GUI на WPF

Часть 1

Юрий Литвинов
yurii.litvinov@gmail.com

10.11.2020г

Windows Presentation Foundation

- Появилась в .NET 3.0, как альтернатива WinForms
  - Использует DirectX для отображения контролов
- Отделение разметки пользовательского интерфейса от кода — язык XAML (eXtensible Application Markup Language)
  - Специальная среда разработки — Microsoft Blend
- Несколько “веток” WPF — Silverlight, Windows Runtime XAML Framework
- Архитектурно сильно отличается от WinForms, несколько сложнее в изучении
  - Data binding
    - Паттерн Model-View-Viewmodel (MVVM)
  - Templates (Styles)
  - Resources

XAML

eXtensible Application Markup Language

- На самом деле, язык описания правил создания и инициализации произвольных объектов
  - Есть отдельный XAML-парсер, позволяющий создавать дерево объектов по XAML-описанию
  - Не путать с JSON и механизмами сериализации
- Базируется на XML
  - Тэги, атрибуты, пространства имён

XAML:

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  Content="OK" />
```

C#:

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
b.Content = "OK";
```

“Полная форма” записи атрибутов

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
   <Button.Content>
      <Rectangle Height="40" Width="40" Fill="Black" />
   </Button.Content>
</Button>
```

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
System.Windows.Shapes.Rectangle r = new System.Windows.Shapes.Rectangle();
r.Width = 40;
r.Height = 40;
r.Fill = System.Windows.Media.Brushes.Black;
b.Content = r;
```

Конвертеры типов

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   Content="OK" Background="White" />
```

```
System.Windows.Controls.Button b = new System.Windows.Controls.Button();
b.Content = "OK";
b.Background = System.Windows.Media.Brushes.White;
```

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   Content="OK" />
   <Button.Background>
      <SolidColorBrush Color="White" />
   </Button.Background>
</Button>
```

Расширения

- Похожи на конвертеры
  - Возможность вызывать произвольный код в процессе создания объекта
- Есть куча встроенных расширений, можно писать свои

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Background="{x:Null}"
  Height="{x:Static SystemParameters.IconHeight}"
  Content="{Binding Path=Height, RelativeSource={RelativeSource Self}}" />
```

Дети элементов

- Content
- Элементы коллекции
- Результат вызова конвертеров

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <Button.Content>
    <Rectangle Height="40" Width="40" Fill="Black" />
  </Button.Content>
</Button>
```

```
<Button xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <Rectangle Height="40" Width="40" Fill="Black" />
</Button>
```

Коллекции

- Списки

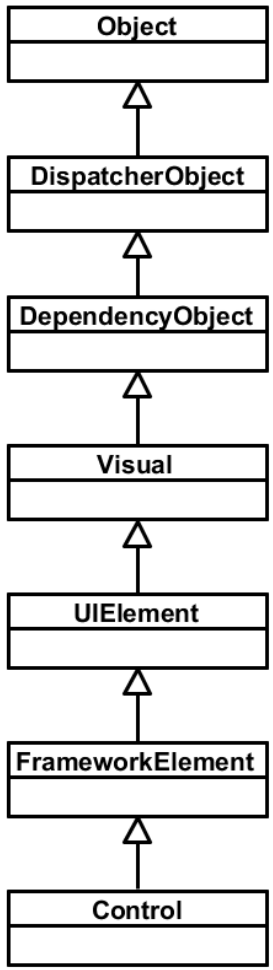
```
<ListBox xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
  <ListBox.Items>
    <ListBoxItem Content="Item 1" />
    <ListBoxItem Content="Item 2" />
  </ListBox.Items>
</ListBox>
```

- Словари

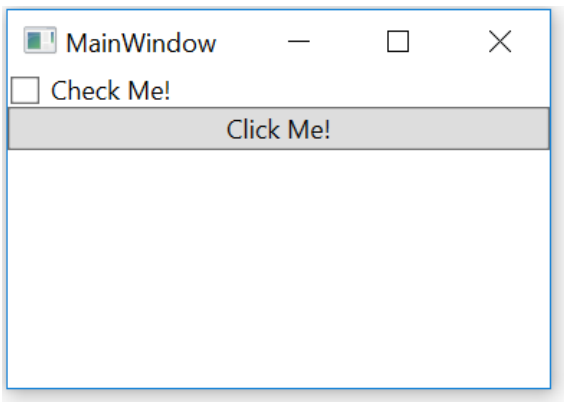
```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Color x:Key="1" A="255" R="255" G="255" B="255" />
  <Color x:Key="2" A="0" R="0" G="0" B="0" />
</ResourceDictionary>
```

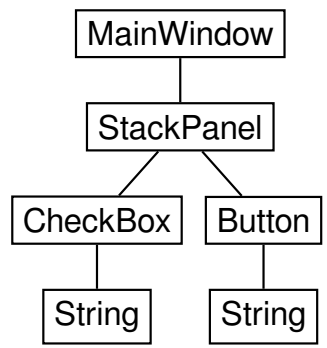
Структура классов WPF

- DispatcherObject — потоки и сообщения
- DependencyObject — продвинутая работа со свойствами
- Visual — общение с движком рисования
- UIElement — лейаут, события
- FrameworkElement — ещё лейаут, стили
- Control — шаблоны

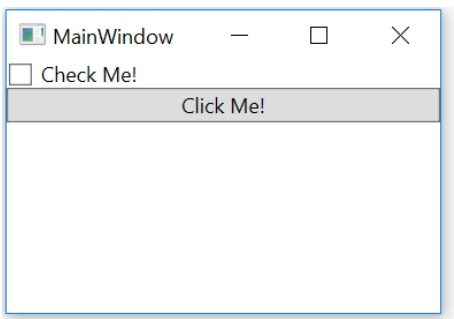


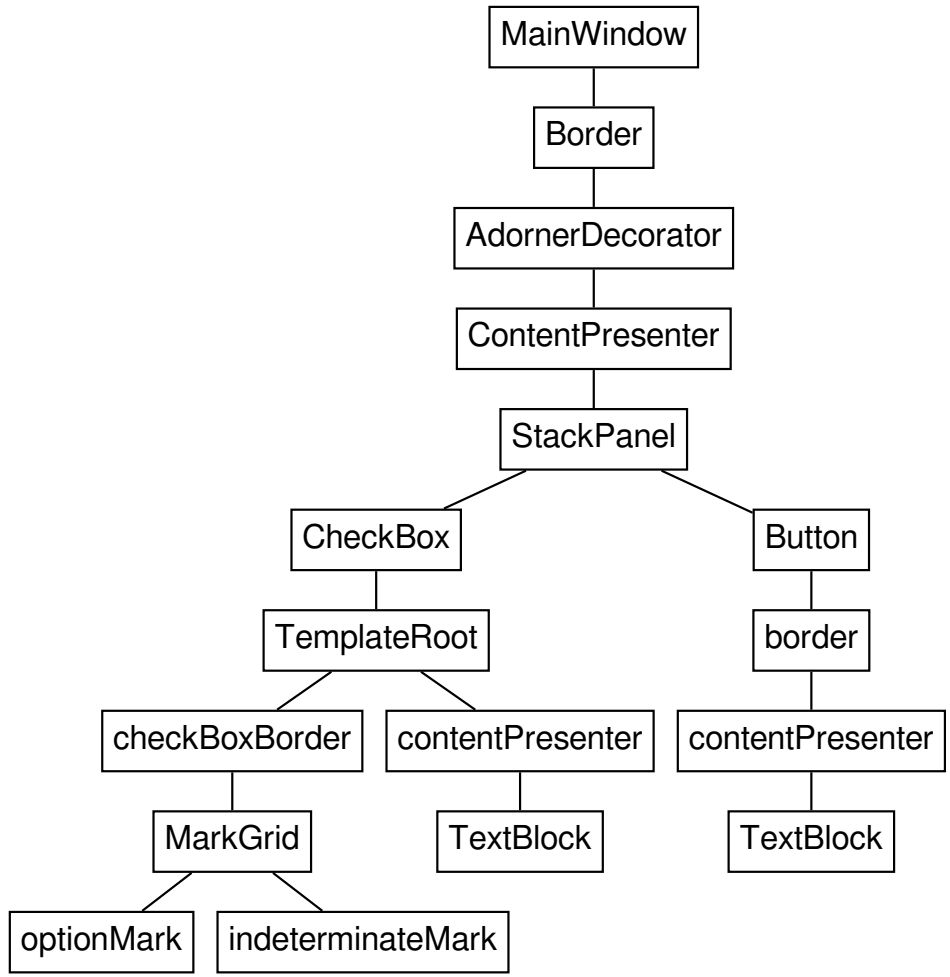
Логическое дерево





Визуальное дерево





Dependency Properties

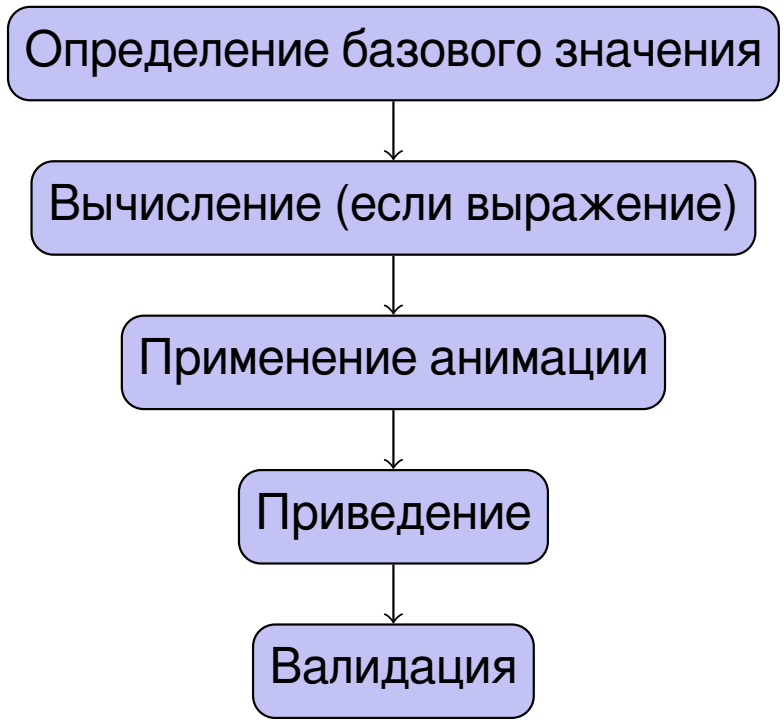
- *Зависят* от “провайдеров”, на основании которых они вычисляют своё текущее значение
- Похожи на обычные свойства, но:
  - Обеспечивают оповещение об изменениях
  - Позволяют наследовать значения свойств от предка в логическом дереве
  - Позволяют добавлять объекту свойства, которых у него не было
- Реализуются как обычные свойства с некоторой дополнительной машинерией, которая прячет за собой хеш-таблицу
- Нужны, чтобы можно было легко менять свойства, делать анимацию и подобного рода вещи прямо из XAML-а
  - Декларативность и Data-Driven Development — базовые принципы архитектуры WPF

Пример реализации зависимого свойства

```
public class Button: ButtonBase
{
   public static readonly DependencyProperty IsDefaultProperty;

   static Button()
   {
      Button.IsDefaultProperty = DependencyProperty.Register(
         "IsDefault",
         typeof(bool),
         typeof(Button),
         new FrameworkPropertyMetadata(
            false,
            new PropertyChangedCallback(OnIsDefaultChanged)
            )
         );
   }

   public bool IsDefault
   {
      get => (bool) GetValue(Button.IsDefaultProperty);
      set => SetValue(Button.IsDefaultProperty, value);
   }

   private static void OnIsDefaultChanged(DependencyObject o,
      DependencyPropertyChangedEventArgs e)
   {
      ...
   }
}
```

Порядок вычисления зависимых свойств

1. Локальное значение
2. Триггер шаблона родителя
3. Шаблон родителя
4. Триггеры стиля
5. Триггеры шаблона
6. Сеттеры стиля
7. Триггеры темы
8. Сеттеры темы
9. Унаследованное значение
10. Значение по умолчанию

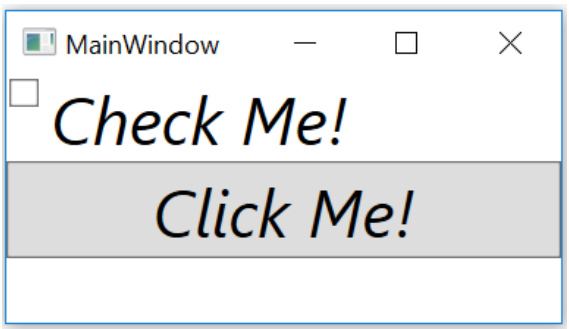


Пример триггера стиля

```
<Button Content="Click Me!">
   <Button.Style>
      <Style TargetType="{x:Type Button}">
         <Style.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
               <Setter Property="Foreground" Value="Blue"/>
            </Trigger>
         </Style.Triggers>
      </Style>
   </Button.Style>
</Button>
```

Attached Properties

```
<StackPanel TextElement.FontSize="30" TextElement.FontStyle="Italic">
    <CheckBox Content="Check Me!"/>
    <Button Content="Click Me!"/>
</StackPanel>
```



Routed Events

```
public class Button : ButtonBase
{
   public static readonly RoutedEvent ClickEvent;

   static Button()
   {
      Button.ClickEvent = EventManager.RegisterRoutedEvent("Click",
         RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(Button));
   }

   public event RoutedEventHandler Click
   {
      add { AddHandler(Button.ClickEvent, value); }
      remove { RemoveHandler(Button.ClickEvent, value); }
   }

   protected override void OnMouseLeftButtonDown(MouseButtonEventArgs e)
   {
      RaiseEvent(new RoutedEventArgs(Button.ClickEvent, this));
   }
   ...
}
```

Routed Events, события

- Стратегии маршрутизации:
  - **Bubbling** — снизу вверх
    - Контролы в WPF по соглашению реагируют только на них
  - **Tunneling** — сверху вниз
    - По соглашению имеют префикс Preview и парное Bubbling-событие
  - **Direct** — как обычные события в C#
- RoutedEventArgs:
  - **Source** — элемент логического дерева
  - **OriginalSource** — элемент визуального дерева
  - **Handled** — заканчивает распространение события
    - На самом деле, нет, можно подписаться и на обработанное
    - Обработка PreviewX отменяет и bubbling-событие X

Подробнее про Data Binding

```
void treeView_SelectedItemChanged(object sender,
    RoutedPropertyChangedEventArgs<object> e)
{
    currentFolder.Text = (treeView.SelectedItem as TreeViewItem).Header.ToString();
    Refresh();
}
```

```
public MainWindow()
{
    InitializeComponent();
    Binding binding = new Binding();
    binding.Source = treeView;
    binding.Path = new PropertyPath("SelectedItem.Header");
    currentFolder.SetBinding(TextBlock.TextProperty, binding);
}
```

То же в XAML

```
public MainWindow()
{
    InitializeComponent();
    Binding binding = new Binding();
    binding.Source = treeView;
    binding.Path = new PropertyPath("SelectedItem.Header");
    currentFolder.SetBinding(TextBlock.TextProperty, binding);
}
```

```
<TextBlock x:Name="currentFolder" DockPanel.Dock="Top"
    Text="{Binding ElementName=treeView, Path=SelectedItem.Header}"
    Background="AliceBlue" FontSize="16" />
```

Привязка к “обычным” свойствам

```
<Label x:Name="numItemsLabel"
  Content="{Binding Source={StaticResource photos}, Path=Count}"
  DockPanel.Dock="Bottom"/>
```

Так оно не будет обновляться, надо, чтобы:

- был реализован INotifyPropertyChanged
- было событие XXXChanged, где XXX — имя свойства
- для коллекций — ObservableCollection

Target — всегда DependencyProperty

Привязка “составных” контролов к коллекциям

```
<ListBox x:Name="pictureBox" DisplayMemberPath="Name"
   ItemsSource="{Binding Source={StaticResource photos}}" ...>
   ...
</ListBox>
```

```
<ListBox x:Name="pictureBox"
   ItemsSource="{Binding Source={StaticResource photos}}" ...>
   <ListBox.ItemTemplate>
      <DataTemplate>
         <Image Source="{Binding Path=FullPath}" Height="35"/>
      </DataTemplate>
   </ListBox.ItemTemplate>
   ...
</ListBox>
```

DataContext

```
<StackPanel DataContext="{StaticResource photos}">
   <Label x:Name="numItemsLabel"
      Content="{Binding Path=Count}" .../>
   ...
   <ListBox x:Name="pictureBox" DisplayMemberPath="Name"
      ItemsSource="{Binding}" ...>
      ...
   </ListBox>
   ...
</StackPanel>
```

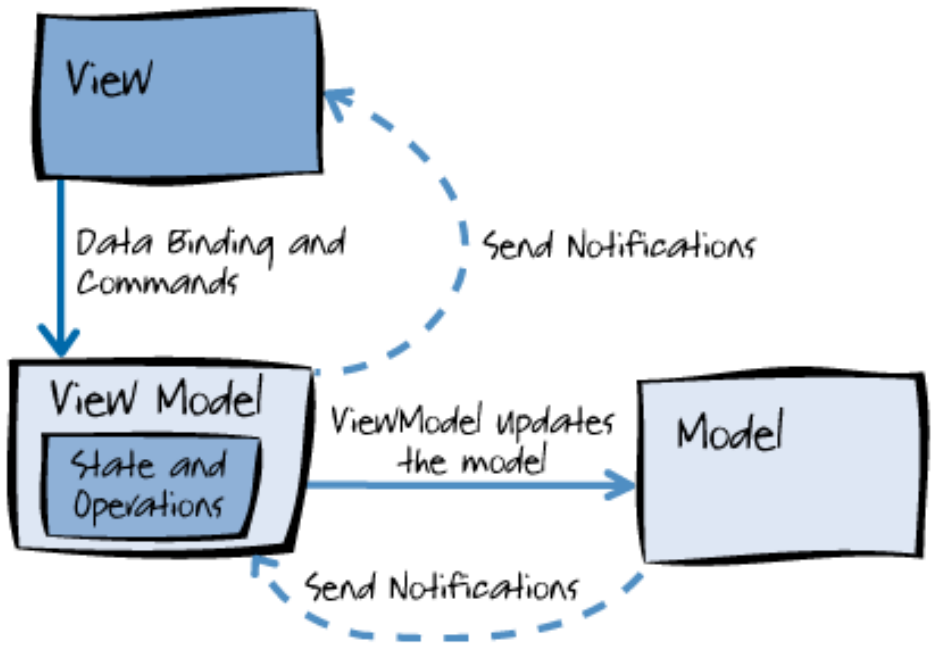
Конвертеры

```
<Window.Resources>
   <local:CountToBackgroundConverter x:Key="myConverter"/>
</Window.Resources>
...
<Label Background="{Binding Path=Count, Converter={StaticResource myConverter},
   Source={StaticResource photos}}" .../>


public class CountToBackgroundConverter : IValueConverter
{
   public object Convert(object value, Type targetType, object parameter,
      CultureInfo culture)
   {
      if (targetType != typeof(Brush))
         throw new InvalidOperationException("The target must be a Brush!");
      int num = int.Parse(value.ToString());
      return (num == 0 ? Brushes.Yellow : Brushes.Transparent);
   }

   public object ConvertBack(object value, Type targetType, object parameter,
      CultureInfo culture)
   {
      return DependencyProperty.UnsetValue;
   }
}
```

Направления привязки

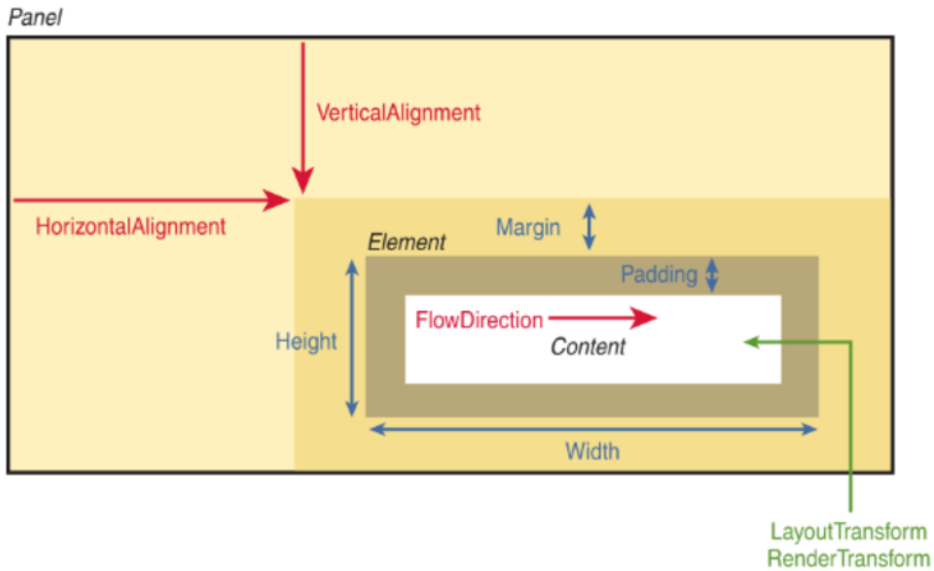
- OneWay — то, что мы делали до этого
  - По умолчанию для большинства свойств
- TwoWay — в обе стороны, для редактируемых контролов
  - По умолчанию для, например, TextBox.Text
  - UpdateSourceTrigger — PropertyChanged, LostFocus, Explicit
- OneWayToSource — от цели к источнику, для полей ввода
- OneTime — без нотификаций об изменении вовсе

Паттерн “Model-View-View Model”



© https://msdn.microsoft.com/en-us/library/hh848246.aspx

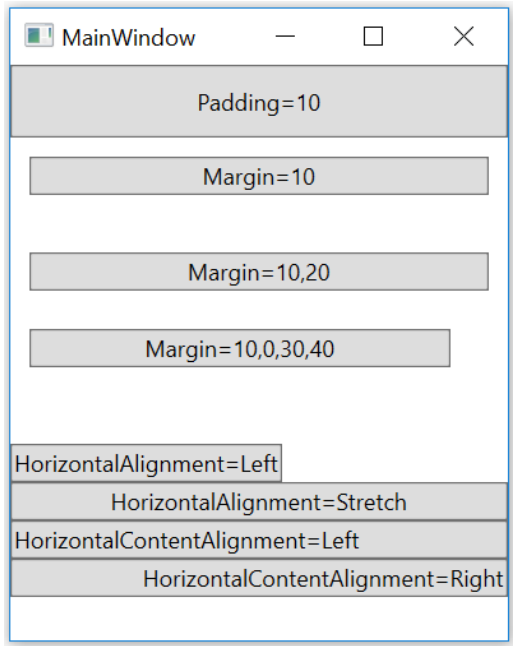
Геометрия контрола



© Из книги Adam Nathan, WPF 4.5 Unleashed.

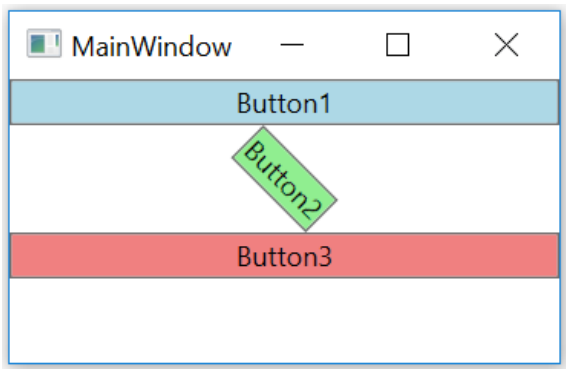
Управление положением элемента

- Абсолютное
  - Padding
  - Margin
  - Тип System.Windows.Thickness
- Выравнивание внутри родителя
  - HorizontalAlignment, VerticalAlignment
  - HorizontalContentAlignment, VerticalContentAlignment

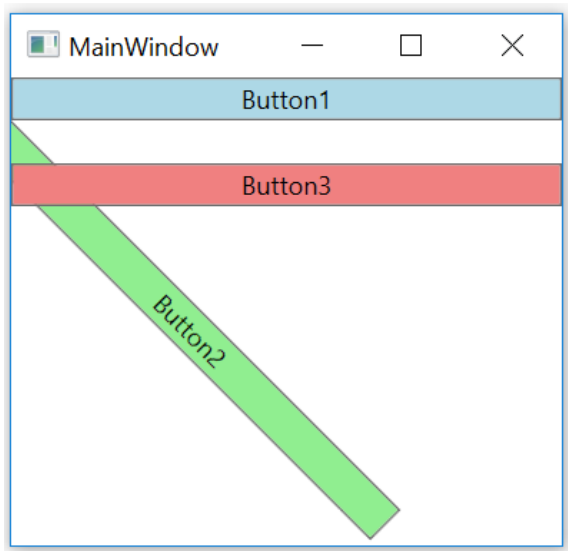


Преобразования системы координат

```
<StackPanel>
  <Button Content="Button1" Background="LightBlue"/>
  <Button Content="Button2" Background="LightGreen" >
    <Button.LayoutTransform>
      <RotateTransform Angle="45"/>
    </Button.LayoutTransform>
  </Button>
  <Button Content="Button3" Background="LightCoral"/>
</StackPanel>
```




```
<StackPanel>
  <Button Content="Button1" Background="LightBlue"/>
  <Button Content="Button2" Background="LightGreen" >
    <Button.RenderTransform>
      <RotateTransform Angle="45"/>
    </Button.RenderTransform>
  </Button>
  <Button Content="Button3" Background="LightCoral"/>
</StackPanel>
```



Литература

Adam Nathan, WPF 4.5 Unleashed. Sams Publishing, 2013. 864pp.