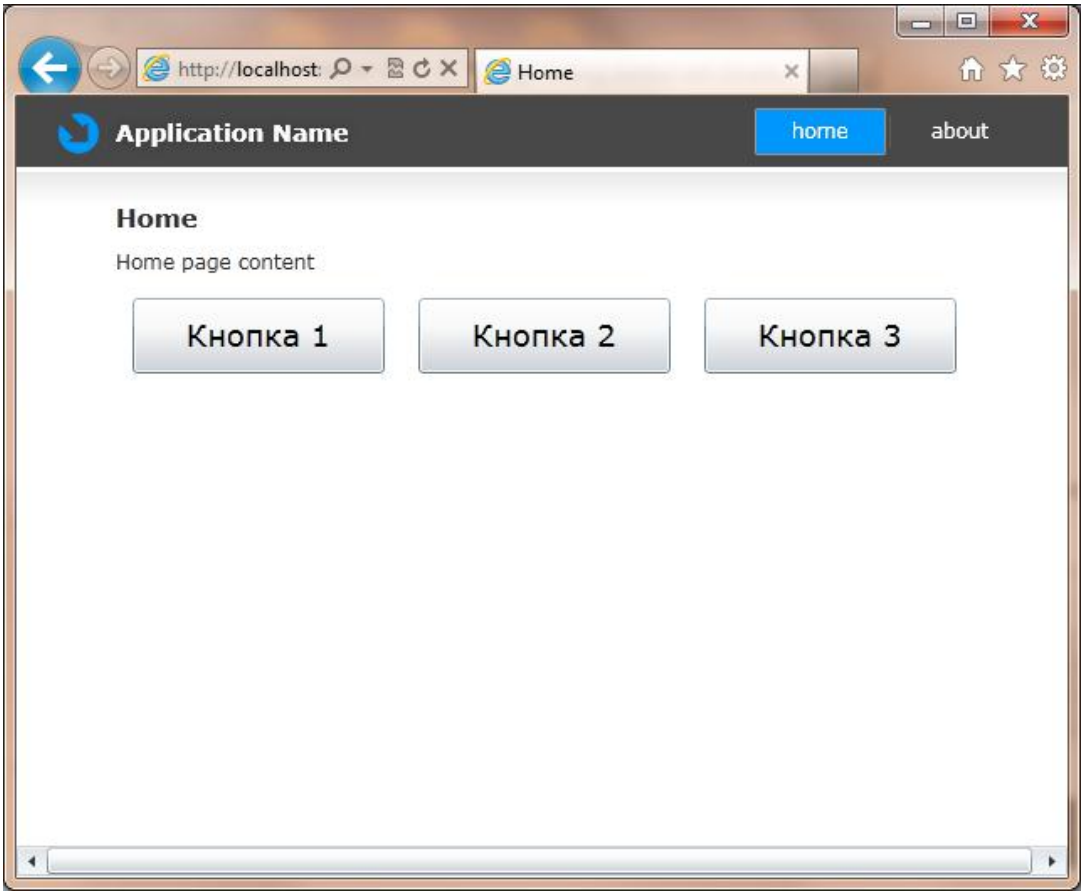
При запуске приложения (F5) увидим:



Или, если изменить ориентацию на горизонтальную (отличие в коде – только атрибут Orientation элемента StackPanel):

```
<StackPanel Orientation="Horizontal">
    <Button Margin="10" Content="Кнопка 1" FontSize="18" Width="150" Height="45" />
    <Button Margin="10" Content="Кнопка 2" FontSize="18" Width="150" Height="45" />
    <Button Margin="10" Content="Кнопка 3" FontSize="18" Width="150" Height="45" />
</StackPanel>
```

При запуске (F5) увидим:



Элемента StackPanel предоставляет простой способ разместить элементы один за другим вертикально или горизонтально, без указания позиционирования элемента внутри контейнера.

Grid

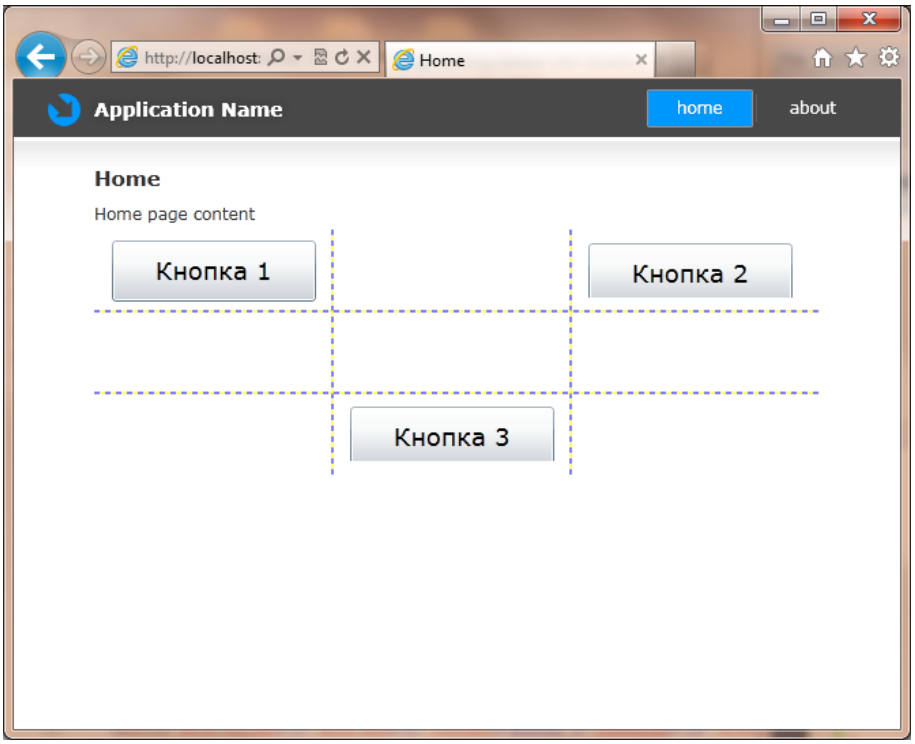
Элемент **Grid** – позволят позиционировать элементы внутри себя максимально гибко, и подходит для большинства сценариев (для большинства, но не для всех). Элемент Grid предоставляет возможность размещать элементы, используя строки и столбцы. Использование XAML элемента Grid отличается от использования элемента <table> при веб-разработке, где контент располагается внутри тегов <tr> и <td>. Разработчик определяет общую структуру сетки Grid, а потом используется присоединенные свойства, чтобы указать, где размещаются элементы.

Рассмотрим следующий код (обратите внимание, для облегчения понимания, включено отображение сетки, чего в рабочем коде, обычно, не предполагается):

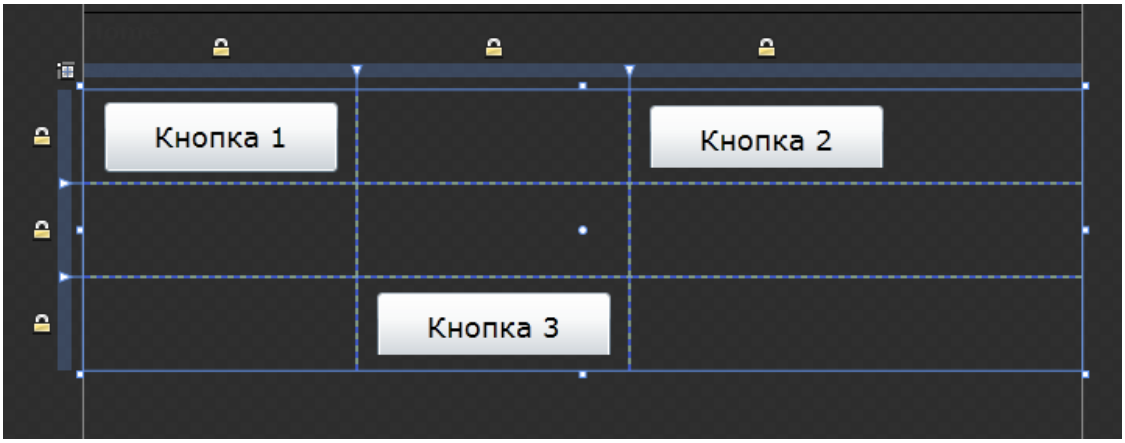
```
<Grid ShowGridLines="True">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="60"/>
                        <RowDefinition Height="60"/>
                        <RowDefinition Height="60"/>
                    </Grid.RowDefinitions>

                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="175" />
                        <ColumnDefinition Width="175" />
                        <ColumnDefinition Width="175" />
                    </Grid.ColumnDefinitions>

<Button Grid.Column="0" Grid.Row="0" Content="Кнопка 1" FontSize="18" Width="150" Height="45" />
```

```
    <Button Grid.Column="2" Grid.Row="0" Margin="10" Content="Кнопка 2" FontSize="18" Width="150" Height="45" />
    <Button Grid.Column="1" Grid.Row="2" Margin="10" Content="Кнопка 3" FontSize="18" Width="150" Height="45" />

</Grid>
```

Мы определили элемент Grid с 3 строками и 3 столбцами с определенной шириной и высотой. Далее, элементы кнопки Button позиционируются внутри элемента Grid с использованием присоединённых свойств. Результат запуска программы будет выглядеть следующим образом:



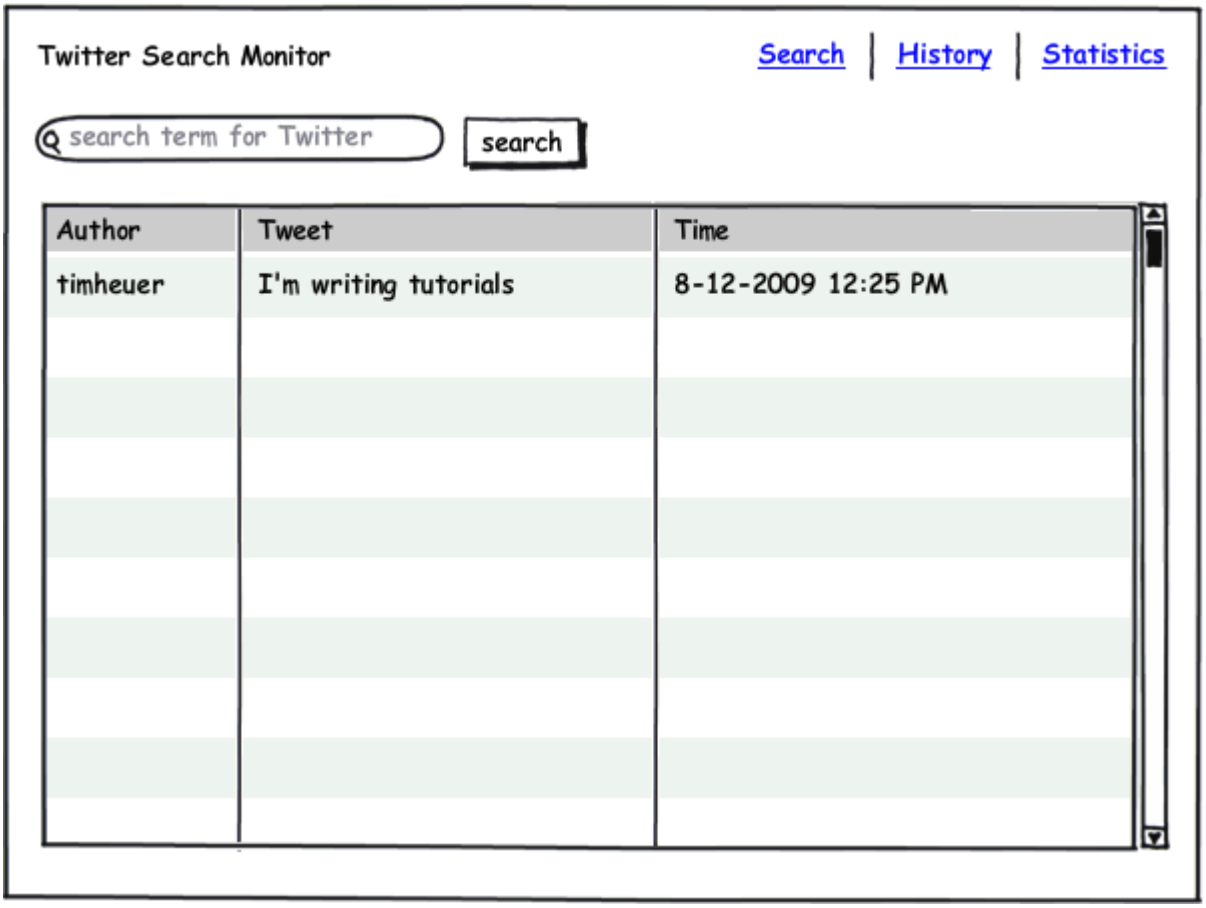
Обратите внимание, как присоединенные свойства кнопки Button (Grid.Column и Grid.Row) указывают, где кнопка располагается в контейнере. В работе с дизайном интерфейса большую помощь может оказать Expression Blend. Он позволяет отредактировать визуально определения столбцов и строк и потом сгенерировать для нас XAML.



В нашем приложении мы будем использовать комбинацию элементов разметки (layout controls). Фактически, тот шаблон, который мы выбрали как стартовую точку, уже использует все рассмотренные типы элементов разметки.

Разработка нашего Twitter приложения

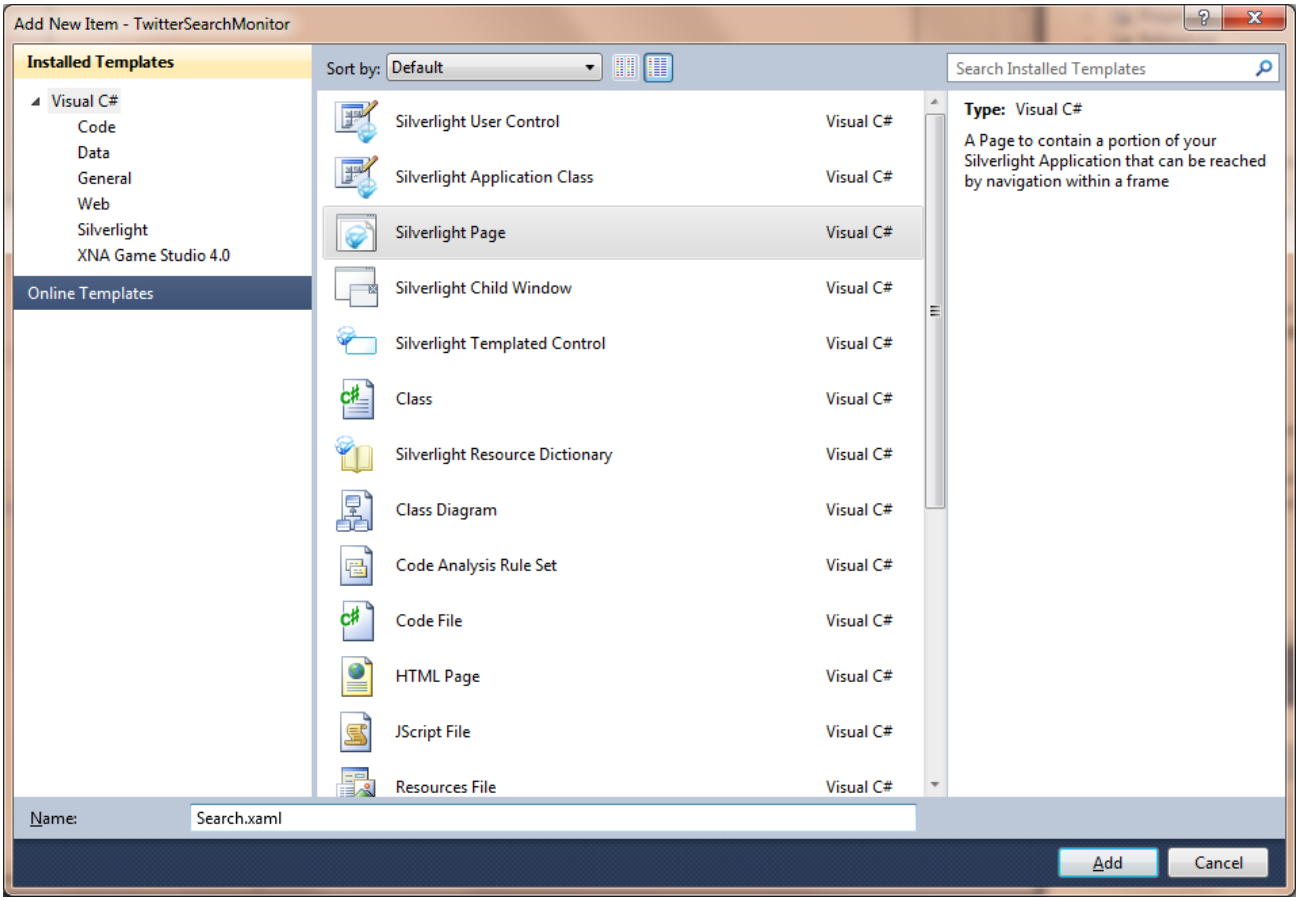
Приступим к разработке нашего приложения. Ниже представлен макет приложения, которое мы собираемся сделать:



У нас есть место, где пользователь может ввести условия поиска, а результат будет отображаться в виде некоторого списка. Также есть элементы навигации, которые позволяют перемещаться между разыми представлениями, такими как история поиска и, возможно, какая-то статистика.

К счастью шаблон навигационного приложения, которое мы выбрали, уже предоставляет нам хорошую базу для создания приложения. Сделаем несколько изменений в файле MainPage.xaml. Приблизительно в 29 строке MainPage.xaml удалим логотип и изменим название приложения в ApplicatonTextBlock ниже на «Twitter Search Monitor».

В следующем подразделе мы вернемся к навигации, а сейчас давайте создадим новое представление. В Visual Studio, в иерархии проекта, надо правым щелчком мыши открыть контекстное меню и выбрать Add, затем New Item, в открывшемся диалоговом окне выбрать Silverlight Page и назвать его Search.xaml



Теперь у нас есть пустая XAML страница, по умолчанию с разметкой Grid. На этой странице мы создадим интерфейс поиска, представленный выше. Заголовок же страницы у нас будет из MainPage.xaml, потому что мы используем элемента Frame для нашей навигации между представлениями.

Навигационная подсистема Silverlight

Сейчас давайте разберёмся с навигационной подсистемой Silverlight. Как вы помните мы начали с шаблона приложения с навигацией. Шаблон по умолчанию создает нам страницу MainPage.xaml и несколько представлений (Home, About). Навигационная подсистема фундаментально состоит из 3 частей: UriMapper, Frame и Page.

UriMapper

Мне удобно представлять, что UriMapper – это некий вариант движка маршрутизации. В нем нет никакой специальной необходимости ни в каком смысле, кроме обфускации и упрощения навигации. Вместо того, чтобы показывать наружу настоящий URI /Views/Home.xaml, мы отображаем его на простой URI /Home, что гораздо удобочитаемо и не раскрывает внутреннюю конфигурацию, а кроме того, может быть изменено в дальнейшем на отображение на что-либо другое. Можно увидеть элемент UriMapper, который является частью элемента Frame на странице MainPage.xaml.

```
<Border x:Name="ContentBorder" Style="{StaticResource ContentBorderStyle}">

            <navigation:Frame x:Name="ContentFrame" Style="{StaticResource ContentFrameStyle}"
                              Source="/Home" Navigated="ContentFrame_Navigated" NavigationFailed="ContentFrame
_NavigationFailed">
```

```
            <navigation:Frame.UriMapper>
              <uriMapper:UriMapper>
                <uriMapper:UriMapping Uri="" MappedUri="/Views/Home.xaml"/>
                <uriMapper:UriMapping Uri="/{pageName}" MappedUri="/Views/{pageName}.xaml"/>
              </uriMapper:UriMapper>
            </navigation:Frame.UriMapper>
        </navigation:Frame>
    </Border>
```

UriMapper может быть частью определения ресурсов и тогда на него можно ссылаться.

Frame

Если вы разработчик ASP.NET, воспринимайте элемент Frame, как элемент управления ContentPlaceholder. Frame – это область в которой может происходить навигация. Мы можем указать представление по умолчанию, но потом может произойти любая навигация – и мы это увидим позже. Если посмотреть на код выше, можно увидеть, что представление по умолчанию – это атрибут Source у Frame – путь /Home.

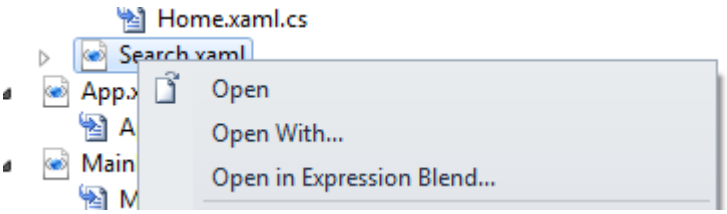
Page

Последняя часть навигационной подсистемы – это элемент Page страница, которую мы создали на последнем шаге. Упрощенно – это область с содержанием, которая отображается в элементе Frame. Элемент Page очень похож на базовый элемент UserControl, который обычно добавляется (что собственно представляет из себя MainPage), отличается он тем, что может взаимодействовать с навигацией. Мы считаем, что элементы Page соответствуют представлениям в нашем приложении.

Создание пользовательского интерфейса для представления поиска

Давайте закончим создание пользовательского интерфейса нашей только что созданной страницы Search.xaml. Сейчас вас могут несколько смущать элементы вида {StaticResource XXXXXXXXXXXX} в XAML. Мы вернемся к этому вопросы при обсуждении стилей/шаблонов в одной из последующих частей, так что пока не будем обращать на них внимание.

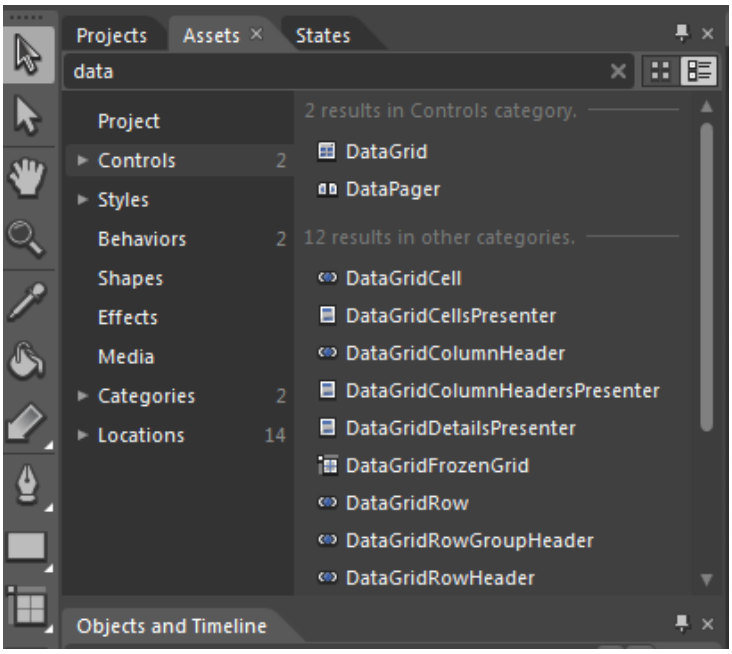
Если посмотреть на наш макет, нам нужно поле для ввода текста, кнопа и таблица для отображения данных. Давайте начнем делать дизайн страницы Search.xaml, используя Blend. Чтобы начать, в Visual Studio, щелкните правой копкой мыши по файлу Search.xaml и выберите Edit in Expression Belnd:



Поскольку Blend и Visual Studio используют одну и ту же проектную структуру, можно открыть файл в Blend и отредактировать его в визуальном редакторе XAML, перед тем как начать непосредственное кодирование.

В Blend сделаем нашему элементу Grid две строки: одну для поля ввода поискового запроса и кнопки, вторую для представления результата. В верхнюю строчку добавим элемент StackPanel, а в него уже добавим TextBox и Button, установив ориентацию панели в горизонтальную Orientation=Horizontal.

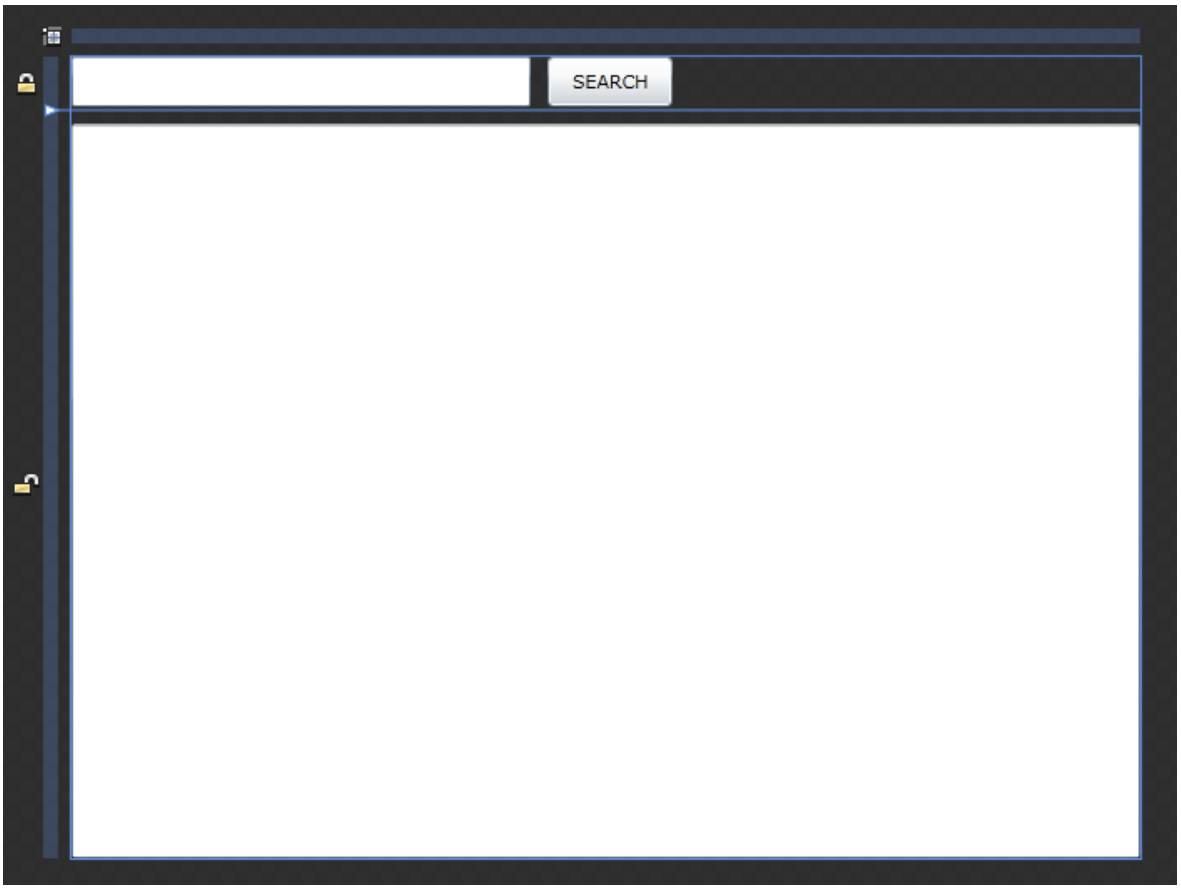
Следующее, что надо сделать – добавить элемент управления DataGrid для отображения наших данных. Поскольку DataGrid не входит в базовые элементы управления, он в библиотеках SDK, необходимо добавить на него ссылку. Это можно сделать разными способами. Blend это сделает автоматически. В Toolbox в строке поиска наберите data и найдите DataGrid:



Когда найдете, перетащите его во вторую строчку. Это автоматически добавит ссылку на SDK и XAML будет выглядеть похоже на следующий:

```
<navigation:Page
           xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
           xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
           xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
           xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
           xmlns:navigation="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Navigation"
           xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk" x:Class="TwitterSearchMon
itor.Views.Search"
           mc:Ignorable="d"
           d:DesignWidth="640" d:DesignHeight="480"
           Title="Search Page">
          <Grid x:Name="LayoutRoot">
                    <Grid.RowDefinitions>
                              <RowDefinition Height="32"/>
                              <RowDefinition/>
                    </Grid.RowDefinitions>
                    <StackPanel HorizontalAlignment="Left" Margin="0,-
32,0,0" VerticalAlignment="Top" Grid.Row="1"  Orientation="Horizontal">

          <TextBox x:Name="SearchTerm" FontSize="14.667" Margin="0,0,10,0" Width="275" TextWrapping="Wrap"/>
                              <Button x:Name="SearchButton" Width="75" Content="SEARCH"/>
                    </StackPanel>

                    <sdk:DataGrid x:Name="SearchResults" Margin="0,8,0,0" Grid.Row="1"/>

          </Grid>
</navigation:Page>
```

Обратите внимание на xaml:sdk в верхних строчках. В данном случае – добавлена ссылка на SDK, которое содержит дополнительные элементы управления Silverlight. Далее, определенный префикс sdk, используется для добавления элемента sdk:DataGrid. В результате внешний вид вашего приложения должен стать похожим на следующий:



Обратите внимание, что в нашем XAML для элементов управления определены x:Name для удобной работы с ними в дальнейшем в коде: SearchTerm для TextBox, SerachButton для Button и SearchResults для DataGrid.

Если теперь вернуться в VisualStudio можно увидеть запрос на перезагрузку проекта. Это связано с тем, Blend изменил файл проекта, добавив ссылку на элемент. Перезагрузите проект. Этот пример показывает, как инструментарий интегрирован на уровне файла проектов. Итак, перейдем к кодированию в Visual Studio.

Изменение настроек UriMapper. Установка Search.xaml страницей по умолчанию

Теперь, когда мы создали нашу страницу поиска Search (которая является домашней страницей приложения), давайте сделаем несколько изменений в навигации. В файле MainPage.xaml найдите элемент Frame и измените свойство Source так, чтобы сделать страницу Search.xaml страницей по умолчанию, также поменяйте в UriMapper страницу, на которую происходит переход при отсутствии пути. После редактирования, XAML должен выглядеть следующим образом:

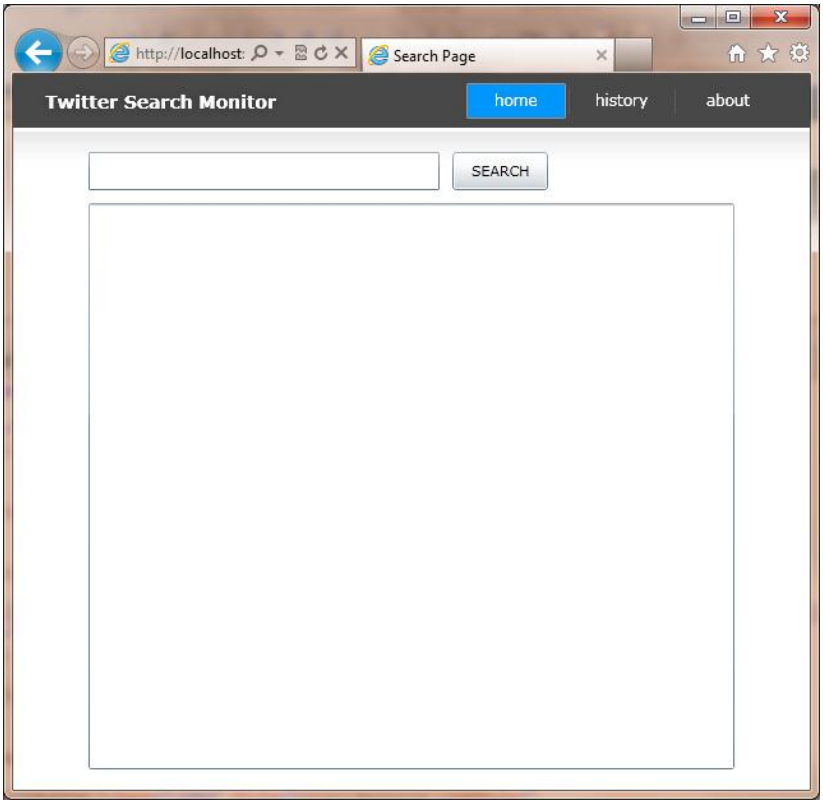
```
<Border x:Name="ContentBorder" Style="{StaticResource ContentBorderStyle}">
```

```
            <navigation:Frame x:Name="ContentFrame" Style="{StaticResource ContentFrameStyle}"
                              Source="/Search" Navigated="ContentFrame_Navigated" NavigationFailed="ContentFra
me_NavigationFailed">
                <navigation:Frame.UriMapper>
                  <uriMapper:UriMapper>
                    <uriMapper:UriMapping Uri="" MappedUri="/Views/Search.xaml"/>
                    <uriMapper:UriMapping Uri="/{pageName}" MappedUri="/Views/{pageName}.xaml"/>
                  </uriMapper:UriMapper>
                </navigation:Frame.UriMapper>
            </navigation:Frame>
        </Border>
```

Теперь нам нет необходимости в странице Home.xaml, удалите её. Добавьте страницу History.xaml и в файле MainPage.xaml поправьте содержимое LinksBorder в соответствии со сделанными изменениями.

```
<Border x:Name="LinksBorder" Style="{StaticResource LinksBorderStyle}">
                <StackPanel x:Name="LinksStackPanel" Style="{StaticResource LinksStackPanelStyle}">

                    <HyperlinkButton x:Name="Link1" Style="{StaticResource LinkStyle}"
                                     NavigateUri="/Search" TargetName="ContentFrame" Content="home"/>

                    <Rectangle x:Name="Divider1" Style="{StaticResource DividerStyle}"/>

                    <HyperlinkButton x:Name="Link2" Style="{StaticResource LinkStyle}"
                                     NavigateUri="/History" TargetName="ContentFrame" Content="history"/>

                    <Rectangle x:Name="Divider2" Style="{StaticResource DividerStyle}"/>

                    <HyperlinkButton x:Name="Link3" Style="{StaticResource LinkStyle}"
                                     NavigateUri="/About" TargetName="ContentFrame" Content="about"/>

                </StackPanel>
            </Border>
```

Теперь при запуске наше приложение будет выглядеть приблизительно следующим образом.



Итак, мы создали основу интерфейса пользователя приложения и можно переходить на следующий шаг.