WPF

Kontrolki

Krzysztof Mossakowski
Wydział Matematyki i Nauk Informacyjnych Politechniki Warszawskiej
http://www.mini.pw.edu.pl/~mossakow

Modele zawartości

- WPF wykorzystuje 4 modele zawartości kontrolek:
  - **ContentControl** – pojedyncza zawartość
    - używane przez: **Button, ButtonBase, CheckBox, ComboBoxItem, ContentControl, Frame, GridViewColumnHeader, GroupItem, Label, ListBoxItem, ListViewItem, NavigationWindow, RadioButton, RepeatButton, ScrollViewer, StatusBarItem, ToggleButton, ToolTip, UserControl, Window**
  - **HeaderedContentControl** – nagłówek i pojedyncza zawartość
    - **Expander, GroupBox, TabItem**
  - **ItemsControl** – kolekcja pozycji zawartości
    - **ComboBox, ContextMenu, ListBox, ListView, Menu, StatusBar, TabControl, TreeView**
  - **HeaderedItemsControl** – nagłówek i kolekcja pozycji zawartości
    - **MenuItem, ToolBar, TreeViewItem**

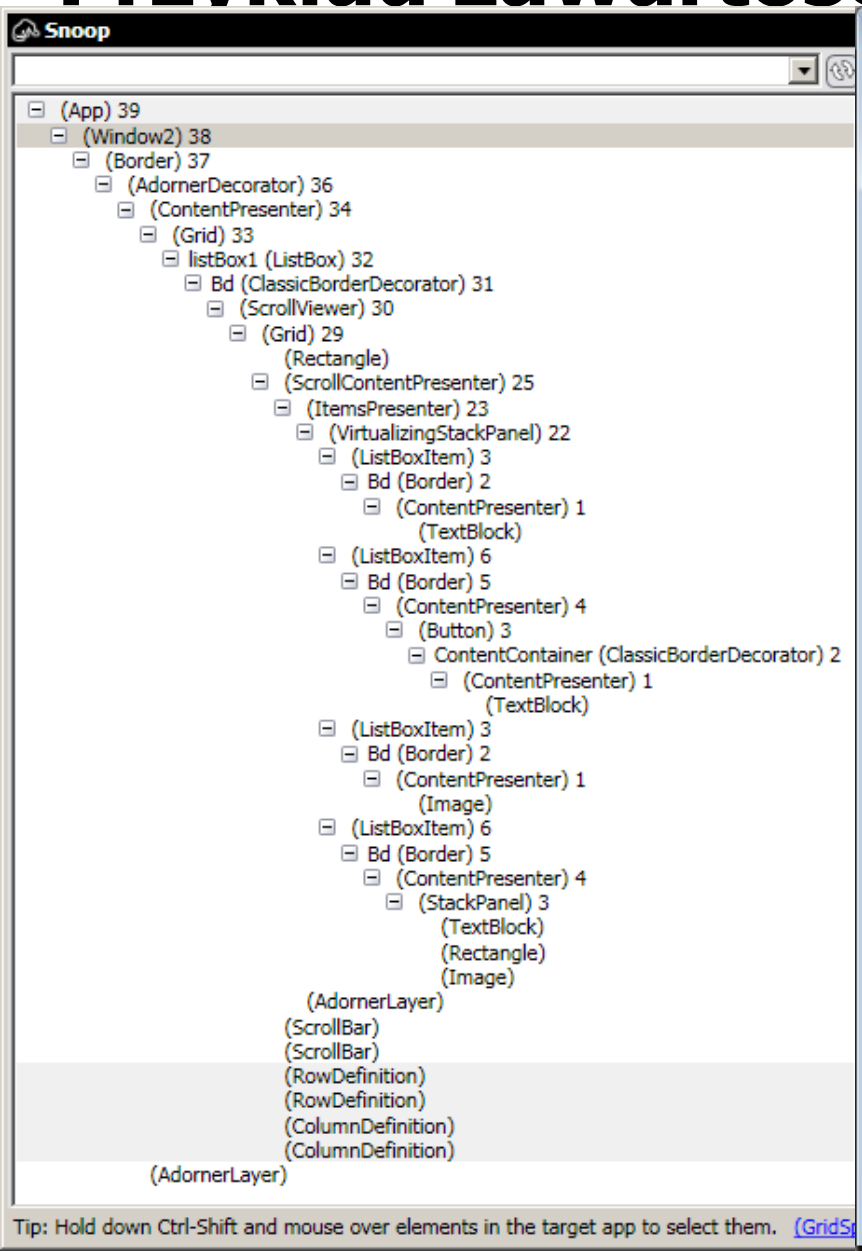
Zawartość

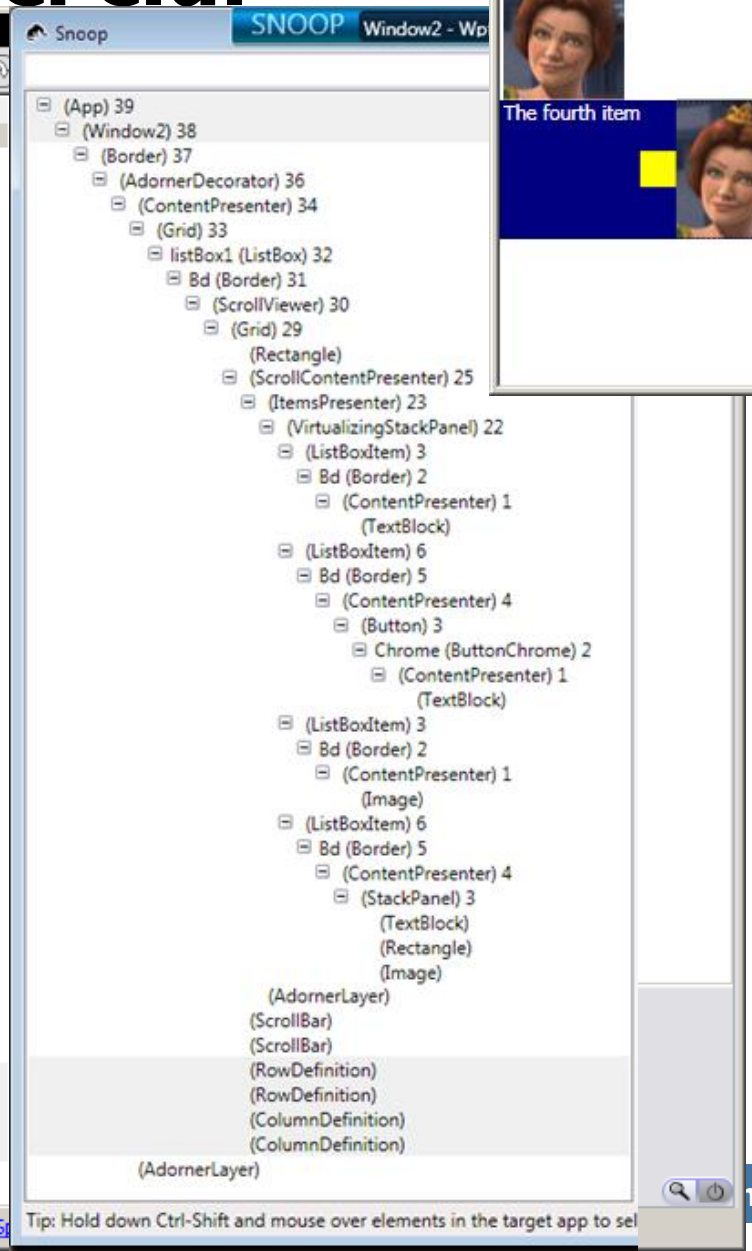
- Klasa **ContentControl** reprezentuje kontrolkę z pojedynczą zawartością
- Klasa **ContentPresenter** jest odpowiedzialna za wyświetlanie zawartości obiektu **ContentControl**; wybierana jest pierwsza pasująca możliwość:
  1. Jeśli **Content** jest typu **UIElement**, to dodaj do drzewa wyświetlania
  2. Jeśli ustawiony jest **ContentTemplate** , to utwórz obiekt **UIElement** i dodaj go do drzewa wyświetlania
  3. Jeśli ustawiony jest **ContentTemplateSelector** , to znajdź wzorze, utwórz wg niego obiekt **UIElement** i dodaj do drzewa
  4. Jeśli obiekt **Content** ma wzorzec danych, to użyj go do stworzenia obiektu **UIElement** i dodaj do drzewa
  5. Jeśli obiekt **Content** ma zdefiniowany **TypeConverter** do typu **UIElement**, to skonwertuj obiekt i dodaj do drzewa
  6. Jeśli obiekt **Content** ma zdefiniowany **TypeConverter** do klasy **String**, to skonwertuj, utwórz dla powstałego napisu **TextBlock** i dodaj do drzewa
  7. W ostateczności wywołaj metodę **ToString** dla obiektu **Content**, otrzymany napisz umieść w **TextBlock** i dodaj go do drzewa

Przykład zawartości

```
<Window x:Class="WpfApplication1.Window2"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window2" Height="300" Width="300">
     <Grid>
         <ListBox Name="listBox1" VerticalAlignment="Stretch">
             <ListBoxItem>
                 The first item
             </ListBoxItem>
             <ListBoxItem>
                 <Rectangle Fill="Yellow" Height="20" Width="20"></Rectangle>
             </ListBoxItem>
             <ListBoxItem>
                 <Image Source="Images/Fiona_67x77.gif"></Image>
             </ListBoxItem>
             <ListBoxItem>
                 <StackPanel Orientation="Horizontal">
                     <TextBlock>The fourth item</TextBlock>
                     <Rectangle Fill="Yellow" Height="20" Width="20"></Rectangle>
                     <Image Source="Images/Fiona_67x77.gif"></Image>
                 </StackPanel>
             </ListBoxItem>
         </ListBox>
     </Grid>
</Window>
```

Przykład zawartości c.d.

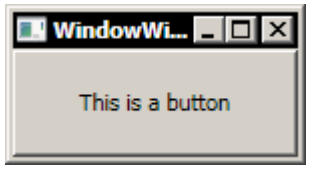


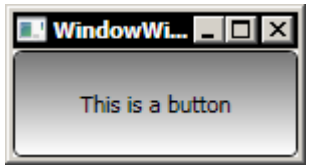


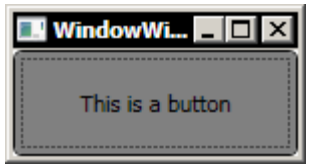
Wzorce

- Wzorce pozwalają na tworzenie nowych drzew wyświetlania
- Wykorzystywane są 2 typy wzorców:
  - **ControlTemplate** tworzy drzewo wyświetlania dla kontrolki
  - **DataTemplate** tworzy drzewo wyświetlania dla danych
- **ControlTemplate** może być wykorzystany do modyfikacji zarówno wyglądu jak i struktury kontrolki
  - jeśli nie został stworzony nowy **ControlTemplate** dla kontrolki, to zostanie użyty domyślny wg ustawień systemu
  - nie ma możliwości zmiany tylko części **ControlTemplate**, zawsze definicja musi być kompletna
- *Triggers* zdefiniowane we wzorcu pozwalają na modyfikację wartości właściwości oraz wykonywanie akcji w odpowiedzi na taką modyfikację (np. animację)

```
<Window x:Class="WpfApplication1.WindowWithButton"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="WindowWithButton" Height="80" Width="150">
  <Button>
    <Button.Template>
      <ControlTemplate TargetType='{x:Type Button}'>
        <Border Name="MyBorder" CornerRadius='4' BorderThickness='1'>
          <Border.BorderBrush>
            <SolidColorBrush Color="Black"/>
          </Border.BorderBrush>
          <Border.Background>
            <LinearGradientBrush EndPoint='0,1'>
              <LinearGradientBrush.GradientStops>
                <GradientStop Offset='0' Color='#FF777777' />
                <GradientStop Offset='1' Color='#FFFFFFFF' />
              </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
          </Border.Background>
          <ContentPresenter HorizontalAlignment='Center' VerticalAlignment='Center' />
        </Border>
        <ControlTemplate.Triggers>
          <Trigger Property="Button.IsPressed" Value="True">
            <Setter TargetName="MyBorder" Property="Border.Background"
             Value="{DynamicResource {x:Static SystemColors.ControlDarkBrushKey}}" />
          </Trigger>
        </ControlTemplate.Triggers>
      </ControlTemplate>
    </Button.Template>
    This is a button
  </Button>
</Window>
```



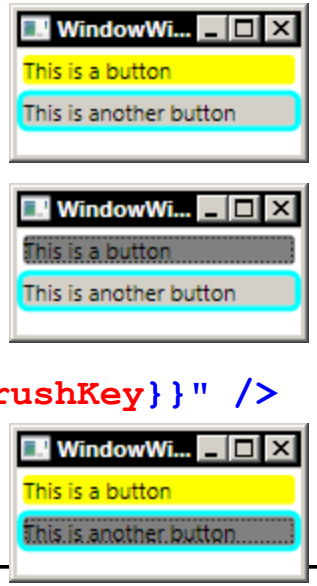




Wiązanie wzorców

- Wiązanie wzorców pozwala na ich parametryzowanie
  - do parametryzacji można wykorzystać właściwości

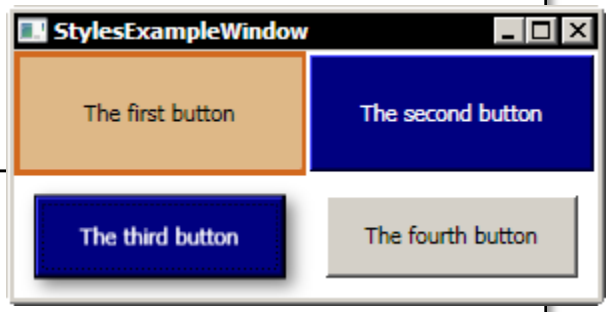
```
<ControlTemplate x:Key="MyButtonTemplate" TargetType='{x:Type Button}'>
  <Border Name="MyBorder" CornerRadius='4'
     BorderThickness='{TemplateBinding Property=BorderThickness}'
     BorderBrush='{TemplateBinding Property=BorderBrush}'
     Background='{TemplateBinding Property=Background}'>
    <ContentPresenter />
  </Border>
  <ControlTemplate.Triggers>
    <Trigger Property="Button.IsPressed" Value="True">
      <Setter TargetName="MyBorder" Property="Border.Background"
         Value="{DynamicResource {x:Static SystemColors.ControlDarkBrushKey}}" />
    </Trigger>
  </ControlTemplate.Triggers>
</ControlTemplate>
```




```
<Button Template="{StaticResource MyButtonTemplate}" Background="Yellow" >
  This is a button
</Button>
<Button Template="{StaticResource MyButtonTemplate}" BorderBrush="Cyan">
  This is another button
</Button>
```

Style

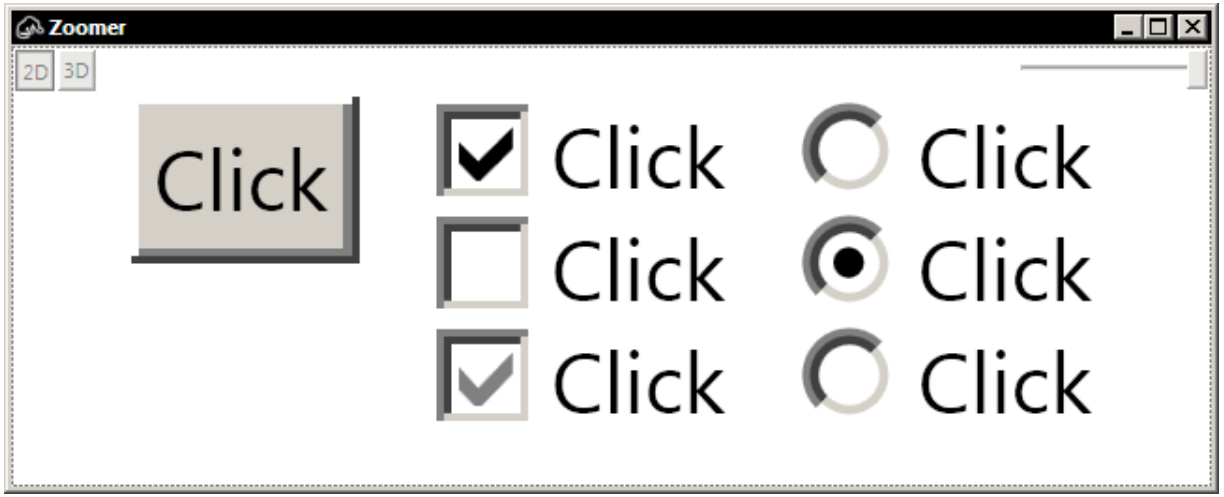
```
<Window.Resources>
    <Style x:Key="NavyButton" TargetType="{x:Type Button}">
        <Setter Property="Background" Value="Navy"/>
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="Margin" Value="2"/>
    </Style>

    <Style x:Key="ShadowedNavyButton"
           BasedOn="{StaticResource NavyButton}"
           TargetType="{x:Type Button}">
        <Setter Property="BitmapEffect">
            <Setter.Value>
                <DropShadowBitmapEffect Opacity ="0.5"/>
            </Setter.Value>
        </Setter>
    </Style>
</Window.Resources>
```



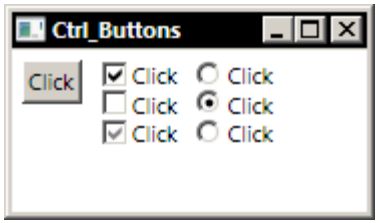

```
<Button Grid.Column="1"
        Style="{StaticResource NavyButton}">
    The second button
</Button>
<Button Grid.Row="1" Margin="10"
        Style="{StaticResource ShadowedNavyButton}">
    The third button
</Button>
```

Przyciski

- ButtonBase
  - Button
  - ToggleButton
    - CheckBox
    - RadioButton



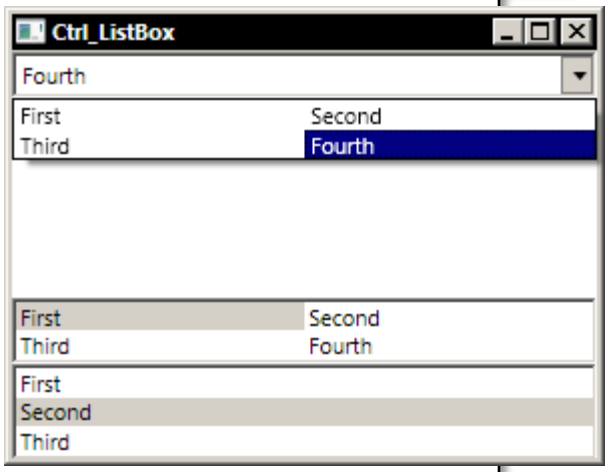

```
<StackPanel Orientation='Horizontal'>
  <Button Margin='5' VerticalAlignment='Top'>Click</Button>
  <StackPanel Margin='5' VerticalAlignment='Top'>
    <CheckBox IsChecked='True'>Click</CheckBox>
    <CheckBox IsChecked='False'>Click</CheckBox>
    <CheckBox IsChecked='{x:Null}'>Click</CheckBox>
  </StackPanel>
  <StackPanel Margin='5' VerticalAlignment='Top'>
    <RadioButton>Click</RadioButton>
    <RadioButton IsChecked='True'>Click</RadioButton>
    <RadioButton>Click</RadioButton>
  </StackPanel>
</StackPanel>
```



Listy

- **ListBox** i **ComboBox**
  - Jako źródło danych wyświetlanych na liście może zostać użyty dowolny obiekt listy (**IList**) lub dowolny obiekt implementujący interfejs **IEnumerable**
    - właściwość **ItemsSource**
    - jako źródło danych może być także wykorzystana kolekcja **ObservableCollection<T>**
  - We właściwości **ItemsPanel** można podać wzorzec, który zostanie użyty do stworzenia wizualnej reprezentacji każdej wyświetlanej na liście pozycji
    - **ItemsPanel** jest typu **ItemsPanelTemplate**, który wymaga, by wzorzec stworzył panel
    - to pozwala na stworzenie dowolnego układu i wyglądu pozycji kontrolki **ListBox** lub **ComboBox**

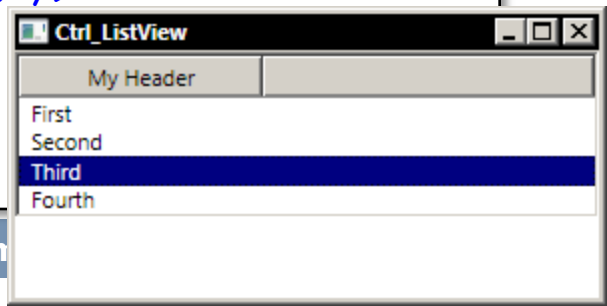
```
<StackPanel>
  <ComboBox>
    <ComboBox.ItemsPanel>
      <ItemsPanelTemplate>
        <UniformGrid Columns='2'/>
      </ItemsPanelTemplate>
    </ComboBox.ItemsPanel>
    <ComboBoxItem>First</ComboBoxItem>
    <ComboBoxItem>Second</ComboBoxItem>
    <ComboBoxItem>Third</ComboBoxItem>
    <ComboBoxItem>Fourth</ComboBoxItem>
  </ComboBox>
  <Rectangle Height="100"></Rectangle>
  <ListBox>
    <ListBox.ItemsPanel>
      <ItemsPanelTemplate>
        <UniformGrid Columns='2'/>
      </ItemsPanelTemplate>
    </ListBox.ItemsPanel>
    <ListBoxItem>First</ListBoxItem>
    <ListBoxItem>Second</ListBoxItem>
    <ListBoxItem>Third</ListBoxItem>
    <ListBoxItem>Fourth</ListBoxItem>
  </ListBox>
  <ListBox>
    <ListBoxItem>First</ListBoxItem>
    <ListBoxItem>Second</ListBoxItem>
    <ListBoxItem>Third</ListBoxItem>
  </ListBox>
</StackPanel>
```



Listy c.d.

- Kontrolka **ListView** dziedziczy z klasy **ListBox**
- Tryb widoku kontrolki **ListView** można ustawić wykorzystując właściwość **View**
  - jednym z dostępnych widoków jest **GridView**, który pozwala na wyświetlenie kolekcji danych jako tabeli z możliwością dostosowania wyglądu wierszy i kolumn

```
<StackPanel>
  <ListView>
    <ListViewItem>First</ListViewItem>
    <ListViewItem>Second</ListViewItem>
    <ListViewItem>Third</ListViewItem>
    <ListViewItem>Fourth</ListViewItem>
    <ListView.View>
      <GridView>
        <GridViewColumn
            Header="My Header"
            Width="200"/>
      </GridView>
    </ListView.View>
  </ListView>
</StackPanel>
```



Listy c.d.

- Kontrolka **TreeView** daje możliwość wyświetlenia informacji w hierarchicznej postaci z użyciem węzłów, które użytkownik może zwijać i rozwijać
- **TreeView** zawiera hierarchię kontrolek **TreeViewItem**
  - **TreeViewItem** dziedziczy z **HeaderedItemsControl** która ma nagłówek (**Header**) i kolekcję pozycji (**Items**)


```
<TreeView>
  <TreeViewItem Header="1"
                IsExpanded="True">
    <TreeViewItem>
      <TreeViewItem.Header>
        <DockPanel>
          <CheckBox/>
          <TextBlock>11</TextBlock>
        </DockPanel>
      </TreeViewItem.Header>
    </TreeViewItem>
    <TreeViewItem Header="12">
      <TreeViewItem Header="121"/>
      <TreeViewItem Header="122"/>
    </TreeViewItem>
  </TreeViewItem>
</TreeView>
```

```
<Window x:Class="WpfApplication1.Ctrl_TemplateList"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:sys="clr-namespace:System;assembly=mscorlib"
   Title="Ctrl_TemplateList" Height="300" Width="300">
  <Grid>
    <ListBox>
      <ListBox.ItemContainerStyle>
        <Style TargetType='{x:Type ListBoxItem}'>
          <Setter Property='Template'>
            <Setter.Value>
              <ControlTemplate TargetType='{x:Type ListBoxItem}'>
                <RadioButton
                 IsChecked='{Binding
                 Path=IsSelected,RelativeSource={RelativeSource TemplatedParent}}'
                 Content='{TemplateBinding Property=Content}' />
              </ControlTemplate>
            </Setter.Value>
          </Setter>
        </Style>
      </ListBox.ItemContainerStyle>
      <sys:String>First</sys:String>
      <sys:String>Second</sys:String>
      <sys:String>Third</sys:String>
      <sys:String>Fourth</sys:String>
    </ListBox>
  </Grid>
</Window>
```

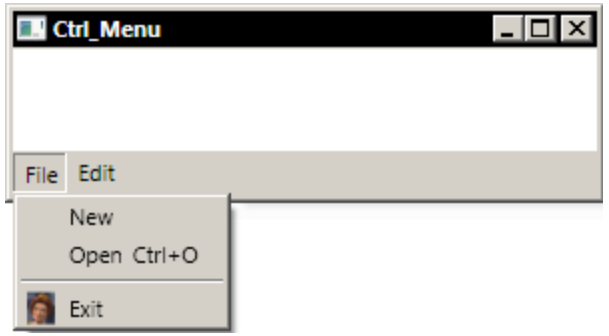
Menu i toolbar

- Menu jest z logicznego punktu odpowiednikiem kontrolki **TreeView** wykorzystującym specjalny wzorzec
  - **Menu** i **TreeView** dziedziczą z klasy **ItemsControl**
  - **TreeViewItem** i **MenuItem** dziedziczą z klasy **HeaderedItemsControl**
- Podstawową funkcjonalnością menu i toolbarów jest uruchamianie przez użytkownika poleceń
  - różnica polega wyglądzie i sposobie interakcji z użytkownikiem
  - zwykle pozycje w menu mają swoje odpowiedniki w postaci przycisków na toolbarach

Menu

- Menu (obiekt klasy **Menu** lub **ContextMenu**) zawiera kolekcję obiektów **MenuItem**
- W aplikacjach WPF menu może znajdować się w dowolnym miejscu okna

```
<Menu DockPanel.Dock="Bottom">
  <MenuItem Header='_File'>
    <MenuItem Header='_New'/>
    <MenuItem Header='_Open' InputGestureText="Ctrl+O" />
    <Separator/>
    <MenuItem Header='E_xit' Click="ExitMenuItem_Click">
      <MenuItem.Icon>
        <Image Source="Images/Fiona_67x77.gif" Width="16" Height="16"></Image>
      </MenuItem.Icon>
    </MenuItem>
  </MenuItem>
  <MenuItem Header='_Edit'>
    <MenuItem Header='_Cut' />
    <MenuItem Header='C_opy' />
    <MenuItem Header='_Paste' />
  </MenuItem>
</Menu>
```



Toolbar

- Toolbar obsługuje tylko jeden poziom zagnieżdżenia
- **ToolBarTray** - specjalna kontrolka hostująca
- Do toolbaru można dodawać dowolne elementy

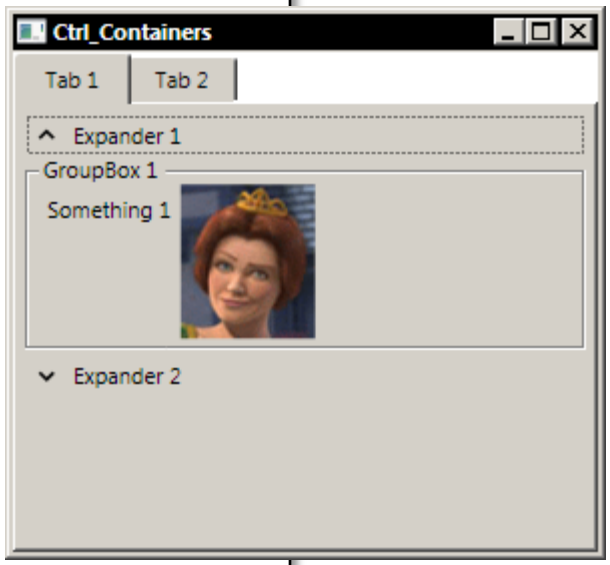
```
<DockPanel>
  <ToolBarTray DockPanel.Dock='Top'>
    <ToolBar>
      <Button>Prev</Button>
      <Button>Next</Button>
    </ToolBar>
    <ToolBar>
      <TextBox Name="AddressTextBox" Width='200' Text="http://www.google.com" />
      <Button Width='23' Click="GoButton_Click">Go</Button>
    </ToolBar>
  </ToolBarTray>
  <WebBrowser Name="ContentWebBrowser"></WebBrowser>
</DockPanel>
```

Kontenery

- **TabControl** udostępnia tradycyjny interfejs użytkownika z zakładkami
  - dziedziczy z klasy **Selector** (i pośrednio z **ItemsControl**)
- **Expander** daje możliwości znane z eksploratora plików w Windows
- **GroupBox** to prosty kontener umożliwiający graficzną separację wybranych elementów
- **TabItem**, **Expander** i **GroupBox** dziedziczą z klasy **HeaderedContentControl**
  - obiekty **HeaderedContentControl** mogą zawierać jeden nagłówek i jedną zawartość
  - tą jedną zawartością może być inny kontener, np. **StackPanel** or **Grid**, dzięki czemu można dodać wiele innych elementów

Przykład kontenerów

```
<TabControl>
  <TabItem Header='Tab 1'>
    <StackPanel>
      <Expander Header='Expander 1' IsExpanded='True'>
        <GroupBox Header='GroupBox 1'>
          <StackPanel Orientation="Horizontal">
            <Label>Something 1</Label>
            <Image Source="Images/Fiona_67x77.gif"></Image>
          </StackPanel>
        </GroupBox>
      </Expander>
      <Expander Header='Expander 2'>
        <StackPanel>
          <GroupBox Header='GroupBox 2'>
            <Label>Something 2</Label>
          </GroupBox>
          <GroupBox Header="GroupBox 3">
            <Label>Something 3</Label>
          </GroupBox>
        </StackPanel>
      </Expander>
    </StackPanel>
  </TabItem>
  <TabItem Header='Tab 2' />
</TabControl>
```

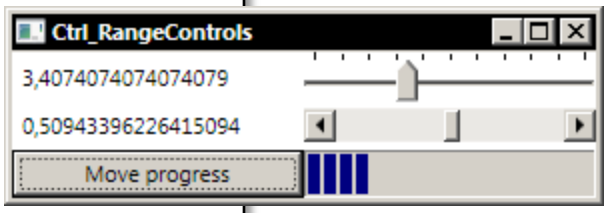


Range Controls

- Slider
- ScrollBar
- ProgressBar

```
private void MySlider_ValueChanged(object sender,
        RoutedPropertyChangedEventArgs<double> e) {
    SliderLabel.Content = MySlider.Value;
}
private void MyScrollBar_ValueChanged(object sender,
        RoutedPropertyChangedEventArgs<double> e) {
    ScrollBarLabel.Content = MyScrollBar.Value;
}
private void ProgressButton_Click(object sender,
        RoutedEventArgs e) {
    MyProgressBar.Value += MyProgressBar.SmallChange * 10;
}
```

```
<UniformGrid Columns="2">
  <Label Name="SliderLabel">0</Label>
  <Slider Name="MySlider"
          ValueChanged="MySlider_ValueChanged"
          TickPlacement="TopLeft" />
  <Label Name="ScrollBarLabel">0</Label>
  <ScrollBar Name="MyScrollBar"
             ValueChanged="MyScrollBar_ValueChanged"
             Orientation="Horizontal" />
  <Button Name="ProgressButton"
          Click="ProgressButton_Click">
    Move progress
  </Button>
  <ProgressBar Name="MyProgressBar"/>
</UniformGrid>
```

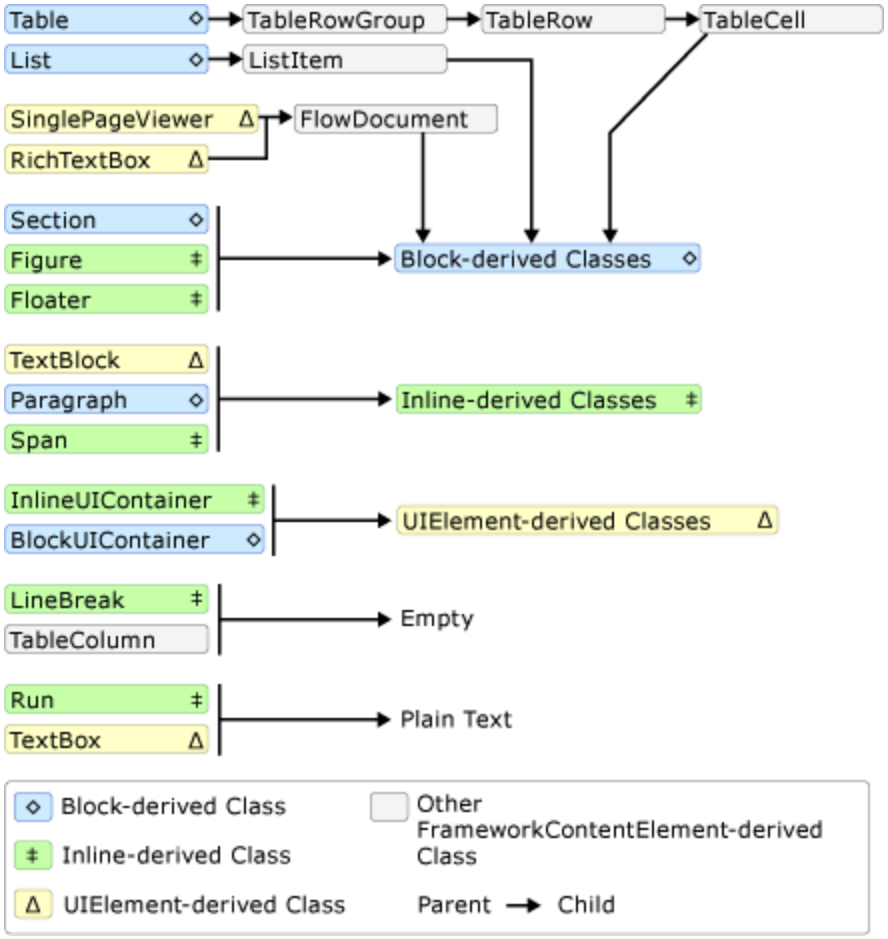


Kontrolki edycyjne

- **PasswordBox** - tradycyjne pole tekstowe do wpisywania hasła
  - hasło jest zabezpieczone (w przeciwieństwie do standardowej kontrolki Windows)
- **TextBox** – tylko prosty tekst bez możliwości formatowania
- **RichTextBox** – złożony tekst, z akapitami, obrazkami, tabelami i in.
  - **RichTextBox** wykorzystuje obiekt klasy **FlowDocument**
- **InkCanvas** – możliwość interakcji piórem (lub myszką) przez użytkownika

Kontrolki edycyjne – reprezentacja tekstu

- Wykorzystywane są dwa rodzaje elementów:
  - *block element* zajmuje ciągły, prostokątny obszar rozpoczynający się od nowej linii
    - np. **Paragraph** lub **Table**
  - *inline element* może obejmować wiele linii
    - np. **Span**, **Run**, **Bold**



Przykłady dokumentów

	Gizmos	Thingamajigs	Doohickies
Blue	*1*	*2*	*3*
Red	*1*	*2*	*3*
Green	*1*	*2*	*3*
Totals	**3**	**6**	**9**

A UIElement element may be embedded directly in flow content by enclosing it in a BlockUIContainer element.

Click me!

The BlockUIContainer element may host no more than one top-level UIElement. However, other UIElements may be nested within the UIElement contained by an BlockUIContainer element. For example, a StackPanel can be used to host multiple UIElement elements within a BlockUIContainer element.

Choose a value:

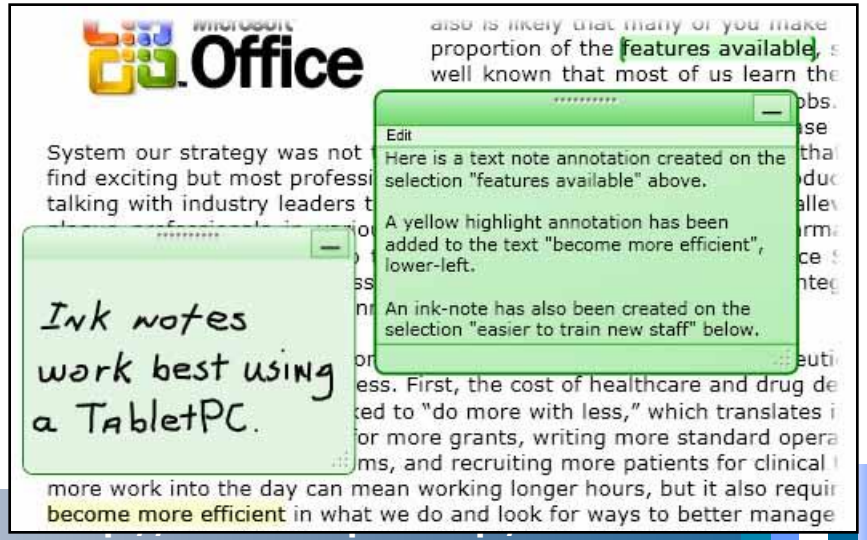
a

Choose a value:

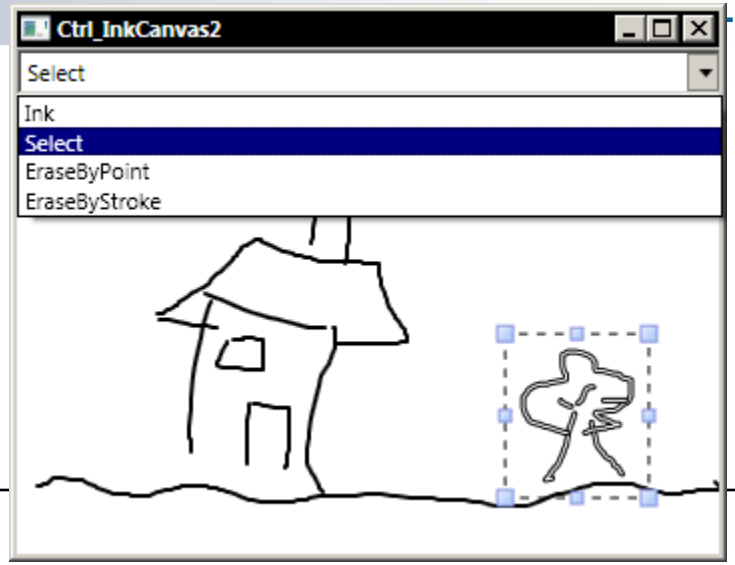
- x
- y
- z

Enter a value:

A text editor embedded in flow content.



Przykład InkCanvas

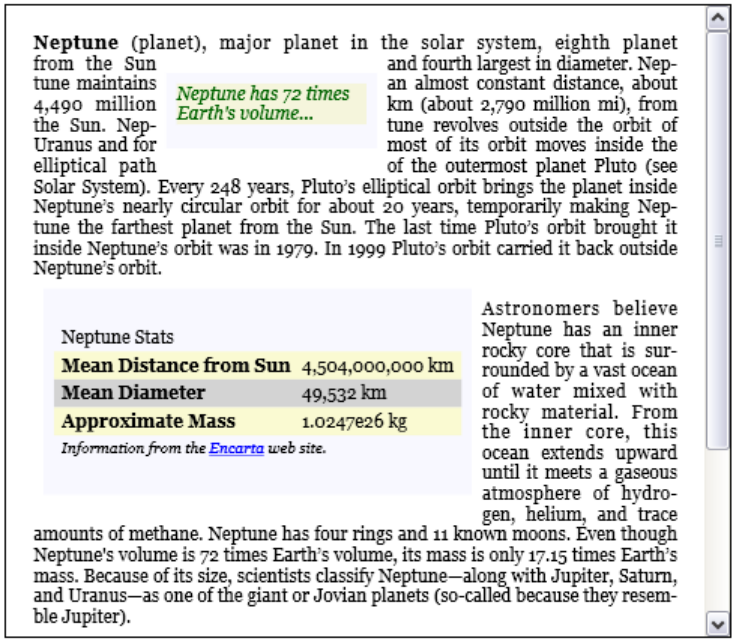



```
<Canvas>
  <Canvas.Resources>
    <!--Define an array containing the InkEditingMode Values.-->
    <x:Array x:Key="MyEditingModes" x:Type="{x:Type InkCanvasEditingMode}">
      <x:Static Member="InkCanvasEditingMode.Ink"/>
      <x:Static Member="InkCanvasEditingMode.Select"/>
      <x:Static Member="InkCanvasEditingMode.EraseByPoint"/>
      <x:Static Member="InkCanvasEditingMode.EraseByStroke"/>
    </x:Array>
  </Canvas.Resources>

  <StackPanel>
    <ComboBox Name="cmbEditingMode"
            ItemsSource="{StaticResource MyEditingModes}" />
    <InkCanvas
            EditingMode="{Binding ElementName=cmbEditingMode, Path=SelectedItem}">
    </InkCanvas>
  </StackPanel>
</Canvas>
```

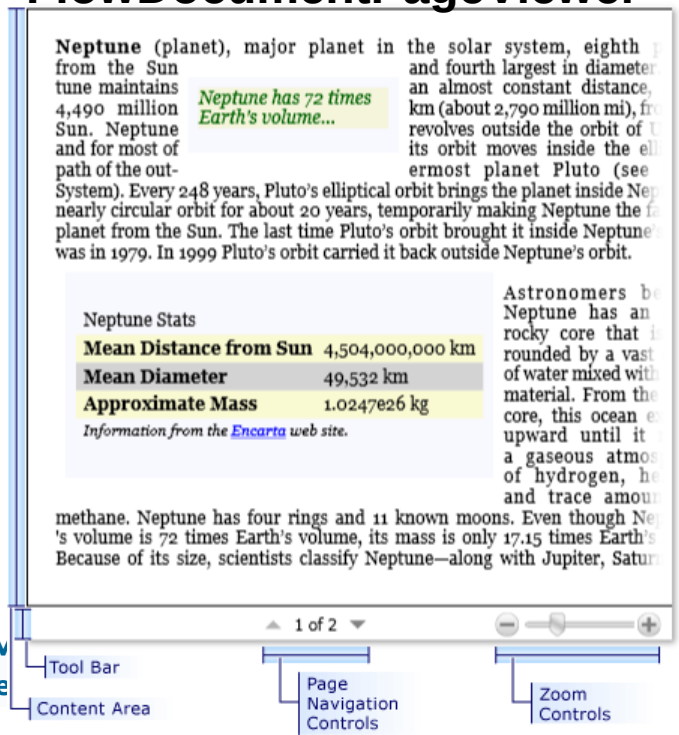
Wyświetlanie dokumentów

- **RichTextBox** – możliwość przeglądania i edycji z przewijaniem zawartości
- **FlowDocumentScrollViewer** – ciągły widok z możliwością przewijania
- **FlowDocumentPageViewer** – widok stronicowany, tylko jedna strona widoczna w danej chwili
- **FlowDocumentReader** – widok do wyboru: ciągły lub stronicowany z wyświetlaniem jednej lub wielu stron naraz
  - użytkownik może przełączać się pomiędzy widokami

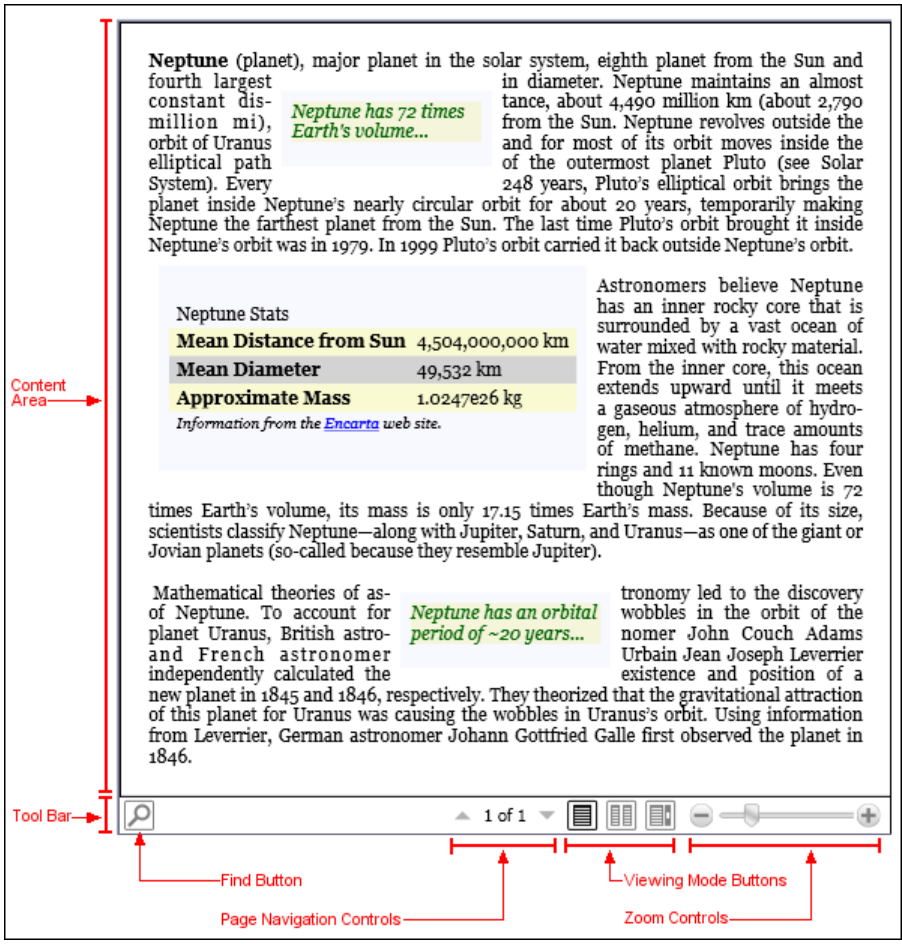
FlowDocumentScrollViewer



FlowDocumentPageViewer



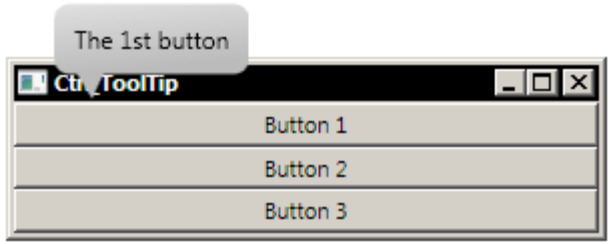
FlowDocumentReader



ToolTip

- W większości przypadków wystarcza wykorzystać właściwość **ToolTip** elementów
- Klasa **ToolTipService** udostępnia właściwości i zdarzenia dające możliwość kontrolki wyglądu i zachowania
- Można zdefiniować własne style i/lub wzorce

```
<Window x:Class="WpfApplication1.Ctrl_ToolTip"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Ctrl_ToolTip" Height="100" Width="300">
  <Window.Resources>
    <Style TargetType="{x:Type ToolTip}">
      [...]
    </Style>
  </Window.Resources>
  <StackPanel>
    <Button ToolTip="The 1st button">Button 1</Button>
    <Button ToolTip="The 2nd button">Button 2</Button>
    <Button ToolTip="The 3rd button">Button 3</Button>
  </StackPanel>
</Window>
```



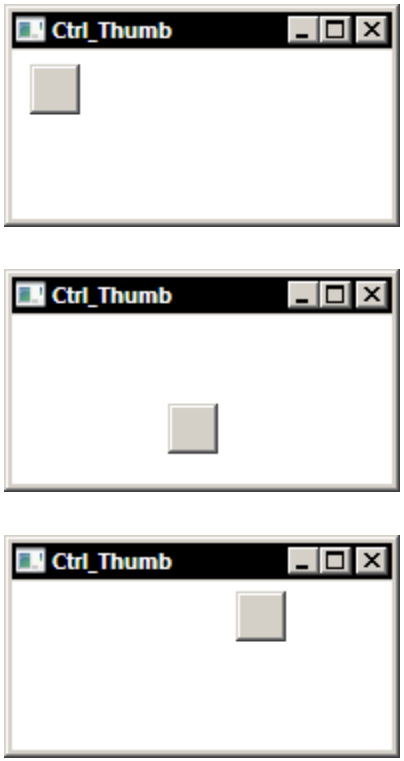
http://thewpfblog.com/?p=61

Thumb

- **Thumb** to kontrolka, która z łatwością może być przenoszone przez użytkownika
  - udostępnia zdarzenia rozpoczęcia, kontynuacji i zakończenia przenoszenia

```
<Canvas>
  <Thumb Canvas.Top = '5' Canvas.Left = '5'
         Width = '25' Height = '25' Name='MyThumb'
         DragStarted='ThumbStart' DragDelta='ThumbMoved' />
</Canvas>
```

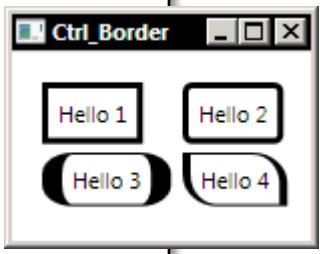
```
void ThumbMoved(object sender, DragDeltaEventArgs e)
{
    Canvas.SetLeft(MyThumb,
       Canvas.GetLeft(MyThumb) + e.HorizontalChange);
    Canvas.SetTop(MyThumb,
       Canvas.GetTop(MyThumb) + e.VerticalChange);
}
```



Border

- **Border** to prostokąt, który może zawierać obiekty potomne
  - większość prostych elementów rysowanych (np. **Rectangle**, **Ellipse**) nie pozwala na dodawanie obiektów potomnych, zatem **Border** może być wykorzystany jako ich rodzic

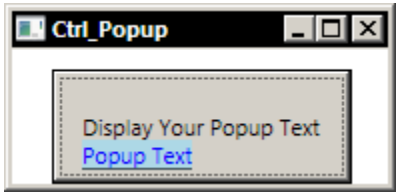
```
<Canvas>
  <Border Canvas.Left='15' Canvas.Top='15' BorderThickness='3'
          CornerRadius='0' BorderBrush='Black' Padding='5'>
    <TextBlock>Hello 1</TextBlock>
  </Border>
  <Border Canvas.Left='85' Canvas.Top='15' BorderThickness='3'
          CornerRadius='3' BorderBrush='Black' Padding='5'>
    <TextBlock>Hello 2</TextBlock>
  </Border>
  <Border Canvas.Left='15' Canvas.Top='50' BorderThickness='10,1,10,1'
          CornerRadius='10' BorderBrush='Black' Padding='5'>
    <TextBlock>Hello 3</TextBlock>
  </Border>
  <Border Canvas.Left='85' Canvas.Top='50' BorderThickness='4,1,4,1'
          CornerRadius='0,15,0,15' BorderBrush='Black' Padding='5'>
    <TextBlock>Hello 4</TextBlock>
  </Border>
</Canvas>
```



Popup

- Kontrolka **Popup** umożliwia wyświetlenie czegoś w oddzielnym oknie jest umieszczone ponad oknem aplikacji ale w odpowiedniej pozycji
- Z tej kontrolki korzystają: **ToolTip**, **ContextMenu**, **ComboBox**, **Expander**

```
<Button HorizontalAlignment="Left" Click="DisplayPopup"
     Width="150" Margin="20,10,0,0">
  <StackPanel>
    <TextBlock>Display Your Popup Text</TextBlock>
    <Popup Name="myPopup">
      <TextBlock Name="myPopupText" Background="LightBlue"  Foreground="Blue">
      Popup Text
      </TextBlock>
    </Popup>
  </StackPanel>
</Button>
```

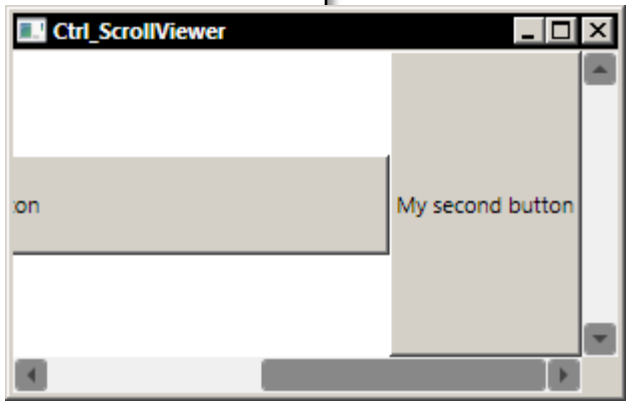



```
private void DisplayPopup(object sender, RoutedEventArgs e) {
    myPopup.IsOpen = !myPopup.IsOpen;
}
```

ScrollViewer

- Kontrolka **ScrollViewer** umożliwia wygodne przewijanie określonej zawartości
  - Zawiera poziomy i pionowy **ScrollBar** oraz kontener dla zawartości
- **ScrollViewer** może zawierać tylko jeden element potomny
  - zwykle jest to **Panel** mogący zawierać wiele **UIElements**

```
<Window x:Class="WpfApplication1.Ctrl_ScrollViewer"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Ctrl_ScrollViewer" Height="300" Width="300">
  <Window.Resources>
    [...]
  </Window.Resources>
  <ScrollViewer HorizontalScrollBarVisibility="Visible">
    <DockPanel>
      <Button Width="400" Height="50">My button</Button>
      <Button>My second button</Button>
    </DockPanel>
  </ScrollViewer>
</Window>
```



http://sachabarber.net/?p=122

Viewbox

- **Viewbox** daje możliwość rozciągnięcia i przeskalowanie jednego obiektu potomnego dla wypełnienia dostępnego obszaru

```
<StackPanel Orientation="Vertical">
  <StackPanel Margin="0,0,0,10" HorizontalAlignment="Center"
              Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Name="btn1" Click="stretchNone">Stretch="None"</Button>
    <Button Name="btn2" Click="stretchFill">Stretch="Fill"</Button>
    <Button Name="btn3" Click="stretchUni">Stretch="Uniform"</Button>
    <Button Name="btn4" Click="stretchUniFill">Stretch="UniformToFill"</Button>
  </StackPanel>
  <StackPanel Margin="0,0,0,10" HorizontalAlignment="Center"
              Orientation="Horizontal" DockPanel.Dock="Top">
    <Button Name="btn5" Click="sdUpOnly">StretchDirection="UpOnly"</Button>
    <Button Name="btn6" Click="sdDownOnly">StretchDirection="DownOnly"</Button>
    <Button Name="btn7" Click="sdBoth">StretchDirection="Both"</Button>
  </StackPanel>
  <TextBlock DockPanel.Dock="Top" Name="txt1" />
  <TextBlock DockPanel.Dock="Top" Name="txt2" />
  <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
    <Viewbox MaxWidth="500" MaxHeight="500" Name="vb1">
      <Image Source="Images/Fiona_67x77.gif"/>
    </Viewbox>
  </StackPanel>
</StackPanel>
```

```
public void stretchNone(object sender, RoutedEventArgs e) {
    vb1.Stretch = System.Windows.Media.Stretch.None;
    txt1.Text = "Stretch is now set to None.";
}
```

WPF

Układ elementów

FrameworkElement

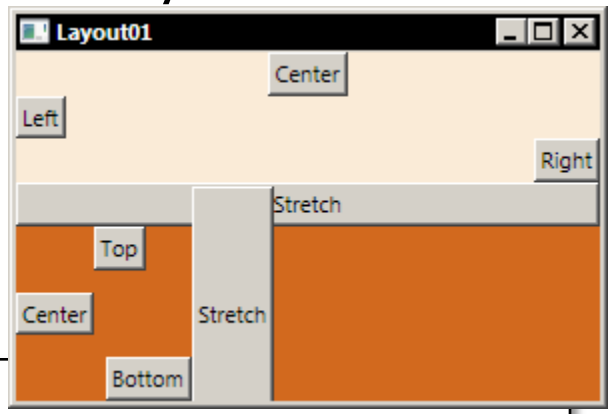
- **FrameworkElement** jest punktem styku pomiędzy klasami reprezentującymi wyświetlane elementy i mechanizmami ich prezentacji zdefiniowanymi jako podstawy WPF
- **FrameworkElement** dziedziczy z **UIElement** dodając następujące możliwości:
  - definicja układu elementów
  - drzewo logiczne elementów
  - zdarzenia towarzyszące działaniu obiektu
  - obsługa wiązania danych i odwołań do dynamicznych zasobów
  - style
  - obsługa animacji

Właściwości klasy FrameworkElement

- **MinWidth**, **MaxWidth**, **MinHeight**, **MaxHeight**
- **Width**, **Height** – zwykle nieustawiane ręcznie, by pozwolić na automatyczne dostosowanie rozmiaru
- **ActualHeight**, **ActualWidth** – aktualny (zwykle obliczony na podstawie układu elementów) rozmiar
- **Margin** – daje możliwość dodania odstępu wewnątrz miejsca przeznaczonego dla elementu
  - można użyć także ujemnych wartości, jeśli element ma wystawać poza przewidziany dla niego obszar
- **HorizontalAlignment**, **VerticalAlignment** – sposób wyrównania pozycji elementu w przewidzianym dla niego obszarze

Obcinanie

- Domyślnie obcinanie nie jest wykonywane
  - WPF używa algorytmu rysującego kontrolki od tyłu
  - Obcinanie jest kosztowne czasowo – dla każdego piksela konieczne jest sprawdzenie, czy powinien być widoczny visible

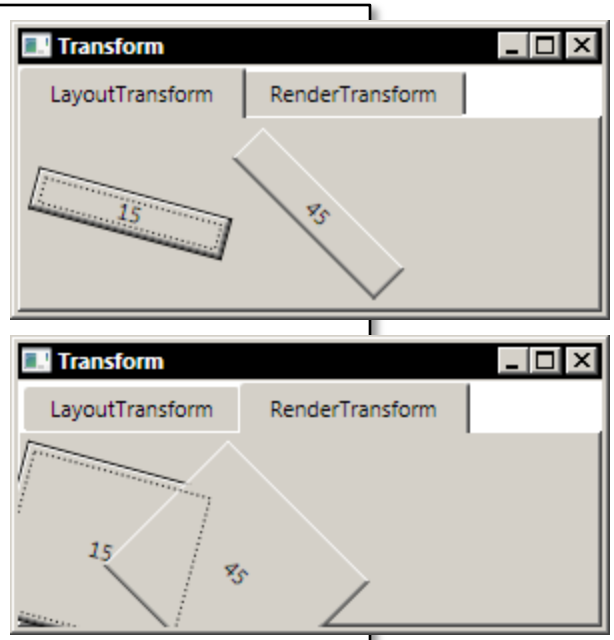



```
<UniformGrid Columns="1">
  <StackPanel Orientation='Vertical' Background="AntiqueWhite">
    <Button HorizontalAlignment='Center'>Center</Button>
    <Button HorizontalAlignment='Left'>Left</Button>
    <Button HorizontalAlignment='Right'>Right</Button>
    <Button HorizontalAlignment='Stretch'>Stretch</Button>
  </StackPanel>
  <StackPanel Orientation='Horizontal' Background="Chocolate">
    <Button VerticalAlignment='Center'>Center</Button>
    <Button VerticalAlignment='Top'>Top</Button>
    <Button VerticalAlignment='Bottom' Margin="-20,0,0,0">Bottom</Button>
    <Button VerticalAlignment='Stretch' Margin="0,-20,0,-20">Stretch</Button>
  </StackPanel>
</UniformGrid>
```

Transformacje

- Dwie właściwości pozwalają na transformacje:
  - **RenderTransform** jest stosowana tuż przed narysowaniem (czyli wpływa wyłącznie na wygląd)
  - **LayoutTransform** jest stosowana przed określeniem miejsca i rozmiaru

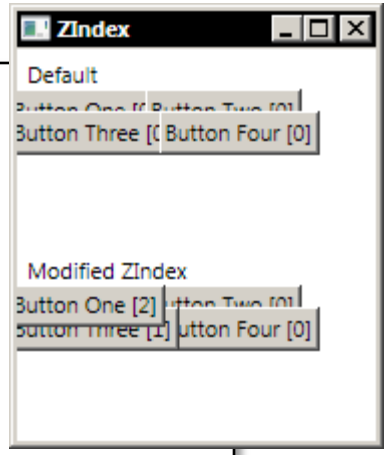
```
<TabControl>
  <TabItem Header="LayoutTransform">
    <StackPanel Orientation="Horizontal">
      <Button Width="100">15
        <Button.LayoutTransform>
          <RotateTransform Angle="15"/>
        </Button.LayoutTransform>
      </Button>
      <Button Width="100">45
        <Button.LayoutTransform>
          <RotateTransform Angle="45"/>
        </Button.LayoutTransform>
      </Button>
    </StackPanel>
  </TabItem>
  <!-- Analogously, but using the RenderTransform -->
</TabControl>
```



Z-Index

- Domyślnie wszystkie kontrolki mają wartość 0 zdefiniowaną dla właściwości **ZIndex**

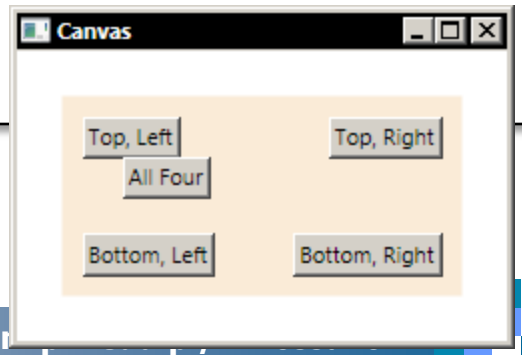
```
<StackPanel>
  <Label>Default</Label>
  <WrapPanel>
    <Button Margin='-5'>Button One [0]</Button>
    <Button Margin='-5'>Button Two [0]</Button>
    <Button Margin='-5'>Button Three [0]</Button>
    <Button Margin='-5'>Button Four [0]</Button>
  </WrapPanel>
  <Rectangle Height="50"/>
  <Label>Modified ZIndex</Label>
  <WrapPanel>
    <Button Panel.ZIndex='2' Margin='-5'>Button One [2]</Button>
    <Button Margin='-5'>Button Two [0]</Button>
    <Button Panel.ZIndex='1' Margin='-5'>Button Three [1]</Button>
    <Button Margin='-5'>Button Four [0]</Button>
  </WrapPanel>
</StackPanel>
```



Canvas

- **Canvas** jest najprostszą kontrolką sterującą układem elementów w WPF
- Udostępnia cztery właściwości: **Top**, **Left**, **Right**, **Bottom**
  - naraz mogą być użyte tylko 2 – jedna pozioma i jedna pionowa
  - pozwalają umiejscowić elementy potomne w pożądanej odległości od odpowiedniego brzegu

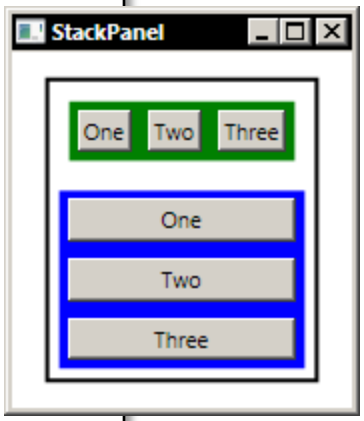
```
<Canvas Width='200' Height='100' Background="AntiqueWhite" >
  <Button Canvas.Right='10' Canvas.Top='10'> Top, Right </Button>
  <Button Canvas.Left='10' Canvas.Top='10'> Top, Left </Button>
  <Button Canvas.Right='10' Canvas.Bottom='10'> Bottom, Right </Button>
  <Button Canvas.Left='10' Canvas.Bottom='10'> Bottom, Left </Button>
  <Button Canvas.Left='30' Canvas.Bottom='30' Canvas.Right='30' Canvas.Top='30'>
    All Four
  </Button>
</Canvas>
```



StackPanel

- **StackPanel** ustawia swoje potomne elementy w kolumnie (ustawienie domyślne) lub w wierszu (wartość **Horizontal** właściwości **Orientation**)
  - każdy z potomnych elementów otrzymuje całą szerokość (albo wysokość) do swojej dyspozycji

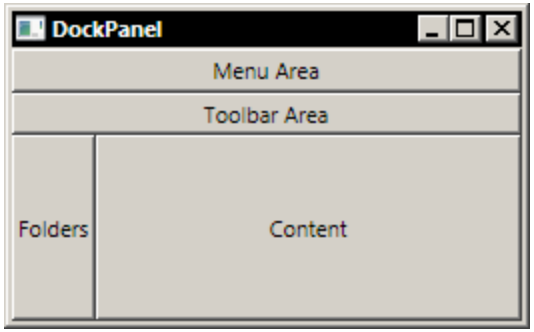
```
<Border BorderBrush='Black' BorderThickness='2'
        HorizontalAlignment='Center' VerticalAlignment='Center'>
  <StackPanel Orientation='Vertical'>
    <StackPanel Margin='10' Background='Green' Orientation='Horizontal'>
      <Button Margin='4'>One</Button>
      <Button Margin='4'>Two</Button>
      <Button Margin='4'>Three</Button>
    </StackPanel>
    <StackPanel Margin='5' Background='Blue' Orientation='Vertical'>
      <Button Margin='4'>One</Button>
      <Button Margin='4'>Two</Button>
      <Button Margin='4'>Three</Button>
    </StackPanel>
  </StackPanel>
</Border>
```



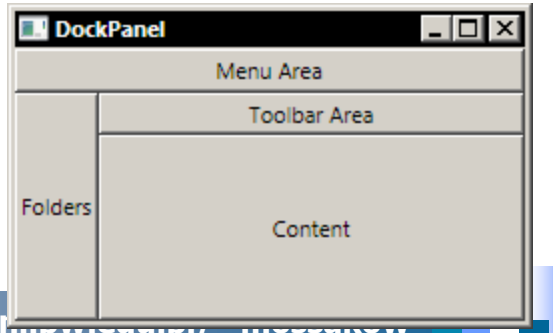
DockPanel

- **DockPanel** pozwala wypełnianie przez elementy potomne obszarów leżących przy swoich brzegach
- Właściwość **LastChildFill** pozwala ostatniemu elementu wypełnić całą pozostałą przestrzeń
- Właściwość **Dock** określa brzeg, do którego kontrolka zostanie doczepiona

```
<DockPanel>
  <Button DockPanel.Dock='Top'>Menu Area</Button>
  <Button DockPanel.Dock='Top'>Toolbar Area</Button>
  <Button DockPanel.Dock='Left'>Folders</Button>
  <Button>Content</Button>
</DockPanel>
```



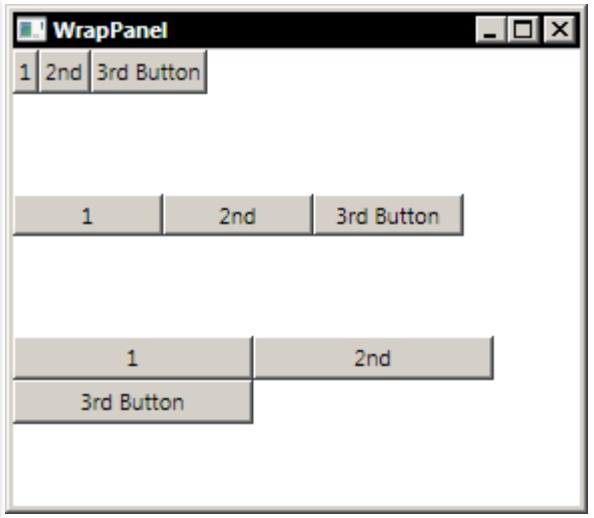

```
<DockPanel>
  <Button DockPanel.Dock='Top'>Menu Area</Button>
  <Button DockPanel.Dock='Left'>Folders</Button>
  <Button DockPanel.Dock='Top'>Toolbar Area</Button>
  <Button>Content</Button>
</DockPanel>
```



WrapPanel

- **WrapPanel** automatycznie rozmieszcza elementy w kolejnych wierszach (domyślnie) lub kolumnach
- Domyślnie **WrapPanel** dostosowuje rozmiary swoich elementów potomnych do ich zawartości
  - można wymusić określoną (jednakową) szerokość lub wysokość elementów za pomocą właściwości **ItemWidth** i **ItemHeight**

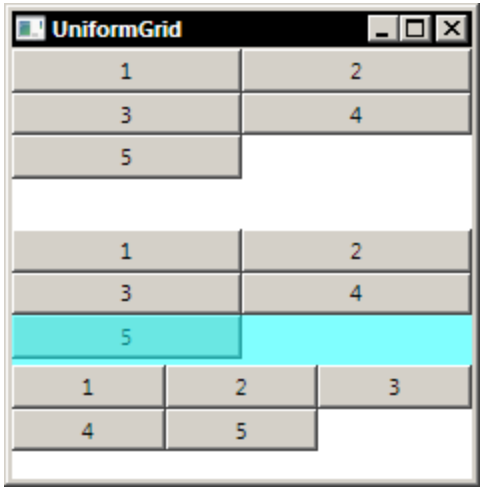
```
<StackPanel>
  <WrapPanel>
    <Button>1</Button>
    <Button>2nd</Button>
    <Button>3rd Button</Button>
  </WrapPanel>
  <Rectangle Height="50"/>
  <WrapPanel ItemWidth="75"> [...] </WrapPanel>
  <Rectangle Height="50"/>
  <WrapPanel ItemWidth="120"> [...] </WrapPanel>
</StackPanel>
```



UniformGrid

- **UniformGrid** to układ tabelaryczny, w którym każda komórka jest takiego samego rozmiaru
- Elementy potomne są przypisywane do kolejnych komórek
- Liczbę kolumn lub wierszy można określić za pomocą właściwości **Columns** i **Rows**

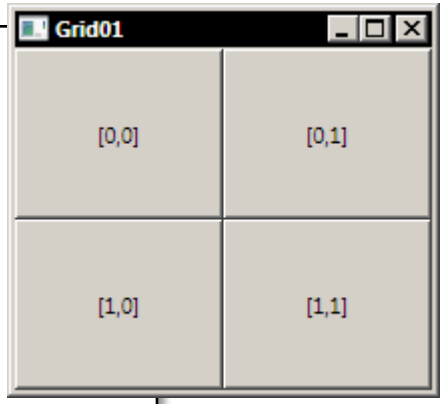
```
<StackPanel>
  <UniformGrid Columns="2">
    <Button>1</Button>
    <Button>2</Button>
    <Button>3</Button>
    <Button>4</Button>
    <Button>5</Button>
  </UniformGrid>
  <Rectangle Height="25"/>
  <UniformGrid Columns="2" Rows="2"> [...] </UniformGrid>
  <Rectangle Height="25" Opacity="0.5" Fill="Cyan" />
  <UniformGrid Rows="2"> [...] </UniformGrid>
</StackPanel>
```



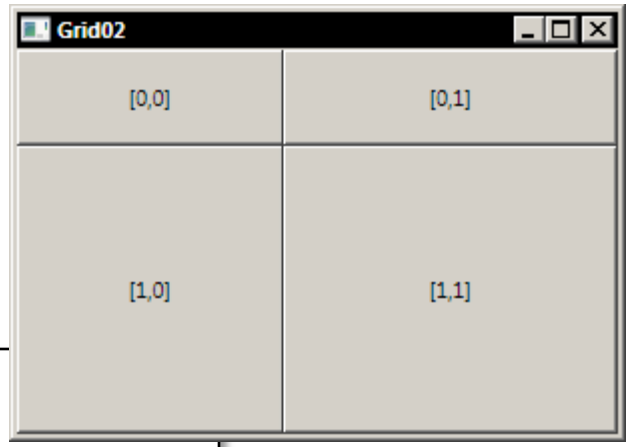
Grid

- **Grid** jest najbardziej skomplikowanym i jednocześnie dającym najwięcej możliwości układem elementów
- Najprostsze jego użycie polega na zdefiniowaniu właściwości **RowDefinitions** i **ColumnDefinitions** i wykorzystaniu dla dodawanych elementów potomnych właściwości **Grid.Row** i **Grid.Column**

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition />
    <RowDefinition />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <Button Grid.Row='0' Grid.Column='0' >[0,0]</Button>
  <Button Grid.Row='1' Grid.Column='1' >[1,1]</Button>
  <Button Grid.Row='0' Grid.Column='1' >[0,1]</Button>
  <Button Grid.Row='1' Grid.Column='0' >[1,0]</Button>
</Grid>
```



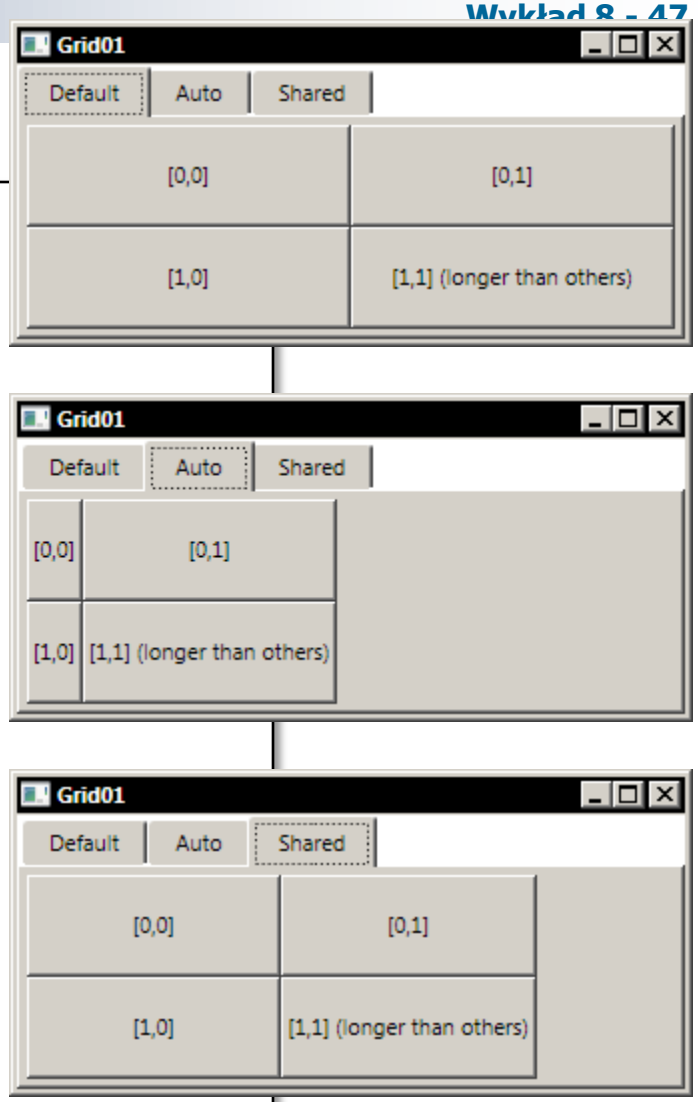
Grid – określanie rozmiaru procentowo




```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="1*"/>
    <RowDefinition Height="3*" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="40*"/>
    <ColumnDefinition Width="50*"/>
  </Grid.ColumnDefinitions>
  <Button Grid.Row='0' Grid.Column='0' >[0,0]</Button>
  <Button Grid.Row='1' Grid.Column='1' >[1,1]</Button>
  <Button Grid.Row='0' Grid.Column='1' >[0,1]</Button>
  <Button Grid.Row='1' Grid.Column='0' >[1,0]</Button>
</Grid>
```

Grid – wspólny rozmiar

```
<Grid>
  [...]
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  [...]
</Grid>
[...]
<Grid>
  [...]
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="Auto"/>
  </Grid.ColumnDefinitions>
  [...]
</Grid>
[...]
<Grid Grid.IsSharedSizeScope="True">
  [...]
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" SharedSizeGroup="ssg01"/>
    <ColumnDefinition Width="Auto" SharedSizeGroup="ssg01"/>
  </Grid.ColumnDefinitions>
  [...]
</Grid>
```



GridSplitter

- **GridSplitter** jest kontrolką pozwalającą na zmianę rozmiaru kolumn lub wierszy kontrolki **Grid**
  - ma cechy normalnej kontrolki (np. obsługę wzorców i stylów)

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="50*" />
    <RowDefinition Height="Auto" />
    <RowDefinition Height="50*" />
  </Grid.RowDefinitions>
  <StackPanel Grid.Row="0"
              Grid.Column="1"
              Background="Tan"/>
  <GridSplitter Grid.Row="1"
          HorizontalAlignment="Stretch"
          VerticalAlignment="Center"
          Background="Black"
          ShowsPreview="True"
          Height="5"

        />
  <StackPanel Grid.Row="2"
              Grid.Column="0"
              Background="Brown"/>
</Grid>
```

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="50*" />

    <RowDefinition Height="50*" />
  </Grid.RowDefinitions>
  <StackPanel Grid.Row="0"
              Grid.Column="1"
              Background="Tan"/>
  <GridSplitter Grid.Row="0"
          HorizontalAlignment="Stretch"
          VerticalAlignment="Bottom"
          Background="Black"
          ShowsPreview="True"
          Height="5"
          ResizeBehavior="CurrentAndNext"
        />
  <StackPanel Grid.Row="1"
              Grid.Column="0"
              Background="Brown"/>
</Grid>
```