Microsoft Students to Business

Universal Windows Platform

XAML

XAML

- Extensible Application Markup Language (XAML)
- Linguagem de marcação para interfaces gráficas
  - Programação declarativa
- Separação entre código e conteúdo (Programação - Lógica)
  - Facilita a colaboração entre designers e desenvolvedores
- Manipulação facilitada para ferramentas
- Arquivos XAML, quando representados como texto, são arquivos XML que geralmente tem a extensão *.xaml*

XAML

- Diferentes categorias de componentes  
  - Controles de layout
  - Controles de IU
  - Primitivas para desenho
  - Manipulação de imagens e mídia

Sintaxe de atributo

- Propriedades de um objeto com frequência podem ser expressas como atributos do elemento de objeto. A sintaxe de atributo é a sintaxe mais simplificada de configuração de propriedades e a mais intuitiva para o desenvolvedores que já usaram alguma linguagem de marcação

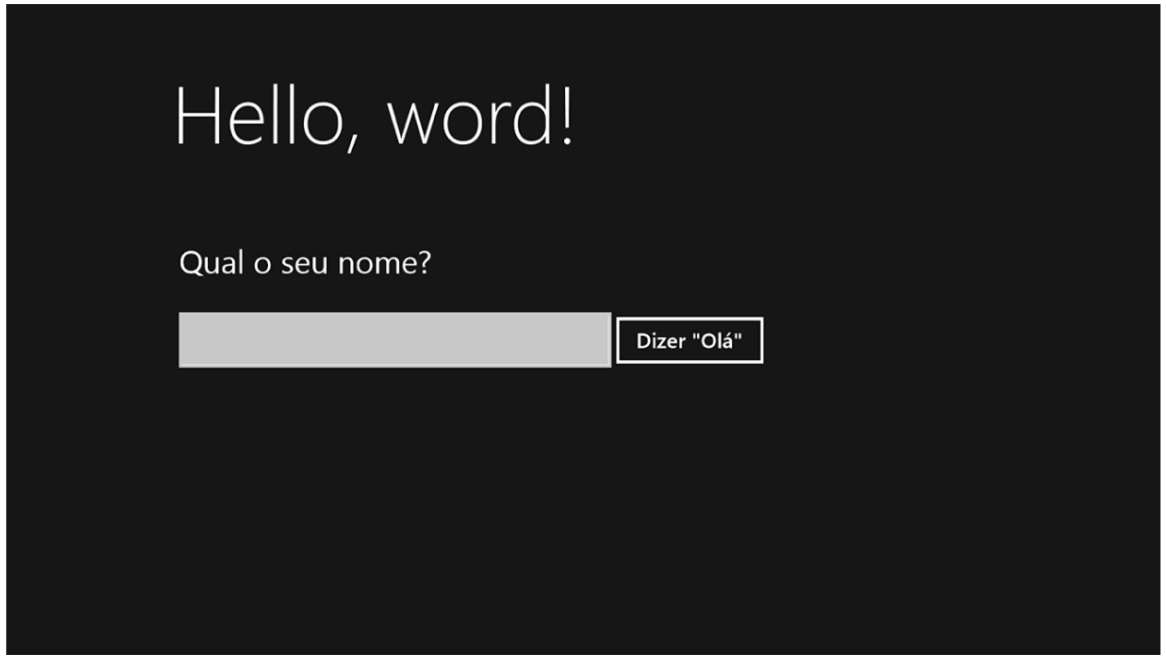
```
<StackPanel>
    <Button Background="Blue"
    Foreground="Red" Content=”Click
    here"/>
</StackPanel>
```

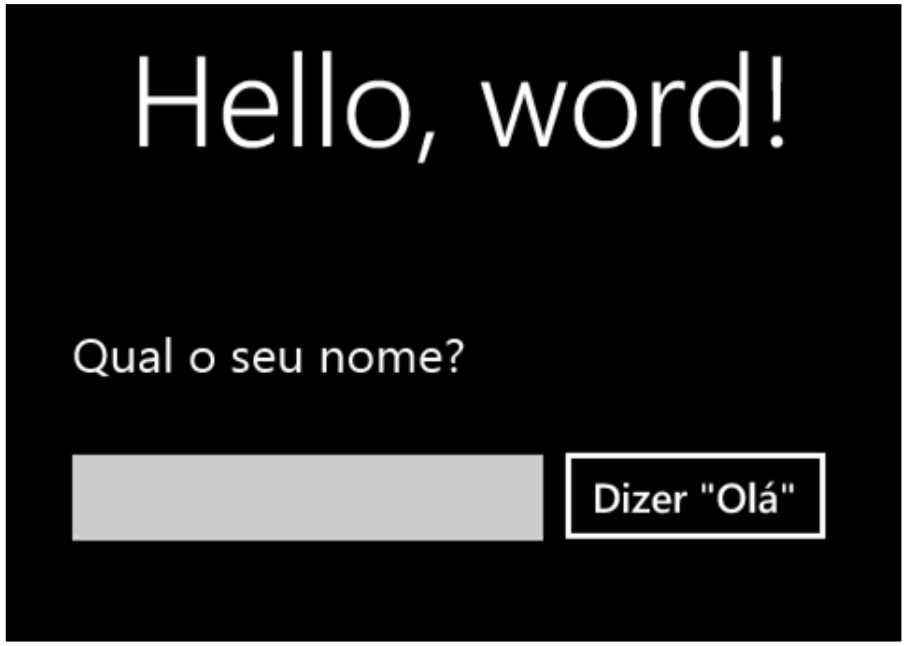
Sintaxe de propriedade de elemento

- Quando o objeto ou as informações necessárias para fornecer o valor da propriedade não podem ser expressos dentro das aspas ou há restrições de caracteres, utilizamos uma construção um pouco mais complexa com o aninhamento de tags.

```
<Button>
    <Button.Background>
     <SolidColorBrush Color="Blue"/>
    </Button.Background>
    <Button.Foreground>
    <SolidColorBrush Color="Red"/>
    </Button.Foreground>
     <Button.Content>
            Click here
    </Button.Content>
</Button>
```

Como funciona?

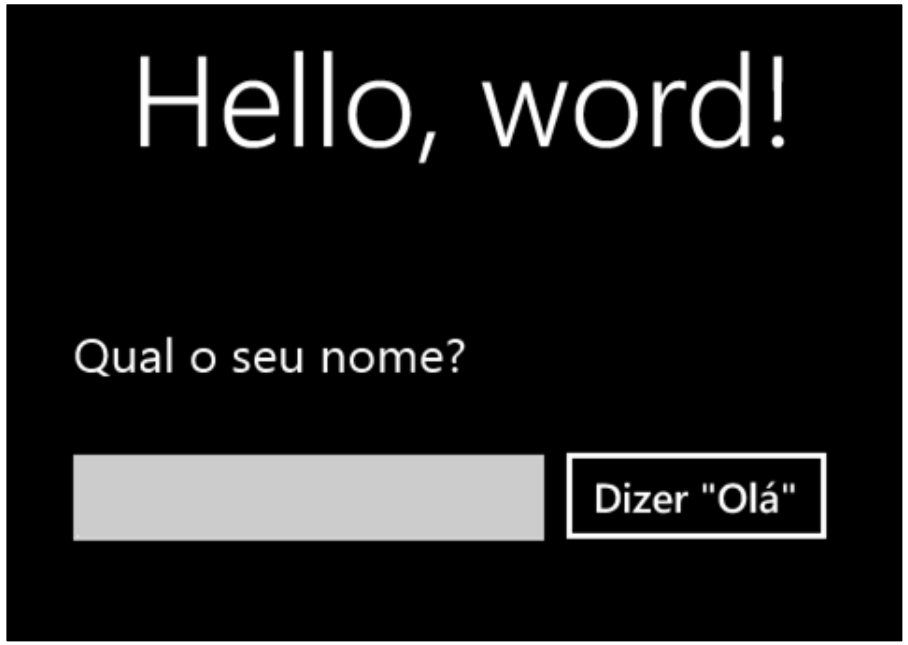





```
<StackPanel Grid.Row="1" Margin="120,30" >
      <TextBlock Text="Qual o seu nome? " />
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
            <TextBox x:Name="nameInput" Width="300"
                     HorizontalAlignment="Left" />
            <Button Content="Dizer &quot;Olá&quot;" />
      </StackPanel>
      <TextBlock x:Name="greetingOutput" />
</StackPanel>
```

```
<StackPanel Grid.Row="1" Margin="30" >
      <TextBlock Text="Qual o seu nome?" />
      <StackPanel Orientation="Horizontal" Margin="0,20,0,20">
            <TextBox x:Name="nameInput" Width="210"
                     HorizontalAlignment="Left" />
            <Button Content="Dizer &quot;Olá&quot; " />
      </StackPanel>
      <TextBlock x:Name="greetingOutput" />
</StackPanel>
```

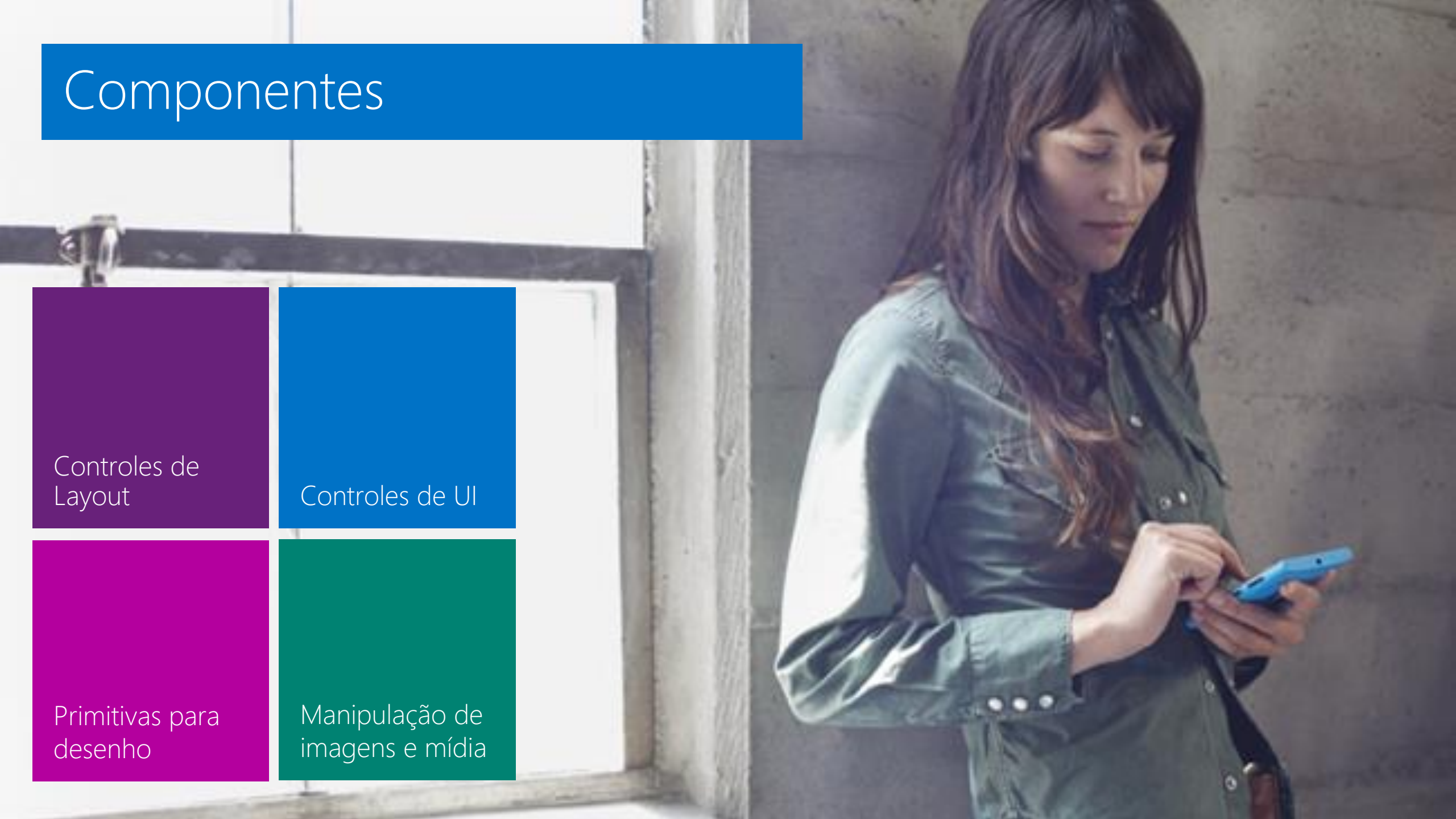
Como funciona?




```
private void Button_Click(object sender,
RoutedEventArgs e)
{
  greetingOutput.Text = "Olá, " +
        nameInput.Text +
        ". Bem vindo ao Windows!";
}
```

Componentes Básicos

Componentes



Controles de Layout

Controles de UI

Primitivas para desenho

Manipulação de imagens e mídia

Componentes

Button

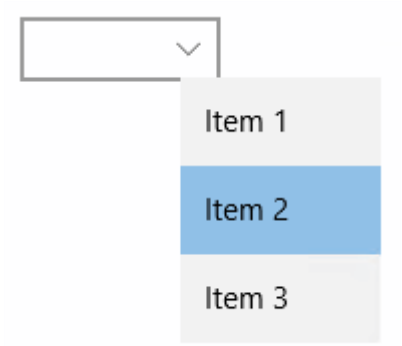
Submit Query

OK

Checkbox

CheckBox CheckBox CheckBox

Combo Box



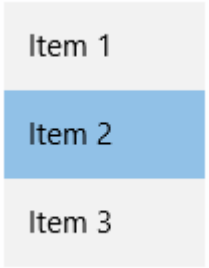
Date Picker

April	17	2015

HyperlinkButton

http://www.buildwindows.com

ListBox



Progress Bar

Radio Button

RadioButton 1

RadioButton 2

RadioButton 3

Slider

42

Time Picker

4	17	PM

Toggle Switch

ToggleSwitch

Off

ToggleSwitch

On

TextBox

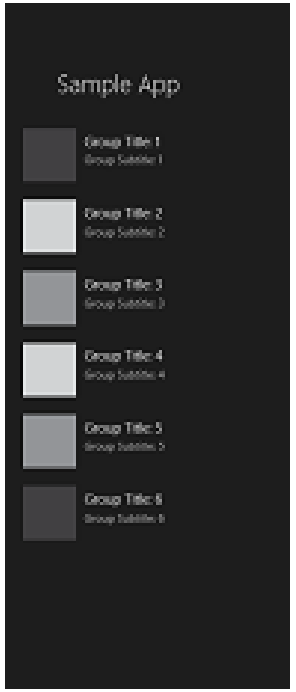
I am a TextBox

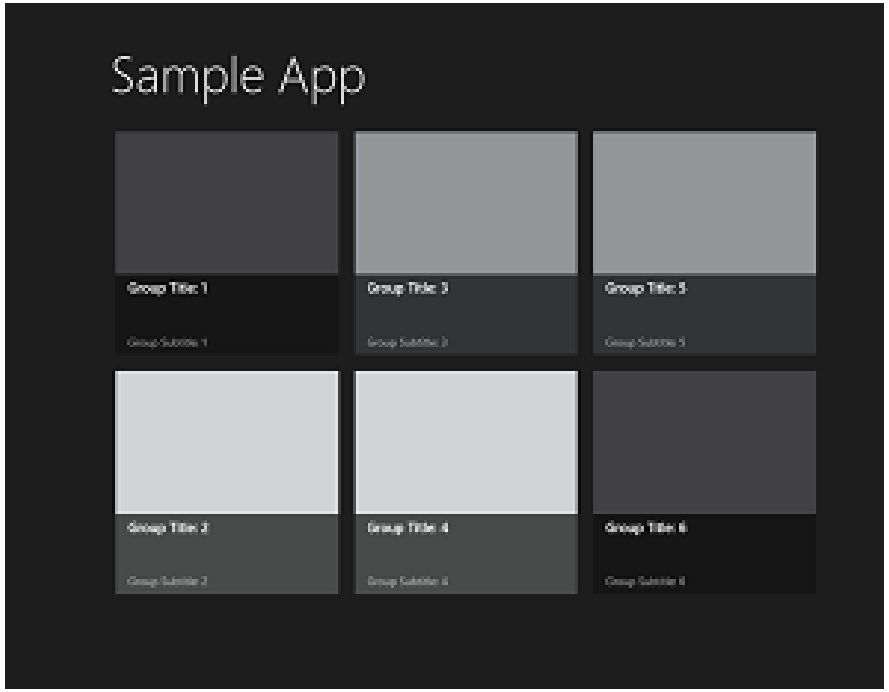
Controles de Layout

- São painéis de layout com o intuito de organizar um grupo de elementos de interface do usuário em seu aplicativo
- O que deve ser considerado principalmente ao escolher um deles é como cada um funciona ao posicionar e dimensionar os seus elementos filhos
- Necessário considerar a sobreposição dos elementos filhos, como vão ficar uns sobre os outros
- Os painéis de layout mais usuais que temos em XAML são
  - Grid
  - StackPanel
  - Canvas
  - RelativePanel

Controles de Layout

- Permitem painéis dentro de painéis
- Podem ser colocados como conteúdo de outros controles
  - Exemplos: FlipView, ListView, GridView, SplitView, etc





Controles de Layout

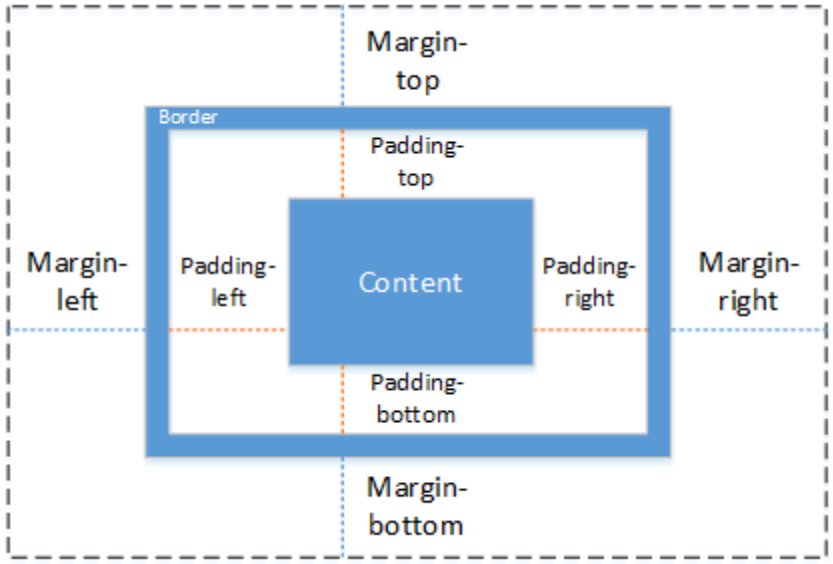
- Canvas: controles filhos provêm seu próprio posicionamento
- Grid: controles filhos são posicionados em linhas e colunas
- StackPanel: controles filhos são “empilhados” verticalmente ou horizontalmente
- RelativePanel: controles filhos são posicionados relativamente a outros controles e ao painél




Controles de Layout

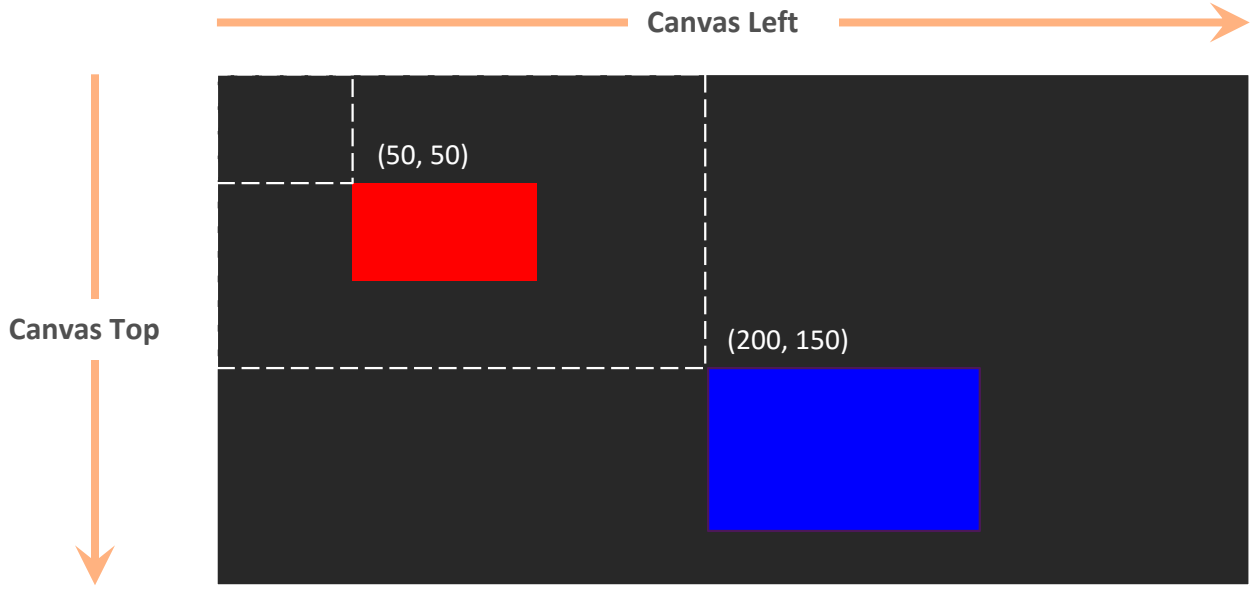
Propriedades básicas para controlar o tamanho e posição do componente

- Width, Height: usados para definir dimensões fixas
- MinWidth, MinHeight, MaxWidth, MinHeight: restringir os tamanhos entre os limites especificados.
- Margin, Padding: define quanto espaço deve ser deixado vazio em torno do controle filho em relação ao pai e dentro do controle filho em torno de seu conteúdo.
- Alignment (Vertical/Horizontal): posição do controle filho em relação ao pai
- Content Alignment (Vertical/Horizontal): alinhamento do conteúdo dentro do controle
- Visibility: mostrar ou esconder o controle



Controles de Layout - Canvas

- Os elementos devem ser explicitamente posicionados (não suporta “fluid UI”)
- Posicionamento relativo às bordas do painel
- Mais utilizado com elementos gráficos

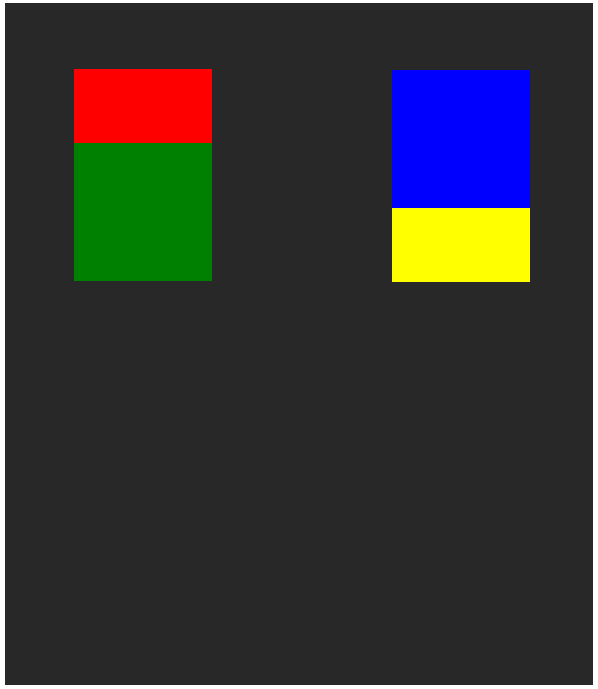



```
<Canvas Background="Transparent">
   <Rectangle Canvas.Left="50" Canvas.Top="80" Fill="red" Width="160" Height="80"/>
   <Rectangle Canvas.Left="250" Canvas.Top="180" Fill="blue" Width="200" Height="120"/>
</Canvas>
```

Controles de Layout - Canvas

- Os elementos são apresentados na ordem em que eles são instanciados
  - Com último vindo na frente dos demais
- Canvas.Zindex - propriedade que ajuda a definir a ordem
  - Valor inteiro arbitrário
  - Menor valor em baixo, maior valor em cima

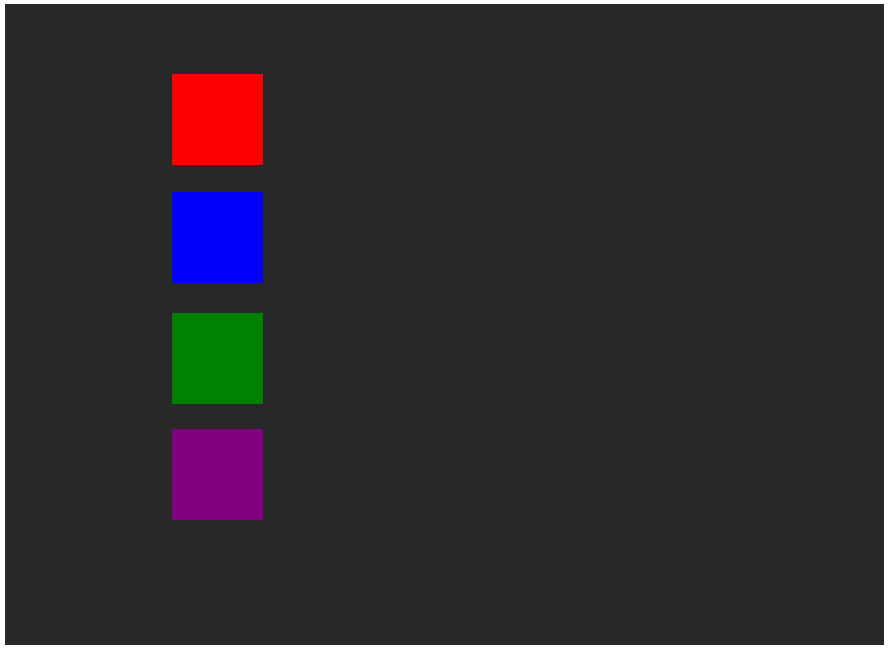
```
<Canvas>
    <Rectangle Width="100" Height="100” Canvas.Left="30" Canvas.Top="30”
    Fill="Red" />
    <Rectangle Width="100" Height="100" Canvas.Left="30" Canvas.Top="45"
    Fill="Green" />
    <Rectangle Width="100" Height="100" Canvas.Left="90" Canvas.Top="30"
    Fill="Blue" Canvas.Zindex="1"  />
    <Rectangle Width="100" Height="100" Canvas.Left="90" Canvas.Top="45"
    Fill="Yellow" Canvas.Zindex="0" />
</Canvas>
```



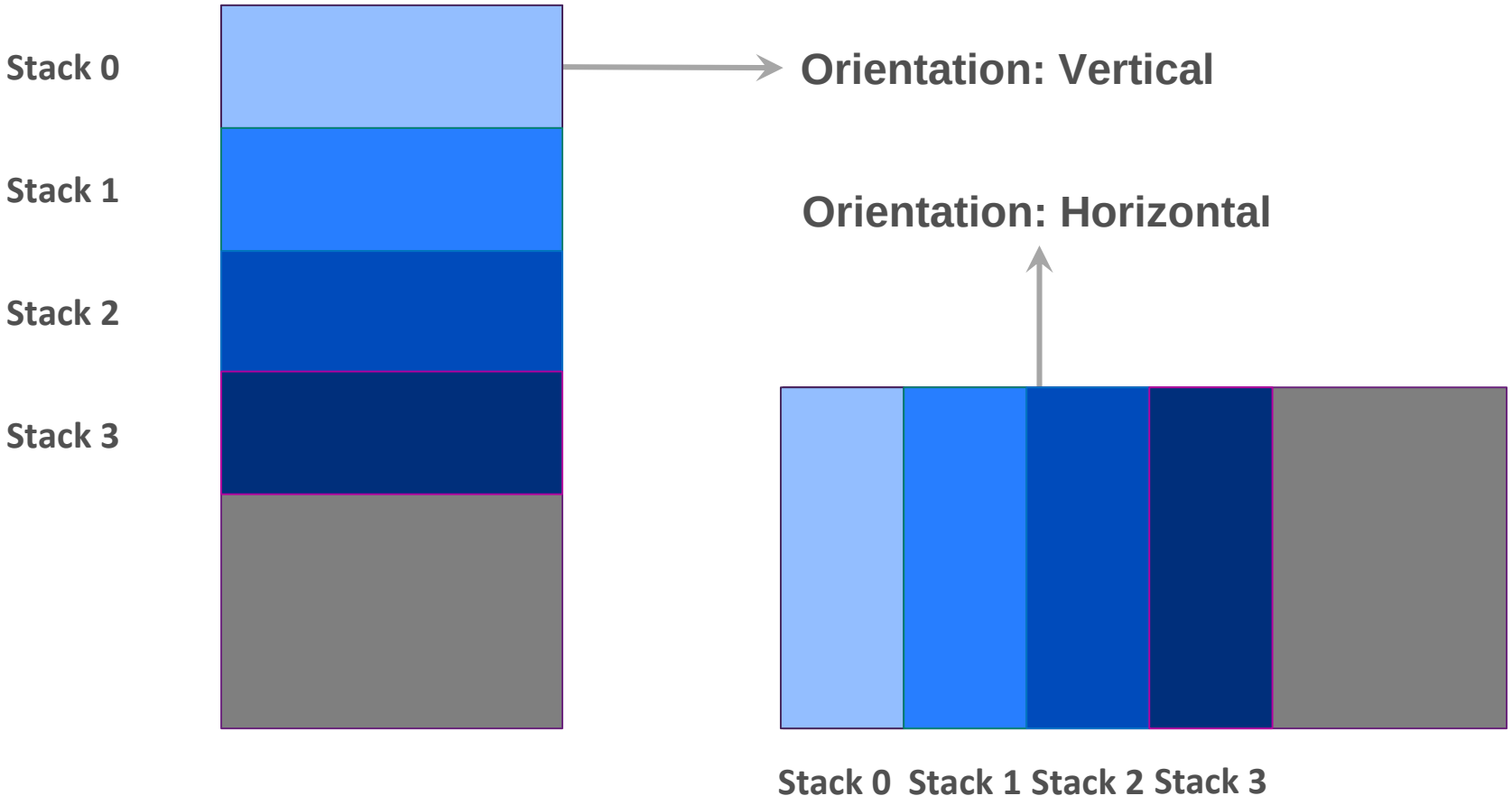
Controles de Layout - StackPanel

- Os elementos podem ser posicionados em painéis verticais, horizontais ou empilhados
- Para evitar, utiliza-se HorizontalAlignment e VerticalAlignment

```
<StackPanel Margin="20">
   <Rectangle Fill="Red" Width="50" Height="50" Margin="5" />
   <Rectangle Fill="Blue" Width="50" Height="50" Margin="5" />
   <Rectangle Fill="Green" Width="50" Height="50" Margin="5" />
   <Rectangle Fill="Purple" Width="50" Height="50" Margin="5" />
</StackPanel>
```



Controles de Layout - StackPanel



Controles de Layout - Grid

- É bastante versátil
- Os elementos podem ser posicionados em linhas e colunas
- Possui uma coleção de definição de linhas e colunas

```
<Grid>
      <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
   <TextBlock Text="Linha 0 Coluna 0" Grid.Row="0" Grid.Column="0"/>
   <Button Content="Linha 1 Coluna 0" Grid.Row="1" Grid.Column="0"/>
   <Button Content="Linha 1 Coluna 1" Grid.Row="1" Grid.Column="1"/>
   <TextBlock Text="Linha 0 Coluna 1" Grid.Row="0" Grid.Column="1"/>
</Grid>
```

Controles de Layout - Grid

- Três definições para altura e largura de linhas e colunas
  - Fixo - especificado em pixel independente
  - Automático - o tamanho necessário
  - Proporcional - tamanhos relativos ao espaço total (suporta multiplicadores ou “pesos”), mudam se o painel é redimensionado

```
<RowDefinition Width="50" />
<RowDefinition Width="Auto" />
<RowDefinition Width="*" />
<RowDefinition Width="2*" />
```

Controles de Layout - Grid

Controles de Layout - Grid

- Elementos podem se expandir por linhas e colunas

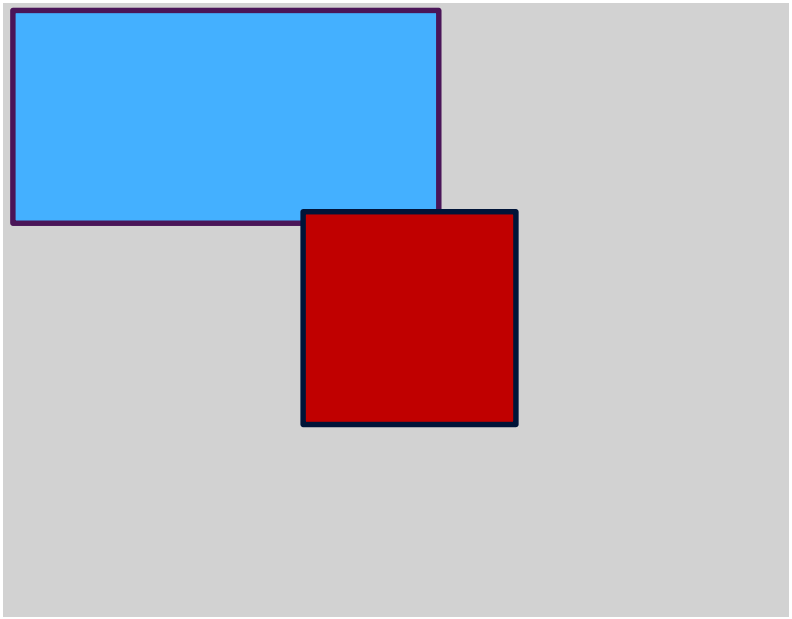
- Propriedades RowSpan e ColumnSpan
  - Especificar o número de linhas ou colunas

```
<Button Grid.Row="0" Grid.Column="0" Grid.RowSpan="2">
2 células
</Button>

<Button Grid.Row="0" Grid.Column="0" Grid.RowSpan="2" Grid.ColumnSpan="2">
4 células
</Button>
```

Controles de Layout - RelativePanel

- Posiciona controles filhos através da declaração de relacionamentos entre eles (simplifica “fluid UI”)
- Um ou dois filhos atuam como elementos âncora relativos ao painel (RelativePanel.AlignHorizontalCenterWithPanel, RelativePanel.AlignVerticalCenterWithPanel)
- Outros filhos são relativos à âncora (RelativePanel.Above, RelativePanel.RightOf, RelativePanel.Below, RelativePanel.LeftOf)

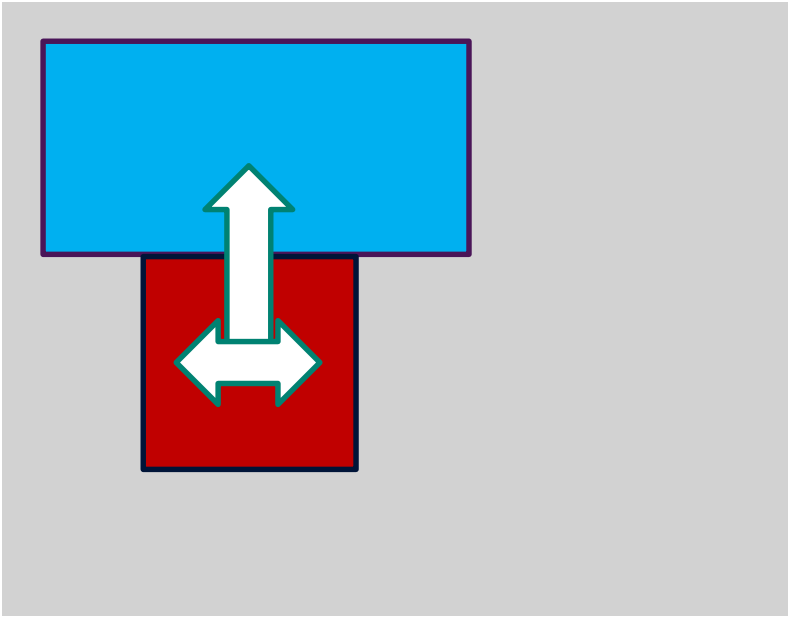
```
<RelativePanel>

    <Rectangle x:Name="RedRect"
        Height="100" Width="100" Fill="Red"
                RelativePanel.AlignHorizontalCenterWithPanel="True"
        RelativePanel.AlignVerticalCenterWithPanel="True" />

    <Rectangle x:Name="BlueRect"
        Height="100" Width="200" Fill="Blue" />

</RelativePanel>
```



Controles de Layout - RelativePanel

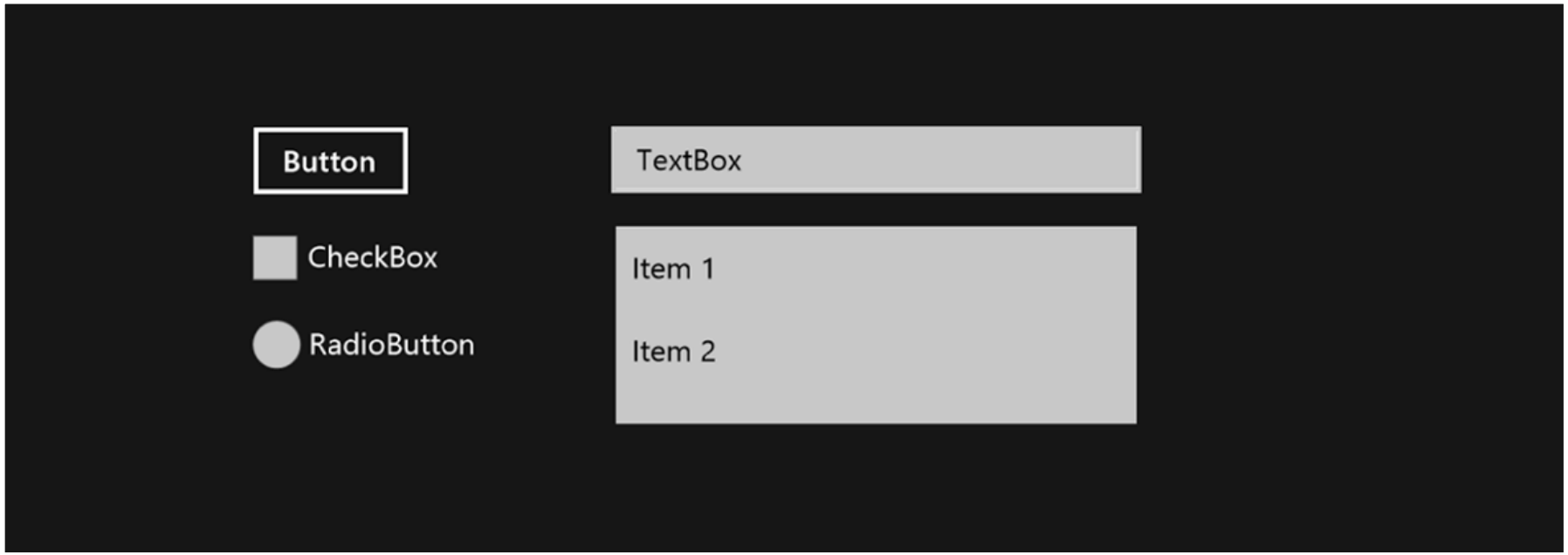
- Posiciona controles filhos através da declaração de relacionamentos entre eles (simplifica “fluid UI”)
- Um ou dois filhos atuam como elementos âncora relativos ao painel (RelativePanel.AlignHorizontalCenterWithPanel, RelativePanel.AlignVerticalCenterWithPanel)
- Outros filhos são relativos à âncora (RelativePanel.Above, RelativePanel.RightOf, RelativePanel.Below, RelativePanel.LeftOf)

```
<RelativePanel>

    <Rectangle x:Name="BlueRect"
        Height="100" Width="100" Fill="Blue" />

    <Rectangle x:Name="RedRect"
        Height="100" Width="100" Fill="Red"
        RelativePanel.Below="BlueRect"
        RelativePanel.AlignHorizontalCenterWith="BlueRect" />

</RelativePanel>
```



Controles de UI

- Controles são objetos individuais ou compostos que possibilitam uma relação iterativa entre si e com o usuário
  - Ex.: Button, CheckBox, ListBox, TextBox, etc



Controles de Texto

- Apresentam uma string de texto
- TextBlock
  - Texto simples somente de leitura
- TextBox
  - Caixa de entrada de texto
  - Suporta linha simples ou múltiplas linhas
- PasswordBox
  - Caixa de entrada de texto mascarada
- RichTextBlock e RichEditBox
  - Texto com formatação rica

Controles de Texto

```
<TextBlock Margin="10" Text="TextBlock" />

<TextBlock Margin="10">
   <Run FontFamily="Arial" FontSize="20">TextBlock</Run>
   <LineBreak />
   <Run FontFamily="Verdana" FontWeight="Bold" FontSize="14">using Inlines</Run>
</TextBlock>

<TextBox Margin="10" HorizontalAlignment="Left" Width="100" Text="Microsoft Innovation Centers are
state of the art technology facilities for collaboration on innovative technology and software
solutions." FontFamily="Arial" TextWrapping="Wrap" />

<RichTextBlock>
     <Paragraph>
        <Bold>Microsoft Innovation Centers (MIC)</Bold> are facilities that provide world
        class resources and support for students, entrepreneurs and startups.
        <LineBreak/>
     </Paragraph>
     <Paragraph><Bold>MICs</Bold> work broadly across their communities, providing services,
solutions and programs that meet the needs of many audiences, including Students, Entrepreneurs,
Governments, and Local Industries.
     </Paragraph>
 </RichTextBlock>
```

Resource Controls

- Source – propriedade que atribui o conteúdo do controle como uma URL absoluta ou relativa
- Ex.: Image, MediaElement, WebView

```
<Image Width="200" Source="Images/myimage.png" />

<MediaElement x:Name="mediaPlayer"
              Source="Videos/video1.mp4"
              Width="400"
              AutoPlay="False"
              AreTransportControlsEnabled="True" />

<WebView x:Name="webView1" Source="http://dev.windows.com" Height="400" Width="800" />
```

Content Controls

- Content – propriedade que atribui o conteúdo do controle (qualquer objeto possível de ser renderizado)
  - Ex: Button, CheckBox, etc
- Esses tipos de controles aceitam somente um único conteúdo filho
- Conteúdo é obtido através de ToString() ou OnRender()
- Importante: mudar o conteúdo de um controle não muda suas características básicas (pense no comportamento de um Button...); para alterar características mais profundas utiliza-se o control template

Content Controls

```
<Button>Conteúdo</Button>

<Button Content="Conteúdo" />
```

```
<Button>
  <Image Source="imagem.jpg“/>
</Button>

<Button>
  <Button.Content>
    <Image Source="imagem.jpg“/>
  </Button.Content>
</Button>
```

Sintaxe do tipo "Property Element"

Controles de Botão

- São controles com ação associada
- Button
  - Dispara eventos de click
- HyperlinkButton
  - Dispara uma navegação entre páginas

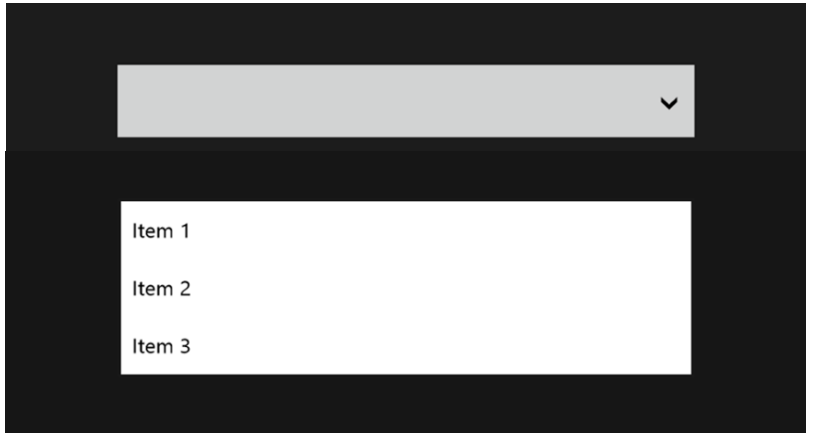
```
<Button>Clique</Button>

<Button Content="Clique" />

<HyperlinkButton Content="Clique aqui para mais informações"
NavigateUri="https://dev.windows.com/en-us" TargetName="_blank" />
```

Item Controls

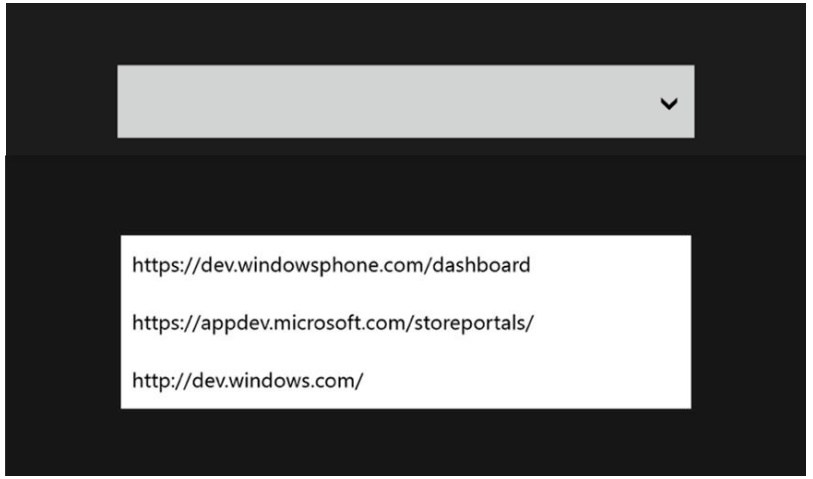
- Items – propriedade que indica a coleção de objetos do tipo de item apropriado para o controle
- ItemsSource – propriedade que indica uma coleção IEnumerable como fonte de dados
- Ex.: ListBox (ListBoxItem), ComboBox (ComboBoxItem), FlipView (FlipViewItem), etc

Item Controls

```
<ComboBox Width="400" Height="50">
    <ComboBox.Items>
        <ComboBoxItem Content="Item 1" />
        <ComboBoxItem Content="Item 2" />
        <ComboBoxItem Content="Item 3" />
    </ComboBox.Items>
</ComboBox>

<ComboBox x:Name="UriBox1" Width="400" Height="50" />
```




```
ObservableCollection<Uri> Uris = new ObservableCollection<Uri>();
Uris.Add(new Uri("https://dev.windowsphone.com/dashboard"));
Uris.Add(new Uri("https://appdev.microsoft.com/storeportals/"));
Uris.Add(new Uri("http://dev.windows.com/"));
UriBox1.ItemsSource = Uris;
```



Elementos Gráficos

- Características
  - 2D e 3D
  - Aceleração de hardware
  - Preferência por desenhos vetoriais
- Categorias de elementos
  - Figuras (Shapes)
  - Pincéis (Brushes)
  - Transformações (Transforms)

Brushes

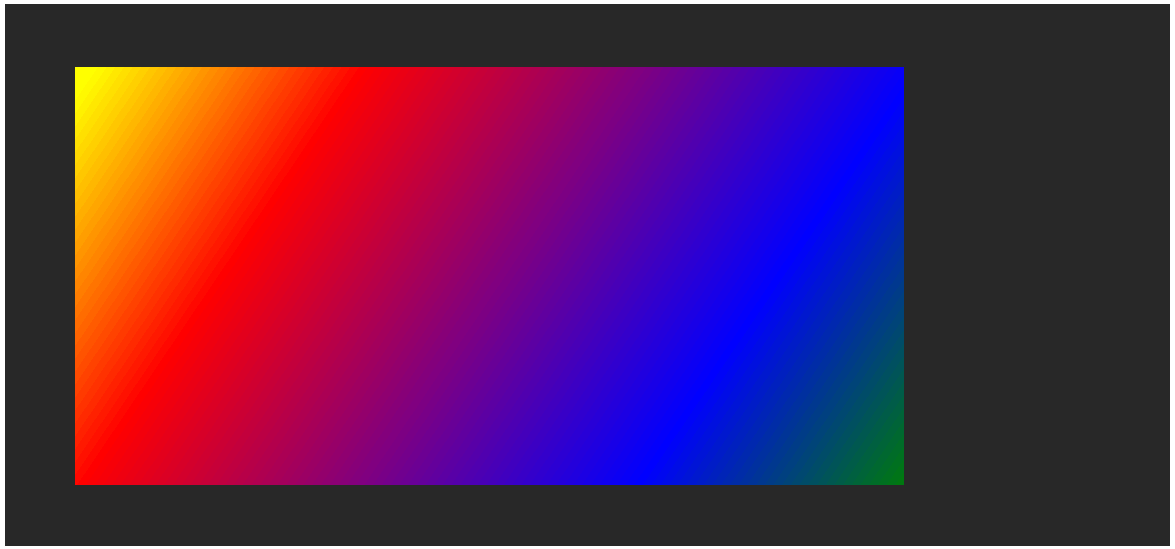
- Responsável pela formatação dos controles
  - SolidColorBrush: diferentes preenchimentos para cores sólidas
  - LinearGradientBrush: preenchimento para gradientes lineares
  - ImageBrush: preenchimento para imagens
  - WebViewBrush: usado para processar o conteúdo de um controle WebView como uma solução para o fato de que o próprio WebView é sempre prestados em cima de outros controles.

SolidColorBrush

```
<StackPanel>
    <Rectangle Width="100" Height="100" Fill="Red"
    />
    <Rectangle Width="100" Height="100"
    Fill="#FFFF0000" />
</StackPanel>
```

```
<Rectangle Width="200" Height="150">
  <Rectangle.Fill>
    <SolidColorBrush Color="Blue"
        Opacity="0.5" />
  </Rectangle.Fill>
</Rectangle>
```

LinearGradientBrush



```
<Rectangle Width="400" Height="200">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0"
          EndPoint="1,1">
      <GradientStop Color="Yellow" Offset="0.0" />
      <GradientStop Color="Red" Offset="0.25" />
      <GradientStop Color="Blue" Offset="0.75" />
      <GradientStop Color=”Green" Offset="1.0" />
    </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

```
<Rectangle Width="400" Height="200">
  <Rectangle.Fill>
    <LinearGradientBrush StartPoint="0,0"
          EndPoint="1,1">
      <GradientStop Color="Yellow” />
      <GradientStop Color="Red" Offset=”1" />
   </LinearGradientBrush>
  </Rectangle.Fill>
</Rectangle>
```

ImageBrush

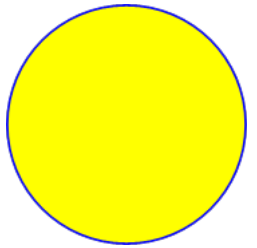
```
<Rectangle Width="400" Height="400">
     <Rectangle.Fill>
          <ImageBrush
               ImageSource="img.png"
               Stretch="None"
               AlignmentX="Left"
               AlignmentY="Center" />
     </Rectangle.Fill>
</Rectangle>
```

Figuras

- Representam elementos primitivos
  - Linhas, elipses, retângulos e polígonos
- Principais classes
  - Shape, Line, Ellipse, Rectangle, Polyline, Polygon, Path
- Derivam de FrameworkElement
  - Possuem o mesmo comportamento dos outros elementos!
- Mais utilizadas dentro de Canvas

Figuras

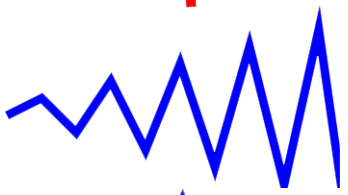
```
<Ellipse Fill="Yellow" Stroke="Blue" Height="100" Width="100" Margin="5"
HorizontalAlignment="Left"></Ellipse>
```



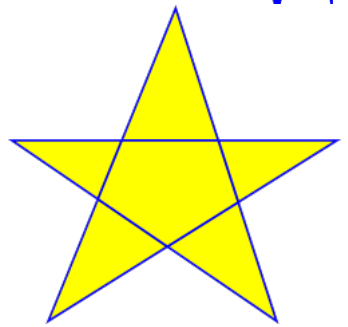
```
<Rectangle Fill="Yellow" Stroke="Blue" Height="50" Width="100" Margin="5"
HorizontalAlignment="Left"></Rectangle>
```

```
<Line Stroke="Blue" X1="0" Y1="0" X2="10" Y2="100" StrokeThickness="5">
</Line>
```

```
<Polyline Stroke="Blue" StrokeThickness="5" Points="10,150 30,140 50,160
70,130 90,170 110,120 130,180 150,110 170,190 190,100 210,240"></Polyline>
```



```
<Polygon Stroke="Blue" StrokeThickness="1" Fill="Yellow" Canvas.Left="10"
Canvas.Top="175" FillRule="Nonzero" Points="15,200 68,70 110,200 0,125
135,125"></Polygon>
```



Tranformações

- Diversas classes que cuidam dessas transformações
  - TranslateTransform
  - RotateTranform
  - ScaleTransform
  - SkewTransform
- Classes adicionais
  - TransformGroup (representa uma transformação composta por outras transformações)
  - MatrixTransform (cria uma matriz para transformação, que consiste em uma transformação linear seguida de uma translação)

Rotate Transform

- Rotaciona o elemento por um ângulo especificado

```
<TextBlock Text="student to business" Foreground="White" FontSize="20">
        <TextBlock.RenderTransform>
                <RotateTransform Angle="90" />
        </TextBlock.RenderTransform>
</TextBlock>
```

Ex.: texto rotacionado num ângulo de 90º no sentido horário.

Scale Transform

- Altera a escala de um objeto 2-D através das coordenadas “ScaleX” e “ScaleY”

```
<TextBlock FontSize="32" FontWeight="Bold" Foreground="White" Text="student to business">
        <TextBlock.RenderTransform>
                <ScaleTransform ScaleX="4" ScaleY="2" />
        </TextBlock.RenderTransform>
</TextBlock>
```

Translate Transform

- Move um objeto de um ponto de coordenadas (x1,y1) para outro ponto de coordenadas (x2,y2)

```
<TextBlock Foreground="White" Text="student to business” FontSize="40" FontWeight="Bold">
    <TextBlock.RenderTransform>
        <TranslateTransform X="9" Y="6" />
    </TextBlock.RenderTransform>
</TextBlock>
```

Ex.: texto movido para o ponto (9,6) a partir de seu ponto inicial (0,0).

Skew Transform

- Coloca um objeto em perspectiva através de ângulos especificados, “AngleX” e “AngleY”. Cria uma ilusão 3-D nos objetos 2-D

```
<TextBlock FontSize="32" FontWeight="Bold" Foreground="White" Text="student to business">
    <TextBlock.RenderTransform>
        <SkewTransform AngleX="20" AngleY="10" />
    </TextBlock.RenderTransform>
</TextBlock>
```

Ex.: texto em perspectiva num ângulo de 20º no eixo X e 10º no ângulo Y.

Eventos e Comandos

Eventos

- Os conceitos de eventos no .NET são semelhantes ao modelo de evento nas linguagens de programação mais usadas

- Define-se um método para tratar o evento quando este mesmo for disparado

Manipuladores de Eventos

- Para objetos que são elementos da interface do usuário e foram declarados em XAML, o código de manipulador de eventos é definido na classe parcial que opera como o code-behind de uma página XAML. Manipuladores de eventos são métodos que você escreve como parte da classe parcial associada ao XAML

```
<Button x:Name="btName” FontSize="40 Click="btName_Click">
        Microsoft Innovation Center
</Button>
```

```
private void btName_Click(object sender, RoutedEventArgs e)
{
            Button bt = (Button)sender;
            if (bt != null)
            {
                 bt.Foreground = new SolidColorBrush(Colors.SpringGreen);
            }
 }
```

Manipuladores de Eventos - Adicionando

- XAML não é a única forma de atribui um manipulador de eventos a um objeto
- Você registra o manipulador fazendo referência ao nome do método manipulador de eventos no lado direito do operador +=

```
btName.Click += btName_Click;
btName.Click += new RoutedEventHandler(btName_Click);
```

- Obs.: Se você está usando código para adicionar manipuladores de eventos a objetos que aparecem na interface do usuário em tempo de execução, uma prática comum é adicionar esses manipuladores em resposta a um evento de ciclo de vida do objeto ou de retorno de chamada – como, por exemplo, Loaded ou OnApplyTemplate, de maneira que os manipuladores de eventos no objeto relevante estejam prontos para eventos iniciados pelo usuário no tempo de execução

Manipuladores de Eventos - Removendo

- Utiliza-se o operador -=

- Casos nos quais pode haver a necessidade de remoção de manipuladores de eventos explicitamente
  - Manipuladores adicionados para eventos estáticos, que não podem sofrer a coleta de lixo de maneira convencional. Exemplos de eventos estáticos na API de Tempo de Execução do Windows são os eventos das classes CompositionTarget e Clipboard.
  - Teste o código quando quiser que o tempo da remoção dos manipuladores seja imediato ou quando quiser trocar antigos/novos manipuladores de eventos para um evento em tempo de execução.
  - A implementação de um acessador remove personalizado.
  - Eventos estáticos personalizados.
  - Manipuladores para navegações de página

```
btName.Click -= btName_Click;
```

Eventos - Routed

- XAML utiliza “routed events”
  - Esse tipo de evento permite que um componente trate um evento originado em outro componente
  - O evento pode ser passado entre múltiplos componentes dentro da hierarquia (chamada de visual tree)

- O evento e seus dados de eventos podem ser tratado em vários objetos ao longo da rota do evento. Se não existem manipuladores vinculados aos elementos, a rota vai até o elemento raíz.

Eventos - Routed

- Objeto RoutedEventArgs
  - Instância compartilhada do evento
  - Propriedades:
    - OriginalSource – retorna referência para o objeto original que lançou o evento (Sender é o objeto no qual o tratador está anexado)
    - Handled – alguns tipos de eventos permitem bloquear o roteamento do evento quando o valor da propriedade é true
- Obs.: Nem todos os eventos podem cancelar uma rota pois eles não terão a propriedade Handled, por exemplo: GotFocus e LostFocus; Alguns eventos sempre usam o cancelamento, por exemplo: Click em Button

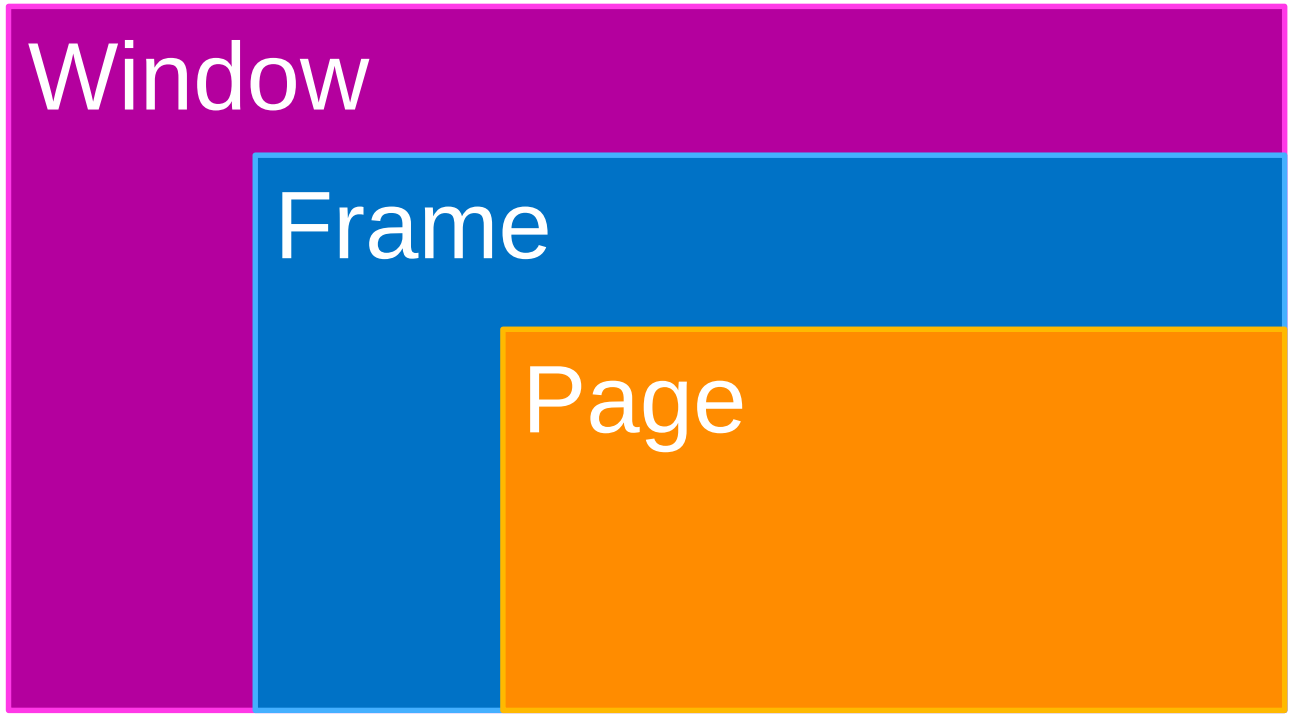
Comandos

- Comandos fornecem uma arquitetura central para o gerenciamento de ações sem a necessidade de um tratador de eventos como ligação
  - Baseados na interface ICommand (método Execute, propriedade CanExecute)
- Comandos podem ser invocados diretamente através de controles ButtonBase e Hyperlink através da propriedade Command
- Um mesmo comando pode ser associado a diversos controles

Navegação

Navegação

- Um window contém tipicamente um frame, dimensionado a 100% da área disponível
- Um frame contém tipicamente um page, dimensionado a 100% da área disponível



Navegação

- Objeto Frame oferece diferentes métodos de navegação
  - Navigate() – navega para uma página específica (indicada pelo tipo do objeto, usualmente via operador *typeof*)
  - GoBack(): navega para o item anterior mais recente
  - GoForward(): navega para o item posterior mais recente

- Templates adicionais oferecem mecanismos alternativos (e úteis) de gerenciamento da navegação (https://github.com/Windows-XAML/Template10)

```
this.Frame.Navigate(typeof(<Pagina>));
```

Navegação

- Método Navigate() permite passar um objeto como parâmetro para a página destino (usar somente tipos baseados em string, char, numéricos, e GUID para suportar serialização no ciclo-de-vida da app)
- Para obter o objeto passado como parâmetro, utiliza-se método de evento *OnNavigatedTo*() que expõe um objeto *NavigationEventArgs* com a propriedade *Parameter*.

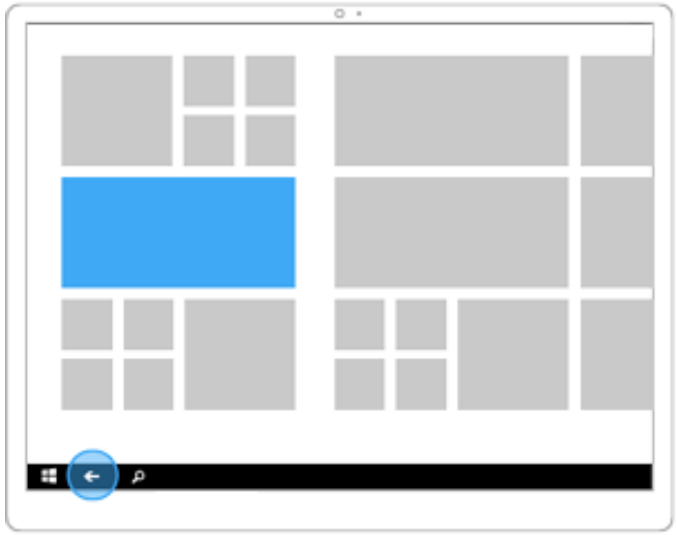
```
this.Frame.Navigate(typeof(MinhaPagina), "objeto");


protected override void OnNavigatedTo(NavigationEventArgs e)
{
    string objeto = e.Parameter as string;
    ...
}
```

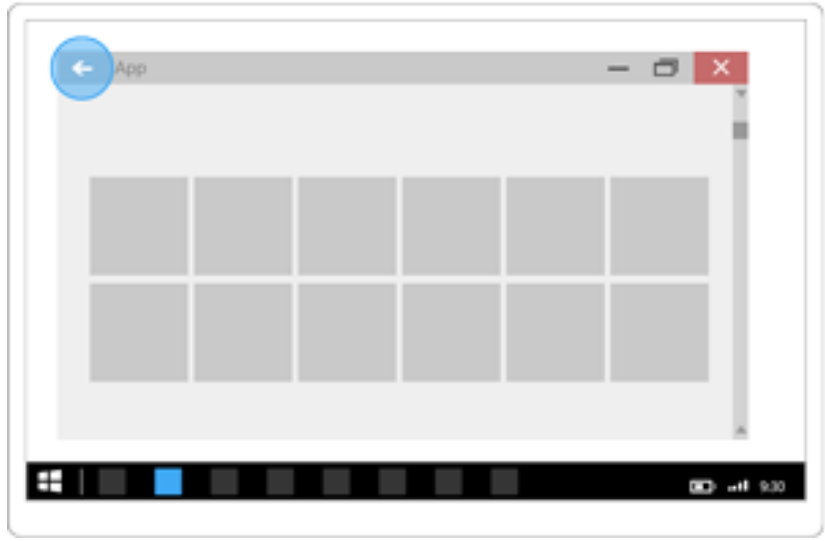
Navegação – Botão Back

- Botão disponível na plataforma UWP em diversas opções
  - Shell
  - Barra de título
  - Dentro da app

Navegação – Botão Back

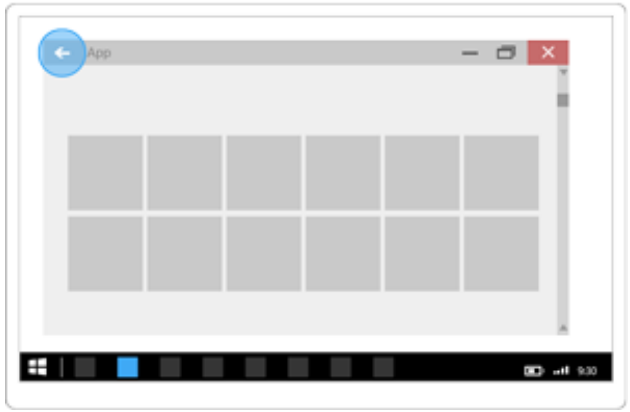
- Fornecido pelo sistema
  - Telefone: sempre disponível; software na “shell” ou hardware; navegação dentro da app e entre apps
  - Tablet: sempre disponível no modo “tablet”; software na “shell”; navegação dentro da app e entre apps
  - PC, Laptop, Tablet: opcional no modo “desktop”; software na barra de título; desabilitado por padrão, app deve solicitar via API; navegação dentro da app





Navegação – Botão Back do Sistema

- Fornecido pelo sistema
  - PC, Laptop, Tablet: opcional no modo “desktop”; software na barra de título; desabilitado por padrão, app deve solicitar via API; navegação dentro da app



```
if (Frame.CanGoBack)
{
    // Setting this visible is ignored on Mobile and when in tablet mode!
    Windows.UI.Core.SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
          AppViewBackButtonVisibility.Visible;
}
```

Navegação – Botão Back Evento

- Evento BackRequested

- Registrar um tratador global ou específico por página

```
Windows.UI.Core.SystemNavigationManager.GetForCurrentView().BackRequested += OnBackRequested;
```

Microsoft®
Students to Business