Лабораторная Работа № 5

Исследование создания графического интерфейса

Цель работы: Ознакомиться основами создания интерфейса пользователя с помощью WPF.

Краткие теоретические сведения

Windows Presentation Foundation (WPF)

Windows Presentation Foundation — современный фреймворк пользовательского интерфейса для создания красивых приложений в среде .NET. На данный момент этот фреймворк может использоваться для создания десктоп-приложений (WPF), мобильных приложений (WP7, WP8) и веб приложений (Silverlight).

К созданию нового движка пользовательского интерфейса команду Microsoft подтолкнула масса проблем с существующим способом создания приложений (WinForms):

- Огромные объемы устаревшего (legacy) кода - отрисовка в WinForms происходит посредством использования родных функций Windows (WinApi) через GDI/GDI+ и библиотеку User32; помимо того, что библиотека присутствует со времен Windows 3.1, это накладывает существенные ограничения по возможностям движка;
- Устаревший подход к организации пользовательского интерфейса — интерфейс строится без оглядки на используемое разрешения экрана и монитора (dpi); компоненты формы как правило имеют фиксированныеразмеры, поскольку разработка форм поддерживающих увеличение очень трудоемка и не всегда приносит желаемый результат;
- Программная прорисовка — не используются ресурсы видеокарты;
- Медлительность;
- Интерфейс определяется в терминах программного кода, что ограничивает вовлеченность дизайнеров в процесс проектирования визуального интерфейса;
- Другие.

Фреймворк WPF призван улучшить способ разработки визуального представления приложений следующим образом:

- Использует DirectX API для прорисовки окна и его элементов — автоматически поддерживается аппаратное ускорение с откатом на программное в некоторых случаях;
- Не завязан на программный код — дизайнеры могут использовать такие средства визуального проектирования интерфейса как Expression Blend и передавать полученный исходный код разработчикам для дальнейшего внедрения и реализации;
- Упрощена работа с анимацией — анимация является неотъемлимой частью платформы;
- Упрощена работа с примитивами — 2d и 3d графикой;
- Доступна возможность изменения внешнего вида абсолютного любого элемента окна;
- Элементы автоматически поддерживают разные плотности экранов (dpi) а также могут изменять свой размер динамически в зависимости от настройки визуального представления при изменении размеров окна и сохранять приличный внешний вид;
- Окно не отрисовывается постоянно, как в WinForms (по событию WM_PAINT), а лишь когда есть изменения в визуальном представлении — значительно уменьшает объем потребляемых ресурсов и ускоряет отображение;
- Множество других.

XAML

Для разработки визуального интерфейса в WPF используется специальный синтаксис — eXtensible Application Markup Language (Расширяемый язык разметки приложений). Он представляет собой модифицированный XML.

Рассмотрим простейший код окна WPF:

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   Title="Window with Button"
```

```
    Width="250" Height="100">
  <!-- Add button to window -->
  <Button Name="button">Click Me!</Button>
</Window>
```

Как видно, есть корневой элемент Window, описывающий окно. При помощи аттрибутов Title, Width и Height можно выставить заголовок, ширину и высоту окна соответственно. Список доступных аттрибутов можно посмотреть при помощи Intellisense в среде разработки Microsoft Visual Studio; среди прочих можно задать минимальный и максимальный размер окна, его исходное размещение на экране.

Комментарии в XAML имеют тот же синтаксис, что и в XML - <!-- Комментарий -->.

Другие элементы представляют собой теги XML с соответствующим названием. Например, для кнопки определен тег Button и свой набор свойств. Важно отметить, что в WPF все элементы тем или иным образом входят как дочерние в другие элементы. Например, если необходимо отобразить картинку на кнопке вместо текста «Click me!» достаточно вставить внутрь ее объект Image:

```
<Button Name="button">
<Image Source="fun.png" />
</Button>
```

В XAML принято описывать лишь визуальную часть интерфейса. Вся логика взаимодействия, точно также как и в WinForms, может быть описана в code-behind файлах. У большинства элементов имеется известных из WinForms набор событий (event) для отработки тех или иных действий. Например, для кнопки событие Click (аналог OnClick в WinForms):

```
<Window
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   x:Class="SDKSample.AWindow"
   Title="Window with Button"
   Width="250" Height="100">
  <!-- Add button to window -->
  <Button Name="button" Click="button_Click">Click Me!</Button>
</Window>
using System.Windows;
namespace SDKSample
{
   public partial class AWindow : Window
   {
      public AWindow()
      {
         InitializeComponent();
      }
      void button_Click(object sender, RoutedEventArgs e)
      {
         MessageBox.Show("Hello, Windows Presentation Foundation!");
      }
   }
}
```

Для взаимодействия с элементами представления по-прежнему можно использовать их имена (в данном примере название кнопки — button). Однако,необходимо отметить, что имена являются опциональными (в отличие от Windows Forms) и как правило не выставляются, поскольку для взаимодействия с элементами визуального интерфейса используется механизм привязок (binding), который будет рассмотрен в следующих лабораторных работах.

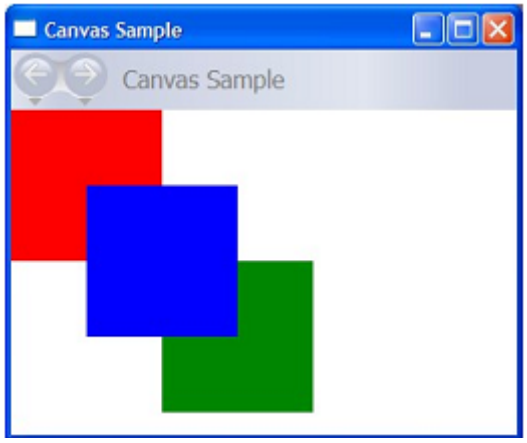
Способы организации элементов интерфейса в окне WPF

В предыдущем примере видно, что кнопка занимает весь размер окна, хотя нигде это не указано. Это происходит потому, что кнопка является единственным дочерним элементом окна, и ей не указаны никакие ограничения по размерам — она автоматически расширяется для заполнения объема пространства родительского элемента. В случае, если попытаться добавить две кнопки в окно — возникнет ошибка. Это связано с тем, что в WPF все элементы делятся на три группы: элементы, которые могут содержать множество других элементов (контейнеры); элементы, которые могут содержать один другой элемент (content control); элементы, не содержащие дочерних элементов.Для организации элементов на экране используются т.н. контейнеры. Вот перечень основных контейнеров: StackPanel, WrapPanel, DockPanel, Grid, Canvas.

Canvas

Канва (Canvas) — наиболее простой элемент организации элементов WPF, представляет собой доску для рисования. Каждый элемент, добавляемый на канву, определяет свое положение относительно левого верхнего угла канвы при помощи координат X и Y. Ввиду своей простоты является наиболее быстродействующим элементом организации контента в приложениях WPF и наиболее гибким, однако не умеет автоматически подстраиваться под изменение размеров окна (ввиду абсолютного позиционирования). Пример:

```
<Page WindowTitle="Canvas Sample"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <Canvas Height="400" Width="400">
    <Canvas Height="100" Width="100" Top="0" Left="0" Background="Red"/>
    <Canvas Height="100" Width="100" Top="100" Left="100" Background="Green"/>
    <Canvas Height="100" Width="100" Top="50" Left="50" Background="Blue"/>
  </Canvas>
</Page>
```

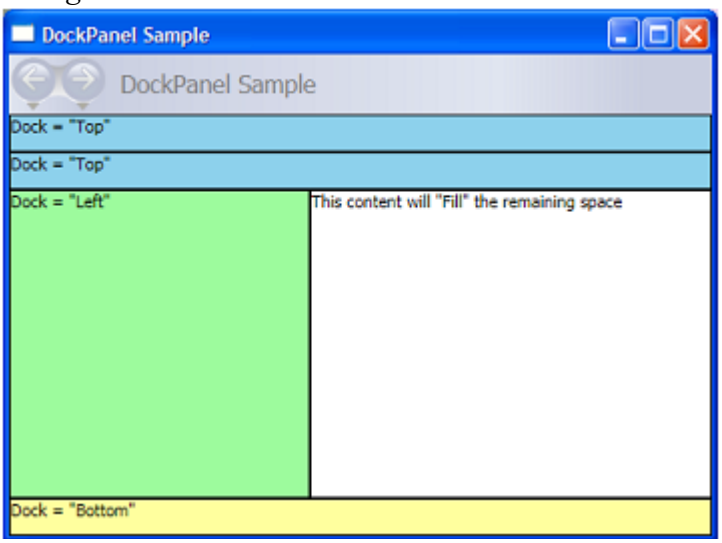


DockPanel

Панель стыковки (DockPanel) может быть известна WinForms разработчикам — позволяет «прикреплять» элементы управления к одному из краев родительского элемента (аналог свойства Dock в WinForms приложениях). Внутренние элементы занимают либо лишь необходимую часть пространства, либо последний элемент занимает все оставшее пространство. Пример:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="DockPanel
Sample">
  <DockPanel LastChildFill="True">
    <Border Height="25" Background="SkyBlue" BorderBrush="Black" BorderThickness="1"
DockPanel.Dock="Top">
      <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
    </Border>
    <Border Height="25" Background="SkyBlue" BorderBrush="Black" BorderThickness="1"
```

```
DockPanel.Dock="Top">
    <TextBlock Foreground="Black">Dock = "Top"</TextBlock>
  </Border>
  <Border Height="25" Background="LemonChiffon" BorderBrush="Black"
BorderThickness="1"
DockPanel.Dock="Bottom">
    <TextBlock Foreground="Black">Dock = "Bottom"</TextBlock>
  </Border>
  <Border Width="200" Background="PaleGreen" BorderBrush="Black"
BorderThickness="1"
DockPanel.Dock="Left">
    <TextBlock Foreground="Black">Dock = "Left"</TextBlock>
  </Border>
  <Border Background="White" BorderBrush="Black" BorderThickness="1">
    <TextBlock Foreground="Black">This content will "Fill" the remaining
space</TextBlock>
  </Border>
 </DockPanel>
</Page>
```

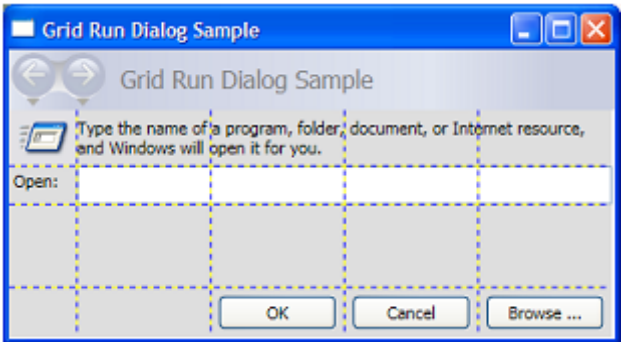


Grid

Сетка (Grid) позволяет позиционировать элементы при помощи таблицы, при этом с возможностью элементам занимать одновременно несколько клеток такой сетки. Сетка определяется при помощи строк (Grid.RowDefinitions) и колонок (Grid.ColumnDefinitions), но не обязательно имеет и строки и колонки. Строки и колонки сетки могут иметь различный размер, при этом можно указать сетке распределять размер между колонками равномерно (занимать остающийся размер родительстого контейнера).Дочерние элементы указывают номер строки и колонки, в которой хотят находится. Пример:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    WindowTitle="Grid Run Dialog Sample"
    WindowWidth="425"
    WindowHeight="225">
  <Grid Background="#DCDCDC"
      Width="425"
      Height="165"
      HorizontalAlignment="Left"
      VerticalAlignment="Top"
      ShowGridLines="True">
    <Grid.ColumnDefinitions>
```

```
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="*"/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="*" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>
  <Image Grid.Column="0" Grid.Row="0" Source="RunIcon.png" />
  <TextBlock Grid.Column="1" Grid.ColumnSpan="4" Grid.Row="0"
TextWrapping="Wrap">
    Type the name of a program, folder, document, or
    Internet resource, and Windows will open it for you.
  </TextBlock>
  <TextBlock Grid.Column="0" Grid.Row="1">Open:</TextBlock>
  <TextBox Grid.Row="1" Grid.Column="1" Grid.ColumnSpan="5" />
  <Button Margin="10, 0, 10, 15" Grid.Row="3" Grid.Column="2">OK</Button>
  <Button Margin="10, 0, 10, 15" Grid.Row="3" Grid.Column="3">Cancel</Button>
  <Button Margin="10, 0, 10, 15" Grid.Row="3" Grid.Column="4">Browse ...</Button>
 </Grid>
</Page
```



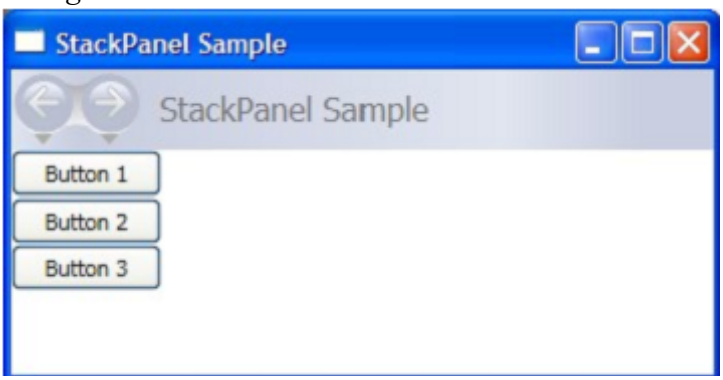
Обратим внимание: задать ширину колонки или высоту строки можно тремя способами: auto, * или ввести число. Auto означает, что строка будет занимать ровно столько, сколько необходимо ее контенту (дочерним элементам). Звездочка означает использовать оставшееся место равномерно; т. е. если указать четыре колонки с размером *, то они будут иметь одинаковый размер,вычисляемый по формуле: [свободная ширина] / 4. Более того, можно использовать множители вместе с «*», например 3*, или 2*. Это означает, что определенная колонка будет в 3 раза шире, чем колонки размера *, но при этом размер все равно считается исходя из оставшегося места. Числовое значение определяет абсолютную ширину колонки. Правилом хорошего тона считается задавание размера при помощи оператора «*» либо Auto, поскольку при изменении размера окна контейнер будет автоматически подстраиваться и увеличивать/уменьшать размеры строк/колонок.

StackPanel

Стековая панель позволяет организовывать элементы плоским списком, один за другим, в любом из двух направлений — вертикально либо горизонтально. При этом, если элементы выходят за границы панели (им необходимо больше места, чем есть в наличии у стековой панели) — они будут обрезаны по доступное пространство. Пример:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="StackPanel
```

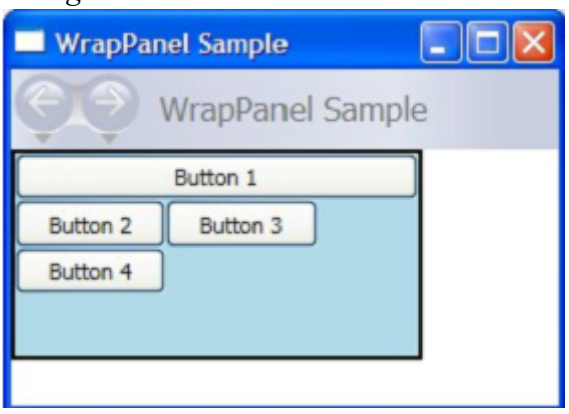
```
Sample">
   <StackPanel HorizontalAlignment="Left"
           VerticalAlignment="Top">
      <Button>Button 1</Button>
      <Button>Button 2</Button>
      <Button>Button 3</Button>
   </StackPanel>
</Page>
```



WrapPanel

Сверточная панель используется для заполнения свободного пространства элементами последовательно, слева направо с использованием переноса строки, подобно тому как текст заполняет страничку — по окончанию строки текст переходит на следующую. Пример:

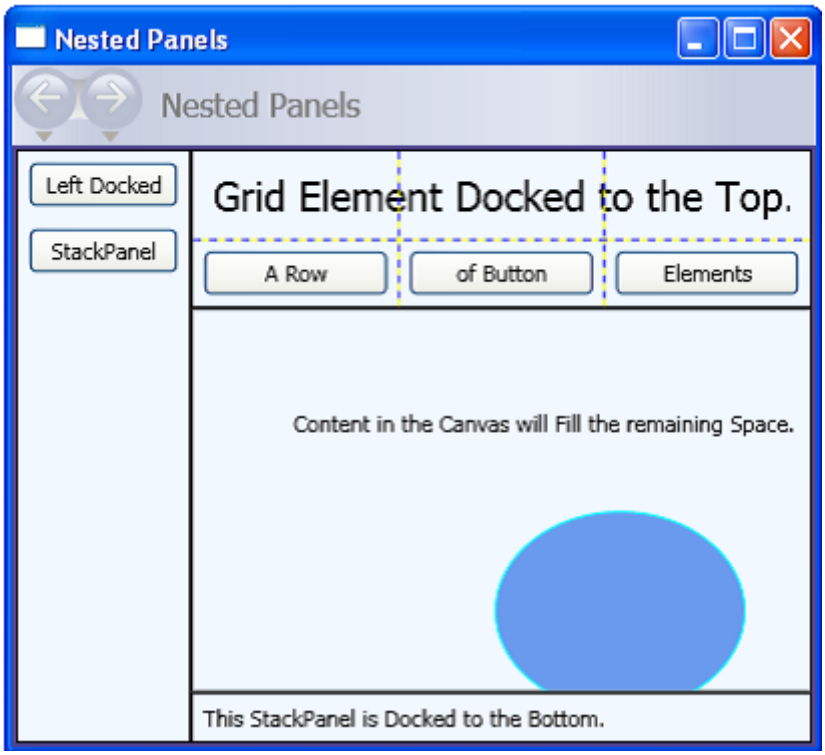
```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="WrapPanel
Sample">
  <Border HorizontalAlignment="Left" VerticalAlignment="Top" BorderBrush="Black"
BorderThickness="2">
      <WrapPanel Background="LightBlue" Width="200" Height="100">
         <Button Width="200">Button 1</Button>
         <Button>Button 2</Button>
         <Button>Button 3</Button>
         <Button>Button 4</Button>
      </WrapPanel>
  </Border>
</Page>
```



Использование нескольких панелей одновременно

Естественно, панели можно объединять для получения более точного контроля за позиционированием элементов в окне. Часто употребляется связка Grid + StackPanel — стековые панели размещаются в ячейках сетки и содержат в себе сразу несколько элементов, один за другим (например, label + textbox). Пример такого симбиоза:

```
<Page xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
WindowTitle="Nested
Panels">
  <Border Background="AliceBlue"
        Width="400"
        Height="300"
        BorderBrush="DarkSlateBlue"
        BorderThickness="2"
        HorizontalAlignment="Left"
        VerticalAlignment="Top">
    <DockPanel>
      <Border BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Left">
        <StackPanel>
          <Button Margin="5">Left Docked</Button>
          <Button Margin="5">StackPanel</Button>
        </StackPanel>
      </Border>
      <Border BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Top">
        <Grid ShowGridLines="True">
          <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition/>
          </Grid.RowDefinitions>
          <Grid.ColumnDefinitions>
            <ColumnDefinition />
            <ColumnDefinition />
            <ColumnDefinition />
          </Grid.ColumnDefinitions>
          <TextBlock FontSize="20" Margin="10" Grid.ColumnSpan="3" Grid.Row="0">Grid
Element Docked to the Top.</TextBlock>
          <Button Grid.Row="1" Grid.Column="0" Margin="5">A Row</Button>
          <Button Grid.Row="1" Grid.Column="1" Margin="5">of Button</Button>
          <Button Grid.Row="1" Grid.Column="2" Margin="5">Elements</Button>
        </Grid>
      </Border>
      <Border BorderBrush="Black" BorderThickness="1" DockPanel.Dock="Bottom">
        <StackPanel Orientation="Horizontal">
          <TextBlock Margin="5">This StackPanel is Docked to the Bottom.</TextBlock>
        </StackPanel>
      </Border>
      <Border BorderBrush="Black" BorderThickness="1">
        <Canvas ClipToBounds="True">
          <TextBlock Canvas.Top="50" Canvas.Left="50">
            Content in the Canvas will Fill the remaining Space.
          </TextBlock>
          <Ellipse Height="100" Width="125" Fill="CornflowerBlue" Stroke="Aqua"
Canvas.Top="100" Canvas.Left="150"/>
        </Canvas>
      </Border>
    </DockPanel>
  </Border>
</Page>
```



Основные элементы управления

Основные элементы управления в WPF весьма похожи на WinForms:

- Button — кнопка;
- ToggleButton — кнопка с двумя состояниями, определяемых свойством IsChecked;
- TextBlock — легковесный аналог Label;
- Label — подпись;
- ListBox — список элементов, основные свойства — Items, ItemsSource;
- TabPanel — панель с вкладками (табами);
- Image — изображение, устанавливается при помощи свойства Source;
- TextBox — поле для ввода текста;
- Border — граница, рамка;
- Expander — выпадающий элемент;
- ContextMenu, Menu — контекстное меню и меню;
- CheckBox - «галочка»;
- ComboBox — выпадающий список;

Задание

1. Ознакомиться с краткими теоретическими сведениями.
2. Разработать визуальный интерфейс при помощи технологии WPF согласно ранее спроектированному. Не менее ¾ всех форм должны быть разработаны полностью (визуально) — без отработки логики.
3. Оформить отчет о выполненной работе.

Контрольные вопросы

1. Что представляет собой платформа WPF ?
2. Какие проблемы платформы WinForms решает WPF ?