Universal Windows 10 Apps

Thomas Mutzl

Windows

Agenda

Universal

XAML controls

Adaptive Apps

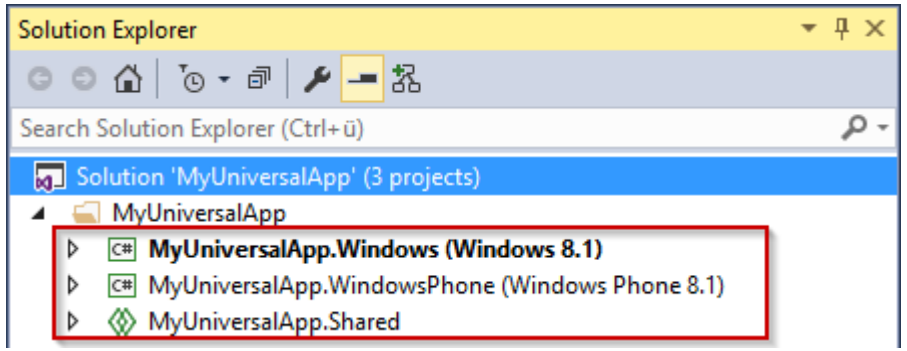
Compiled Binding

Universal

Windows

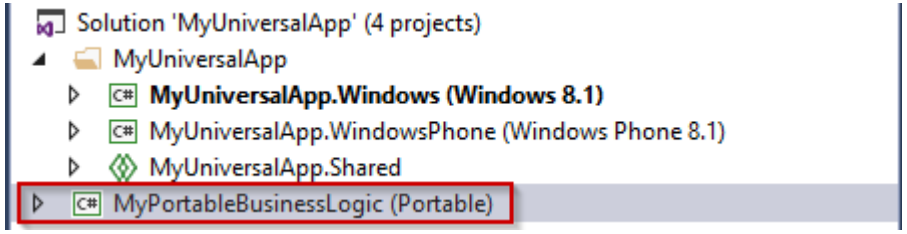
Wie Universal ist Universal?

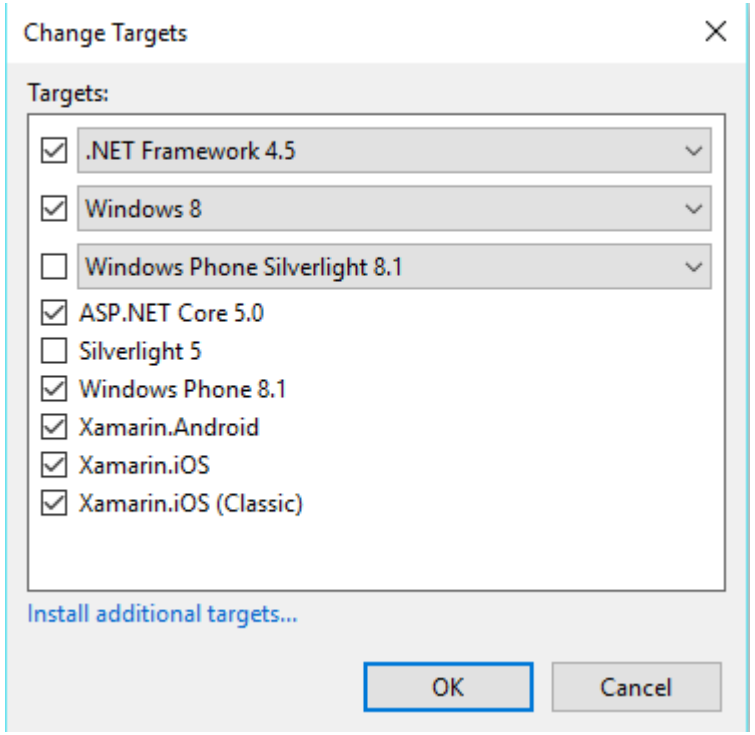
Universal Windows 8.1



Wie Universal ist Universal?

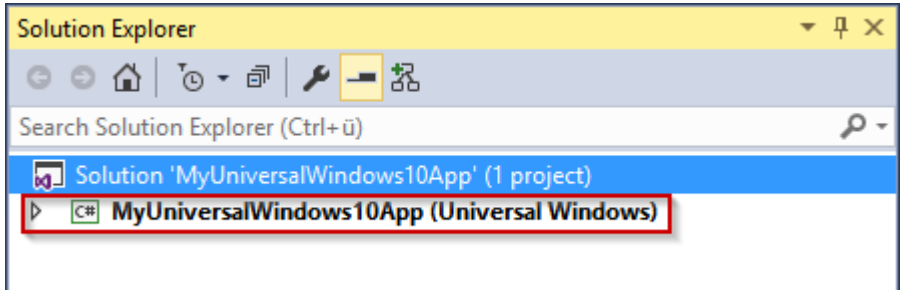
Portable Class Library - PCL

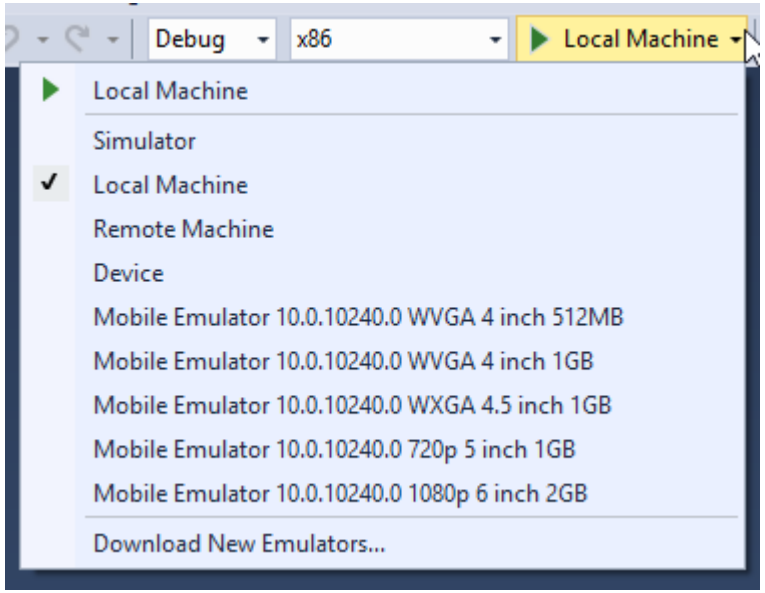




Wie Universal ist Universal?

Universal Windows 10



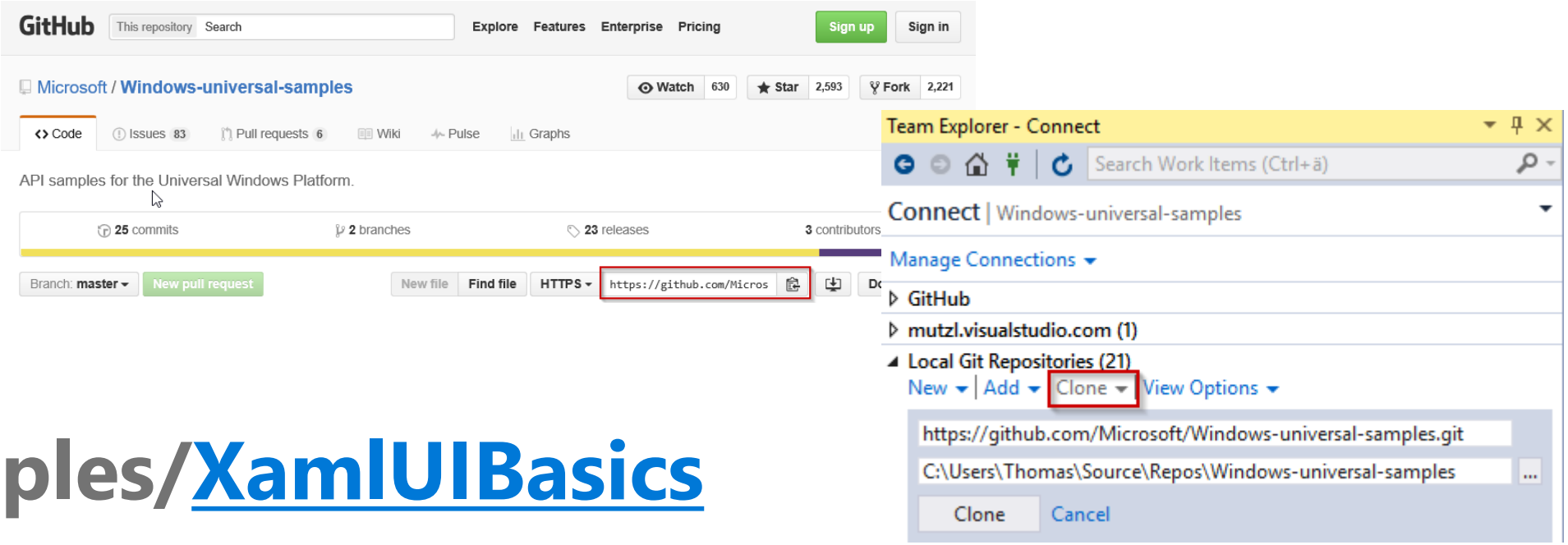


XAML controls

Windows

SDK Samples

https://github.com/Microsoft/Windows-universal-samples.git



/Samples/XamlUIBasics

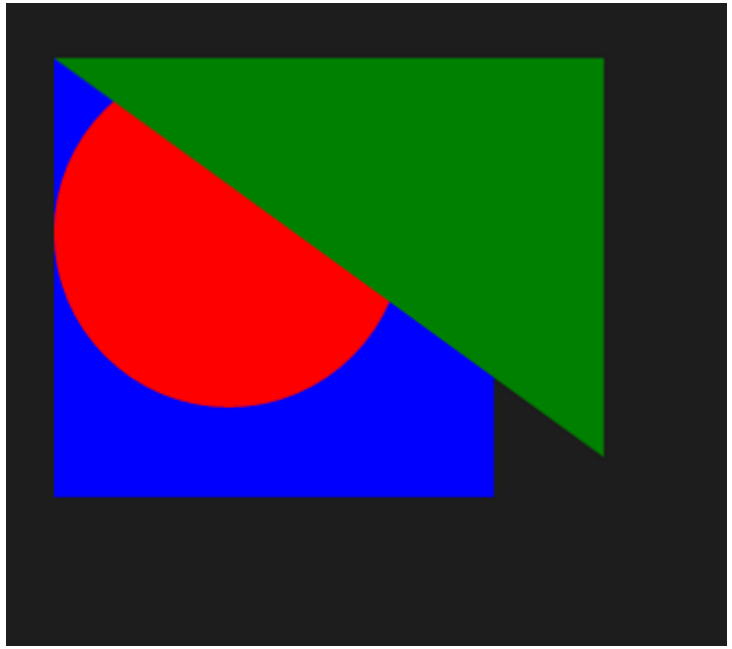
Controls in the XAML toolbox

Layout controls

```
<Border Thickness="" Brush="" />

<Canvas />

<Grid />

<RelativePanel />

<ScrollViewer />

<SplitView DisplayMode="" />

<StackPanel Orientation="" />

<VariableSizedWrapGrid />

<ViewBox Stretch="" />
```

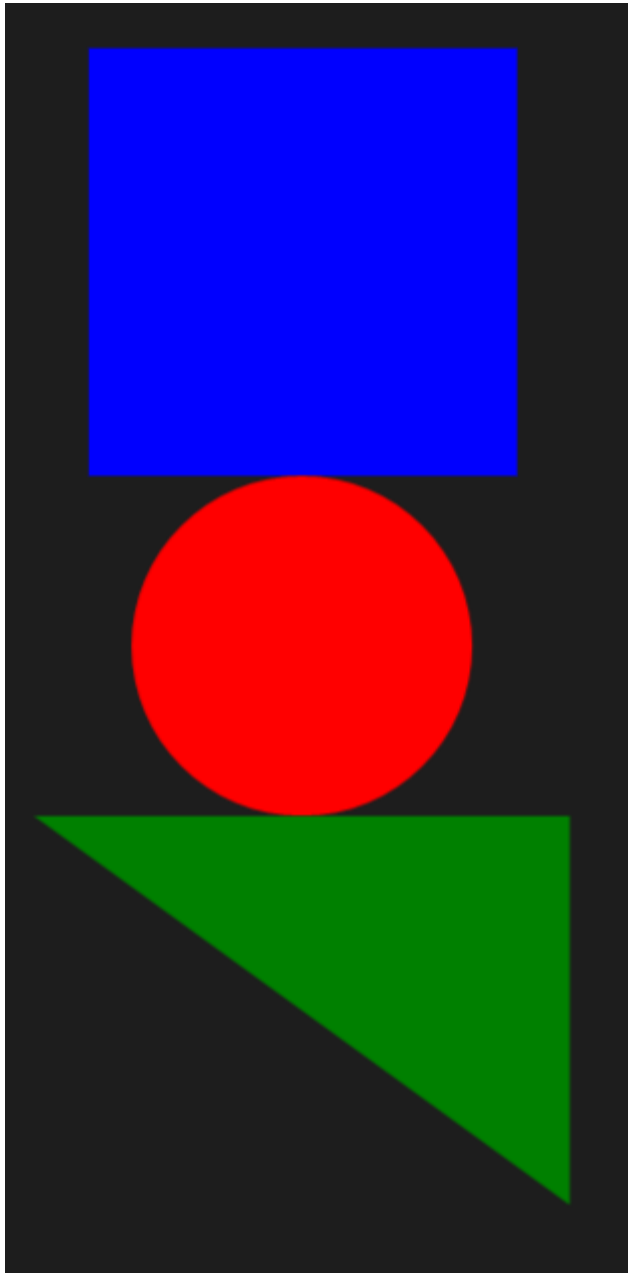
Canvas

```
<Canvas>
    <Rectangle Fill="Blue"
               Width="220"
               Height="220" />
    <Ellipse Fill="Red"
             Width="175"
             Height="175" />
    <Polygon Fill="Green"
             Points="0,0 275,0 275,200"/>
</Canvas>
```



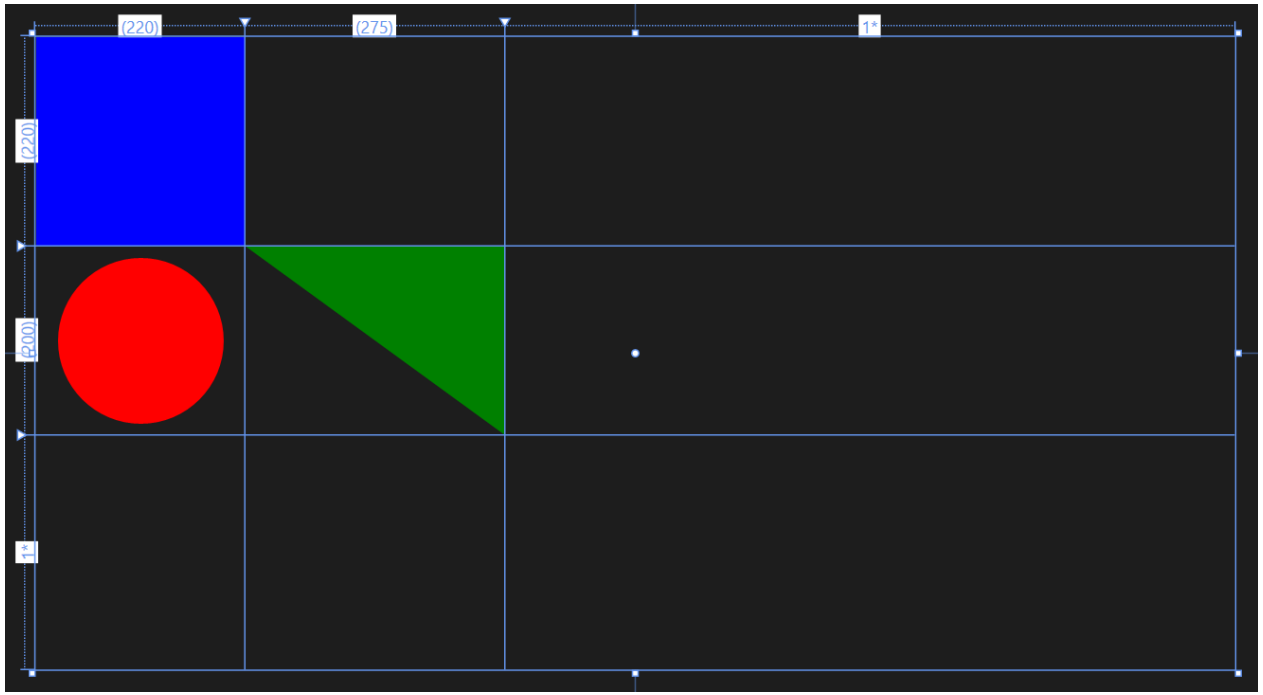
StackPanel

```
<StackPanel Margin="50"
            HorizontalAlignment="Left"
            VerticalAlignment="Top">
    <Rectangle Fill="Blue"
               Width="220"
               Height="220" />
    <Ellipse Fill="Red"
             Width="175"
             Height="175" />
    <Polygon Fill="Green"
            Points="0,0 275,0 275,200"
            StrokeThickness="2" />
</StackPanel>
```



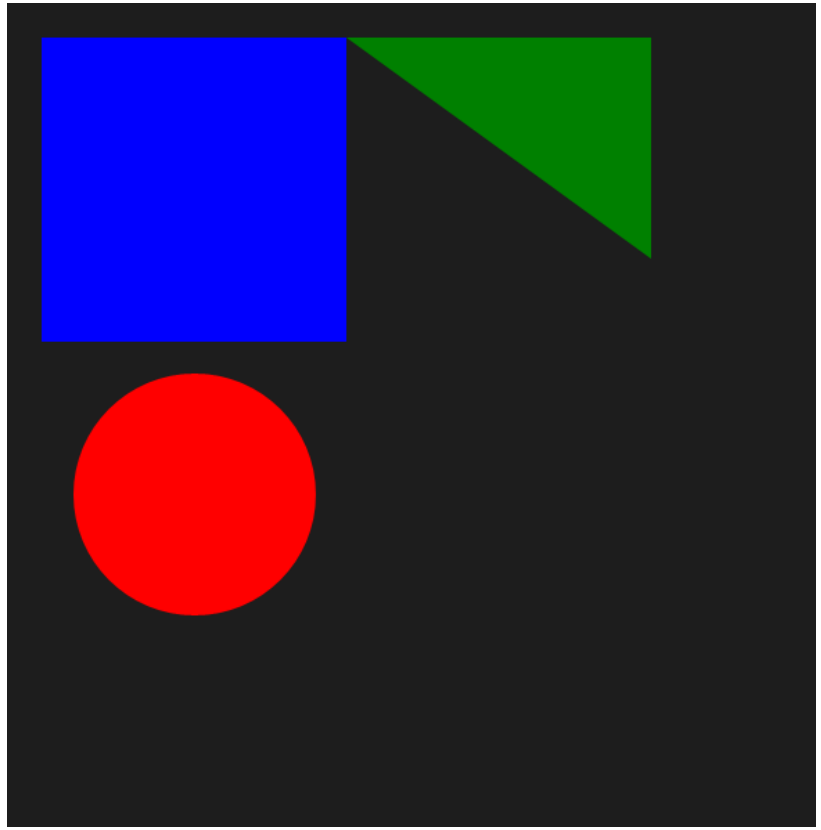
Grid

```
<Grid Margin="50">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Blue" Width="220" Height="220"
               Grid.Row="0" Grid.Column="0" />
    <Ellipse Fill="Red" Width="175" Height="175"
             Grid.Row="1" Grid.Column="0" />
    <Polygon Fill="Green" Points="0,0 275,0 275,200"
             StrokeThickness="2" Grid.Row="1" Grid.Column="1" />
</Grid>
```



WrapGrid

```
<ItemsControl Margin="50">
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <WrapGrid MaximumRowsOrColumns="2"/>
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>

    <ItemsControl.Items>
        <Rectangle Fill="Blue"
               Width="220"
               Height="220" />
        <Ellipse Fill="Red"
             Width="175"
             Height="175" />
        <Polygon Fill="Green"
             Points="0,0 275,0 275,200"/>
    </ItemsControl.Items>
</ItemsControl>
```

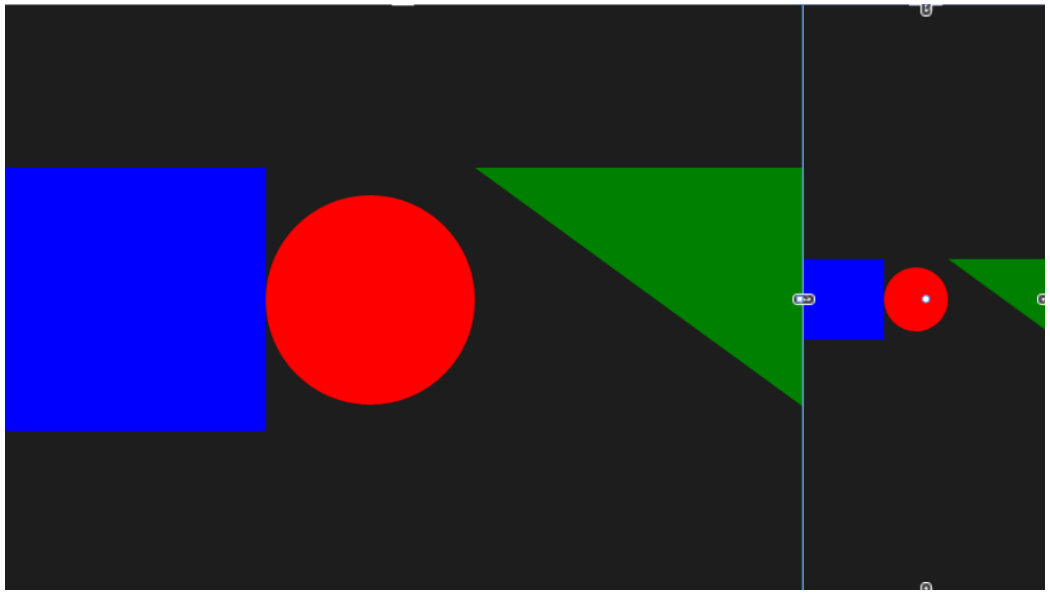


ScrollViewer

```
<ScrollViewer Margin="50">
    <StackPanel Margin="50"
            HorizontalAlignment="Left"
            VerticalAlignment="Top">
        <Rectangle Fill="Blue" Width="220" Height="220" />
        <Ellipse Fill="Red" Width="175" Height="175" />
        <Polygon Fill="Green" Points="0,0 275,0 275,200"/>
        <Rectangle Fill="Blue" Width="220" Height="220" />
        <Ellipse Fill="Red" Width="175" Height="175" />
        <Polygon Fill="Green" Points="0,0 275,0 275,200" />
    </StackPanel>
</ScrollViewer>
```

Viewbox

```
<Viewbox Grid.Column="0">
    <StackPanel Orientation="Horizontal">
        <Rectangle Fill="Blue" Width="220" Height="220" />
        <Ellipse Fill="Red" Width="175" Height="175" />
        <Polygon Fill="Green" Points="0,0 275,0 275,200"
            StrokeThickness="2" />
    </StackPanel>
</Viewbox>

<Viewbox Grid.Column="1" Stretch="Uniform">
    <StackPanel Orientation="Horizontal">
        <Rectangle Fill="Blue" Width="220" Height="220" />
        <Ellipse Fill="Red" Width="175" Height="175" />
        <Polygon Fill="Green" Points="0,0 275,0 275,200"
            StrokeThickness="2" />
    </StackPanel>
</Viewbox>
```

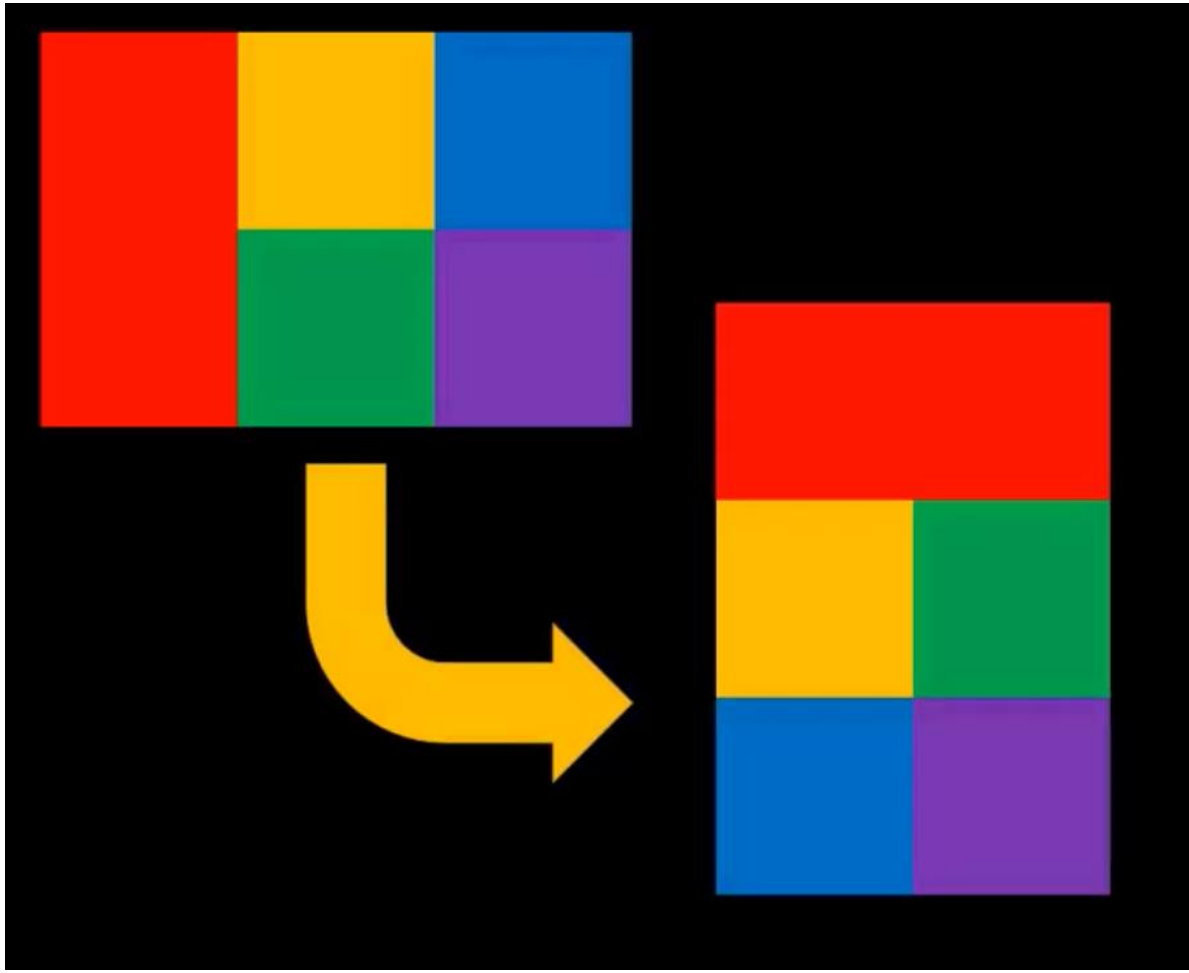


RelativePanel

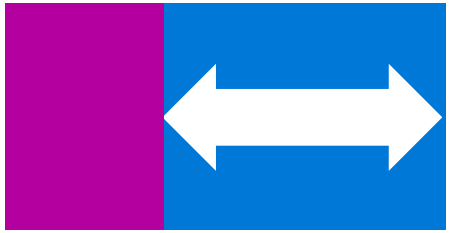
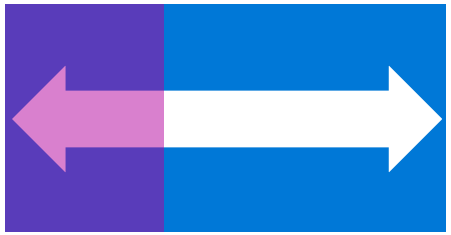
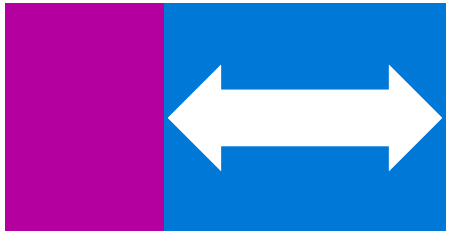
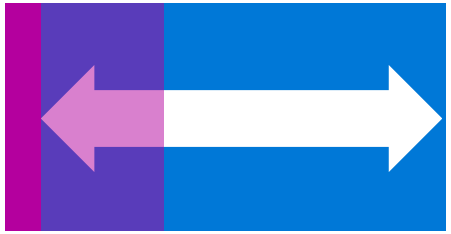
Some child elements act as anchors

Most child elements relate to others

It's a layout technique friendly with States

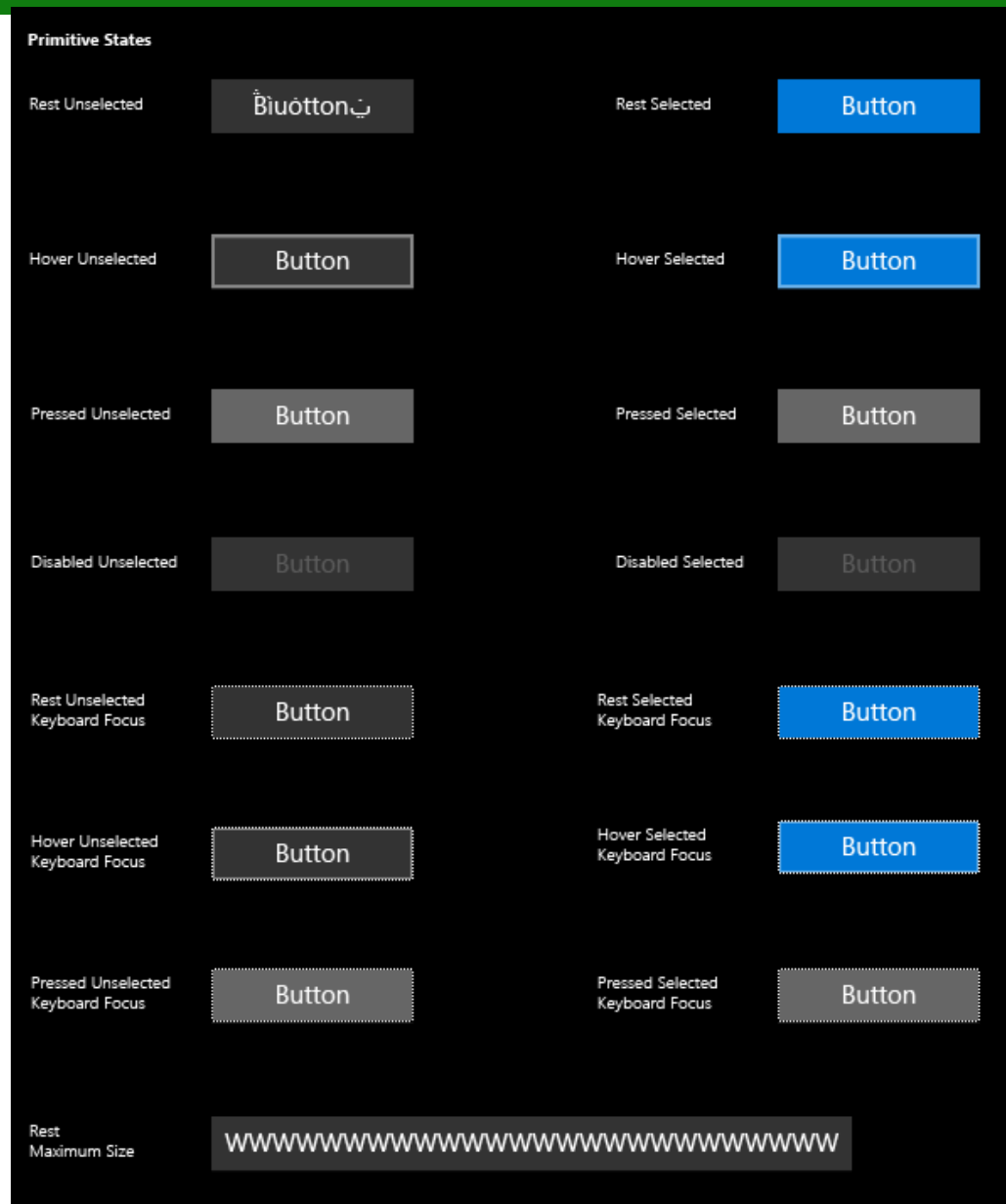


SplitView

	IsPaneOpen="True"	IsPaneOpen="False"
DisplayMode= "Inline"		
DisplayMode= "Overlay"		
DisplayMode= "CompactInline"		
DisplayMode= "CompactOverlay"		

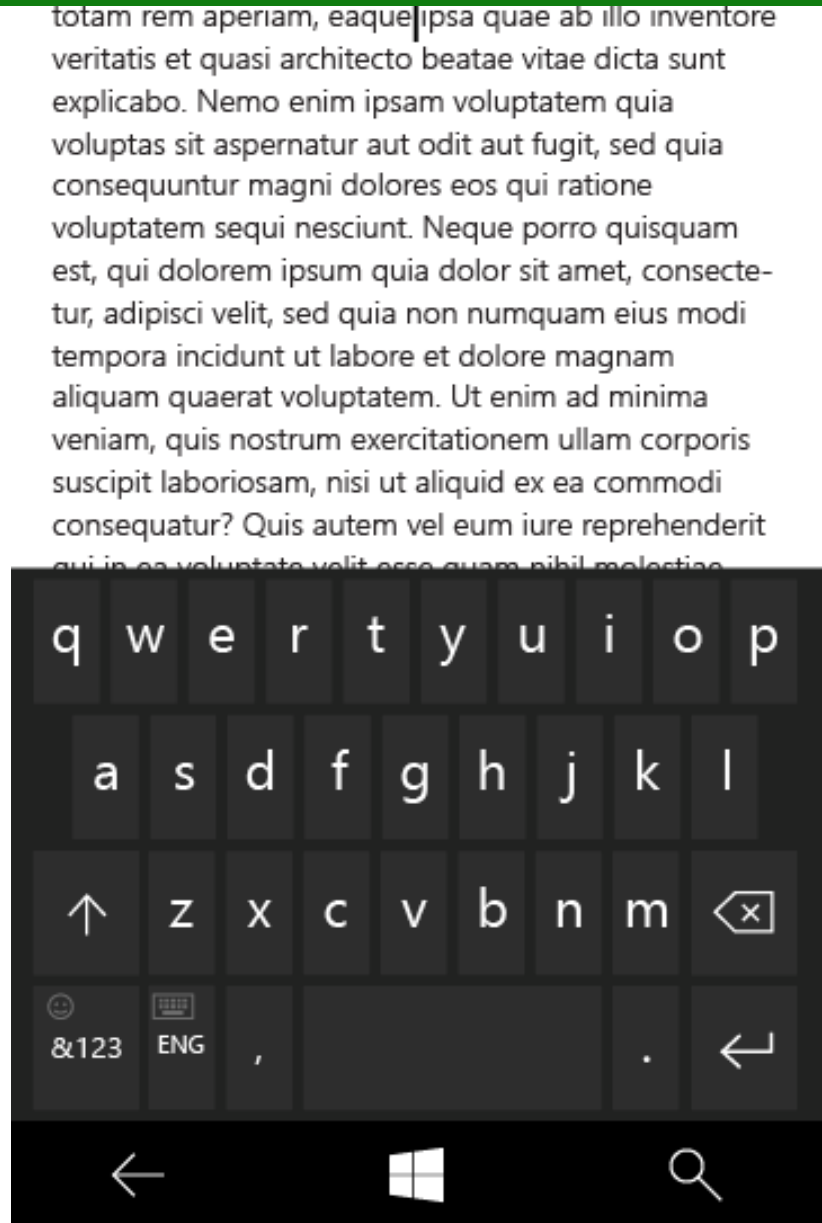
Buttons

```
<Button Content="" />

<HyperlinkButton />

<RepeatButton />

<ToggleButton IsChecked="" />
```



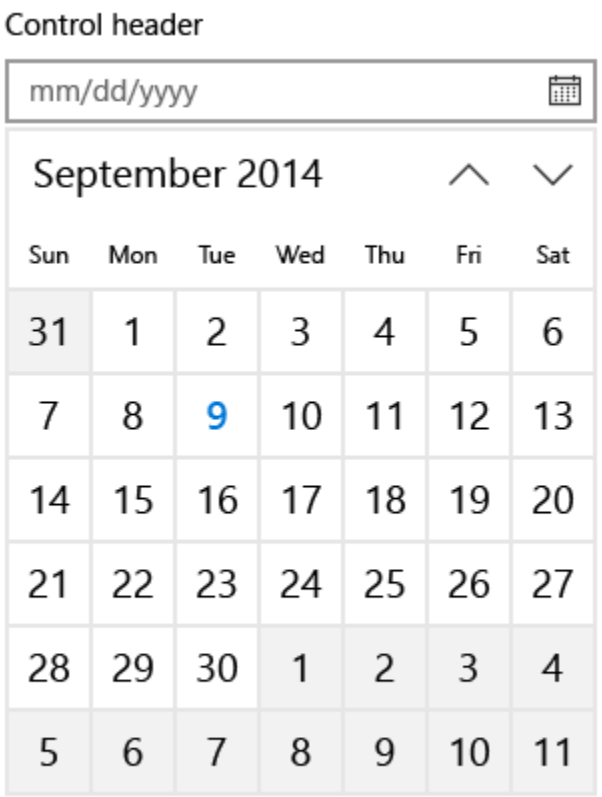
Text controls

```
<TextBox Text="" />

<PasswordBox Text="" />

<TextBlock Text="" />

<RichEditBox Content="" />

<RichTextBlock Content="" />

<BitmapIcon UriSource="" />

<FontIcon Glyph="" />

<SymbolIcon Icon="" />

<PathIcon Data="" />
```



Selection and picker controls

```
<CalendarView />

<DatePicker Date="" />

<TimePicker Time="" />

<ToggleSwitch IsChecked="" />

<CheckBox IsCHecked="" />

<RadioButton IsChecked="" />

<ComboBox />

<ListBox />

<Slider Minimum="" Maximum="" />
```



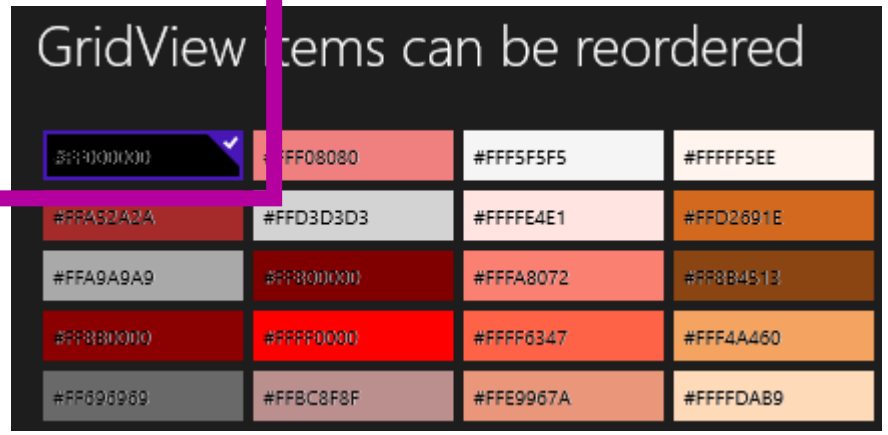
Collection/data controls

```
<ItemsControl ItemsSource="" />

<FlipView />

<GridView />

<ListView />

<SematicZoom />

<Hub Orientation="" />

<ContentControl ControlTemplate="" />
```

```
<DataTemplate />

<ItemsPanelTemplate />

<ControlTemplate />

<ContentPresenter />

<ItemsPresenter />

<ContentControl />
```

Grid View Syntax

```
<GridView ItemsSource="{Binding Source={StaticResource MyCsv}}"
          CanReorderItems="True" CanDragItems="True"
          AllowDrop="True" Padding="120,140,80,80">
    <GridView.ItemTemplate>
        <DataTemplate>
            <Grid Background="{Binding Brush}" Height="50" Width="200">
                <TextBlock FontSize="20" Text="{Binding Text}"
                           VerticalAlignment="Center" Margin="11"
                           Foreground="Gainsboro" />
                <TextBlock FontSize="20" Text="{Binding Text}"
                           VerticalAlignment="Center" Margin="10"
                           Foreground="Black" />
            </Grid>
        </DataTemplate>
    </GridView.ItemTemplate>
</GridView>
```



List View Syntax

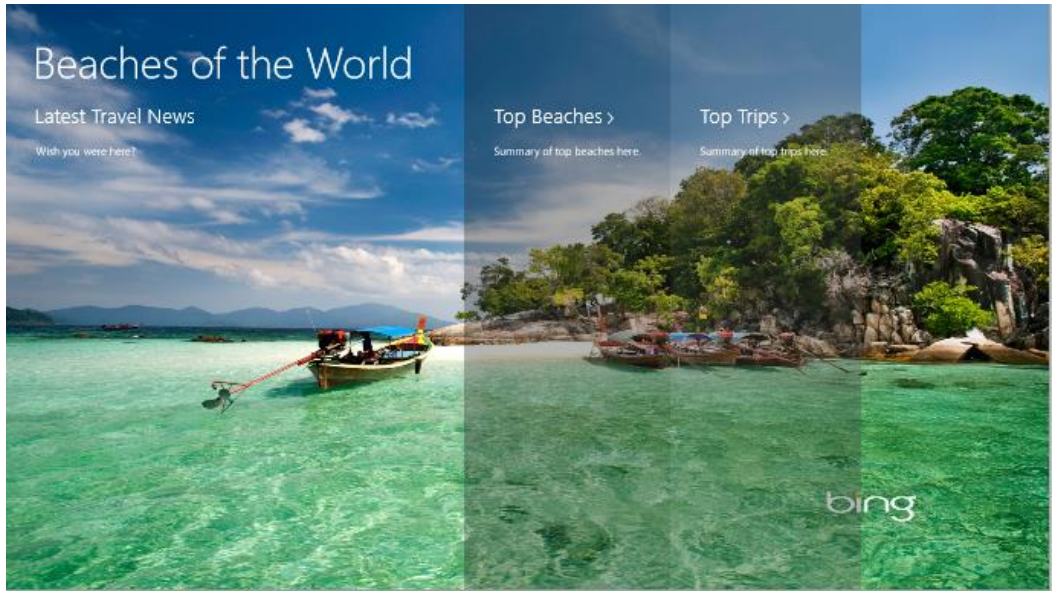
```
<ListView ItemsSource="{Binding Source={StaticResource MyCsv}}"
          CanReorderItems="True" CanDragItems="True"
          AllowDrop="True" Padding="40,40,40,80" Grid.Column="1">
    <ListView.Header>
        <TextBlock Style="{StaticResource HeaderTextBlockStyle}"
                   Margin="0,0,0,60">ListView</TextBlock>
    </ListView.Header>
    <ListView.ItemTemplate>
        <DataTemplate>
            <Grid Background="{Binding Brush}" Height="50" Width="220">
                <TextBlock FontSize="20" Text="{Binding Text}"
                           VerticalAlignment="Center" Margin="11"
                           Foreground="Gainsboro" />
                <TextBlock FontSize="20" Text="{Binding Text}"
                           VerticalAlignment="Center" Margin="10"
                           Foreground="Black" />
            </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Flip View Syntax

```
<FlipView BorderBrush="Black" BorderThickness="1"
          ItemsSource="{Binding Images}">
    <FlipView.ItemTemplate>
        <DataTemplate>
            <Grid>
                <Image Source="{Binding ImageUri}" Stretch="UniformToFill"/>
                <Border Background="#A5000000" Height="80"
                        VerticalAlignment="Bottom">
                    <TextBlock Text="{Binding Title}" FontFamily="Segoe UI"
                               FontSize="26.667" Foreground="#CCFFFFFF"
                               Padding="15,20"/>
                </Border>
            </Grid>
        </DataTemplate>
    </FlipView.ItemTemplate>
</FlipView>
```

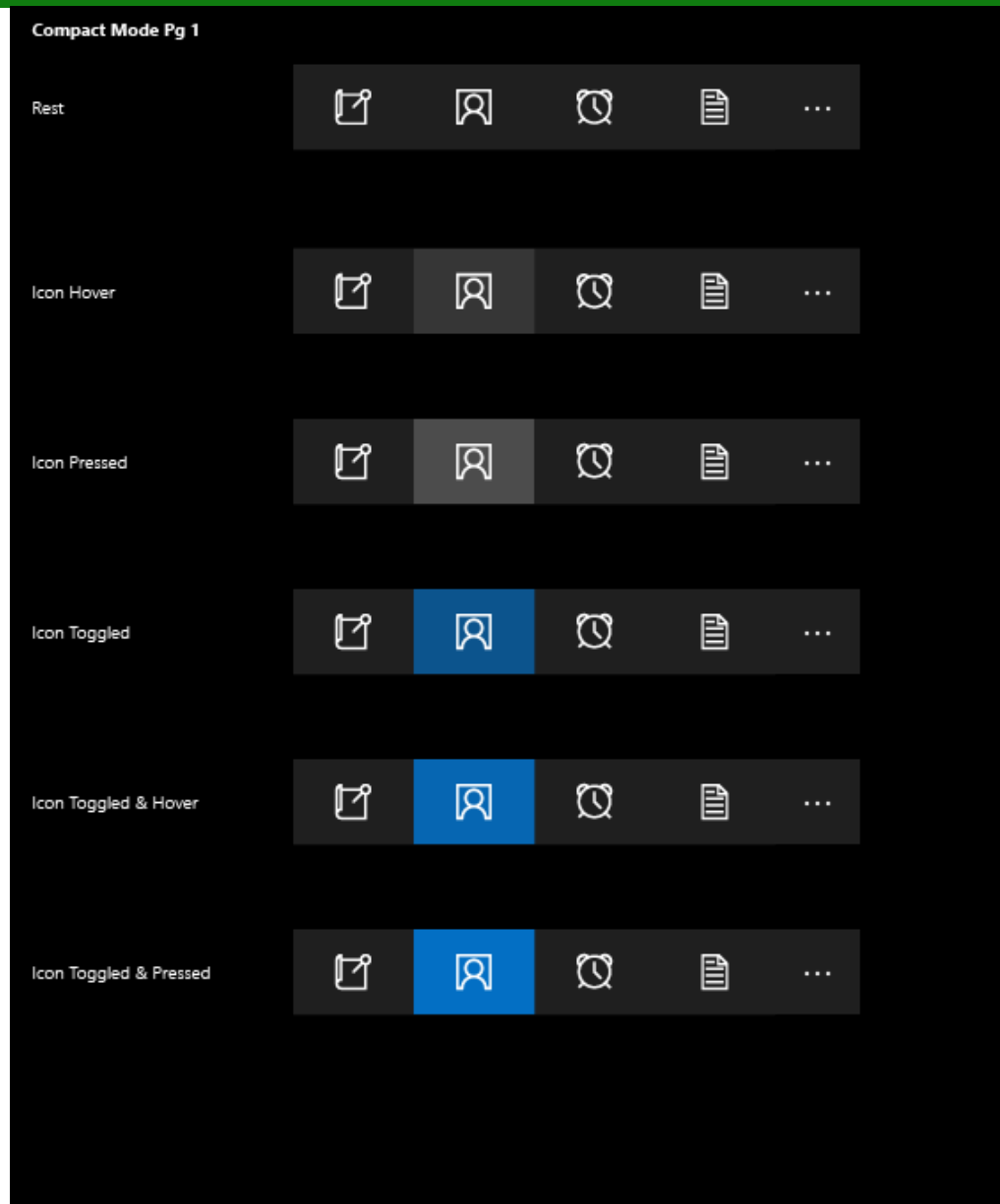
Hub Syntax

```
<Hub Header="Beaches of the World">
    <HubSection Width="600" Header="Latest Travel News">
        <DataTemplate>
            <Grid>
                <TextBlock Text="Wish you were here?" Style="{ThemeResource BodyTextBlockStyle}" />
            </Grid>
        </DataTemplate>
    </HubSection>
    <HubSection Header="Top Beaches" IsHeaderInteractive="True" Background="#55222222" MinWidth="250">
        <DataTemplate>
            <StackPanel>
                <TextBlock Text="Summary of top beaches here."
                   Style="{ThemeResource BodyTextBlockStyle}" />
            </StackPanel>
        </DataTemplate>
    </HubSection>
    <HubSection Header="Top Trips" IsHeaderInteractive="True"
                Background="#55444444" MinWidth="250">
        <DataTemplate>
            <StackPanel>
                <TextBlock Text="Summary of top trips here."
                   Style="{ThemeResource BodyTextBlockStyle}" />
            </StackPanel>
        </DataTemplate>
    </HubSection>
</Hub>
```



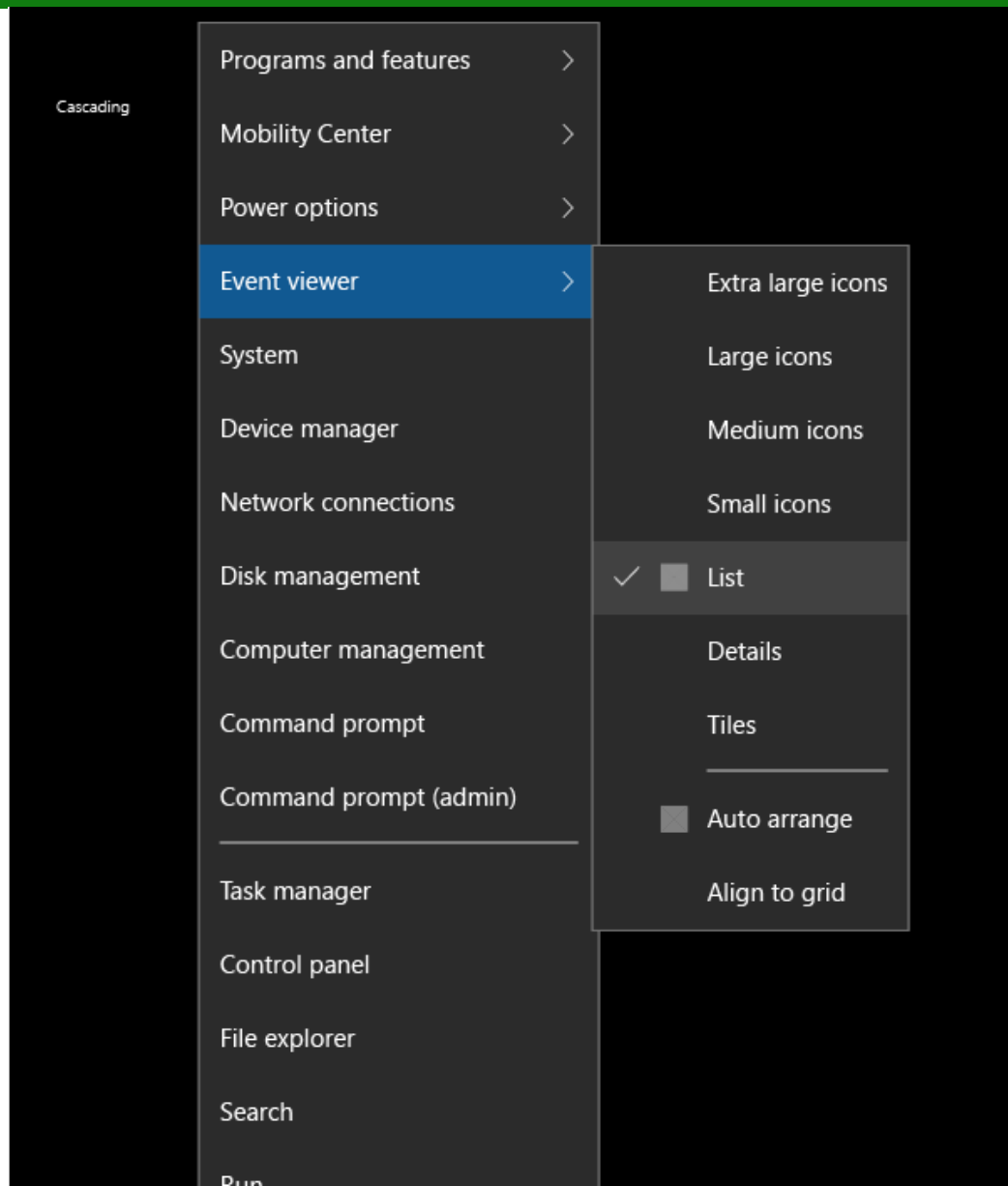
App bars and commands

```
<AppBar />

<CommandBar />

<AppBarButton Label="" Icon="" />

<AppBarToggleButton IsChecked="" />

<AppBarSeparator />
```



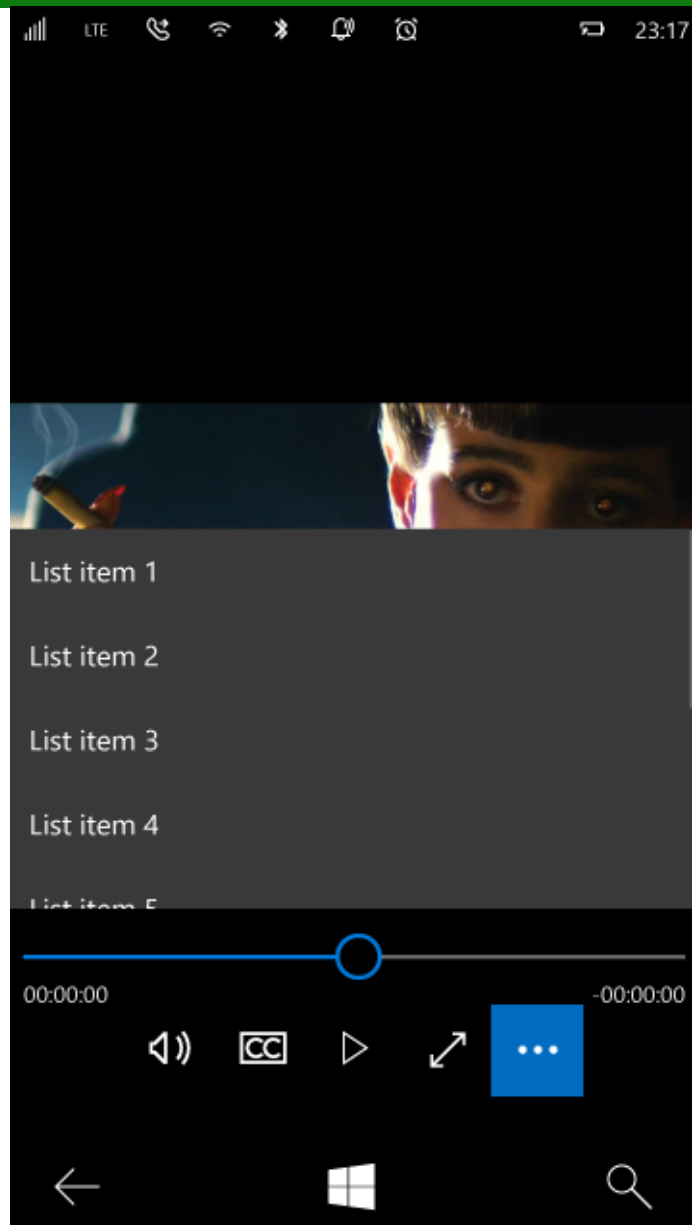
Flyout controls

```
<Popup IsOpen="" />

<ContentDialog Title="" Content="" />

<Flyout Placement="" />

<MenuFlyout />

<ToolTip />
```



Progress, Media & Inking controls

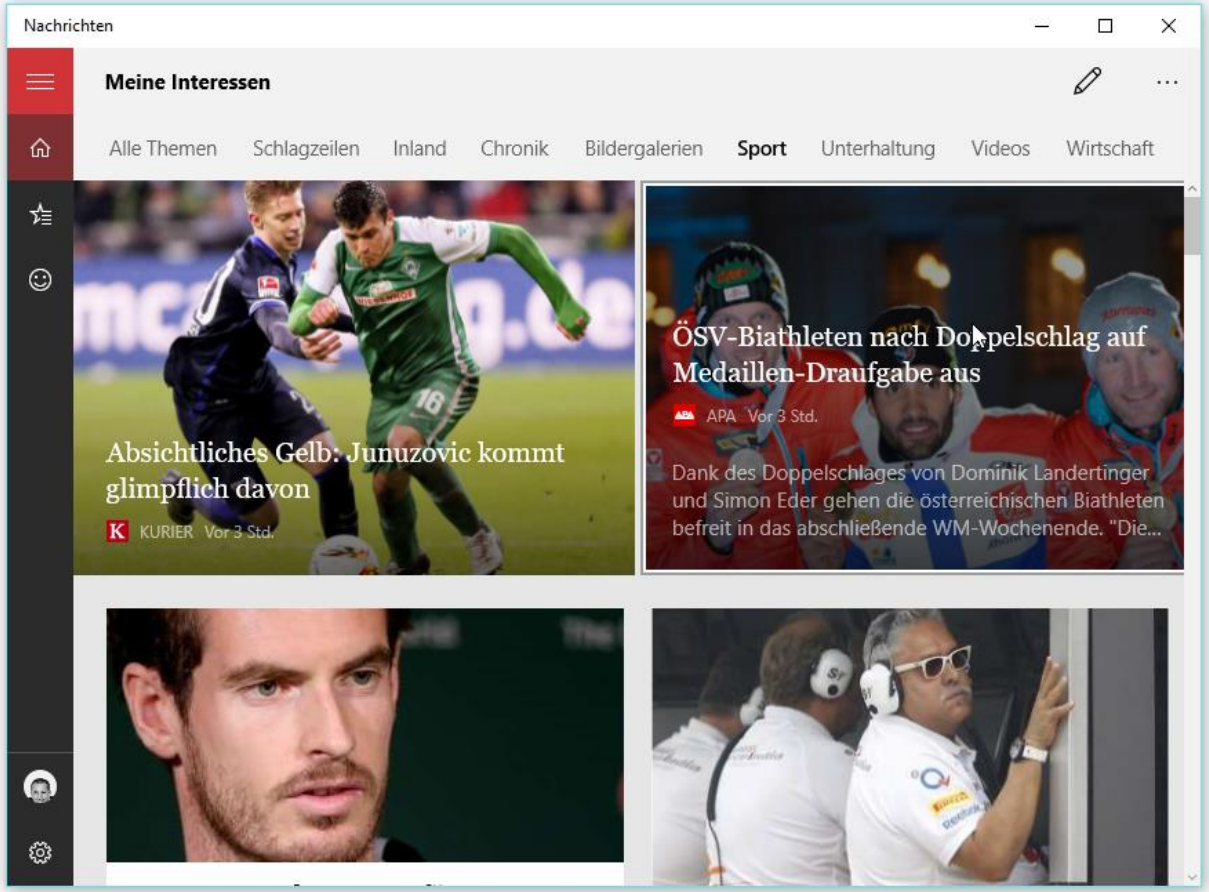
```
<ProgressBar IsIntermediate="" />

<ProgressRing IsActive="" />

<Image />

<InkCanvas />

<MediaElement />

<CaptureElement />
```



Design Änderungen

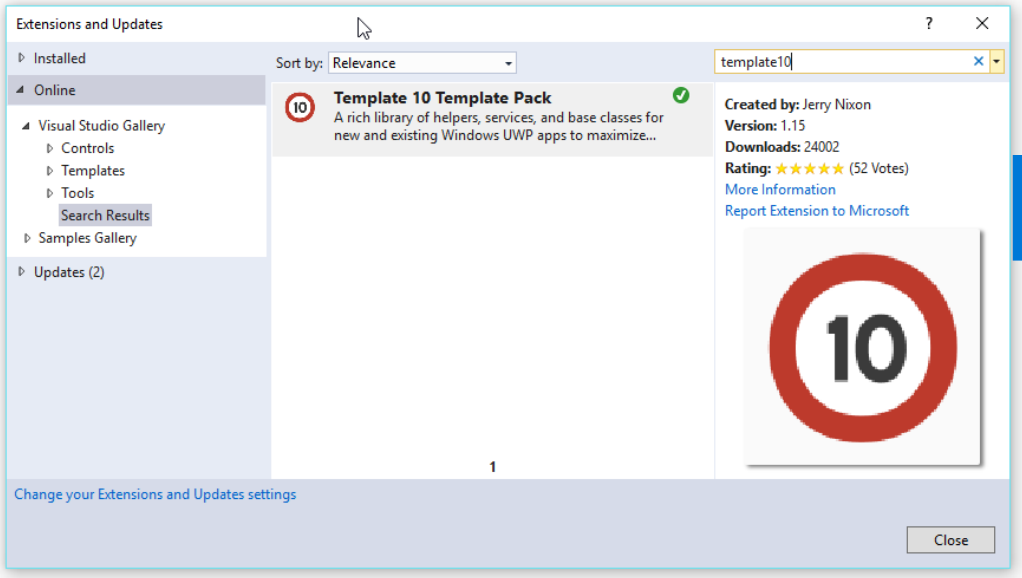
… gegenüber Windows 8.1

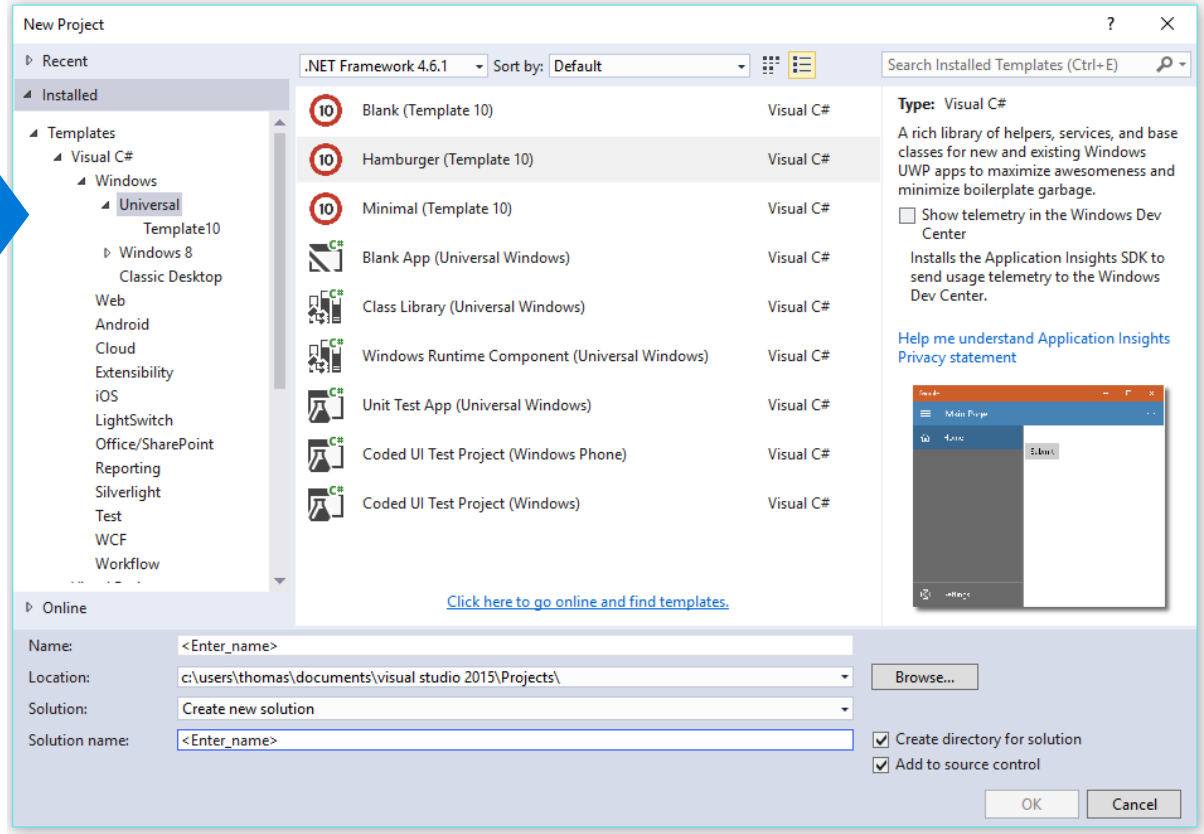
- neues Hamburger Menü
- Back-button in Titelleiste
- Weniger Whitespace
- 4 Pixel-Grid (statt 10)
- Vertikales Scrollen (statt horizontal)
- …



Template 10

http://aka.ms/template10





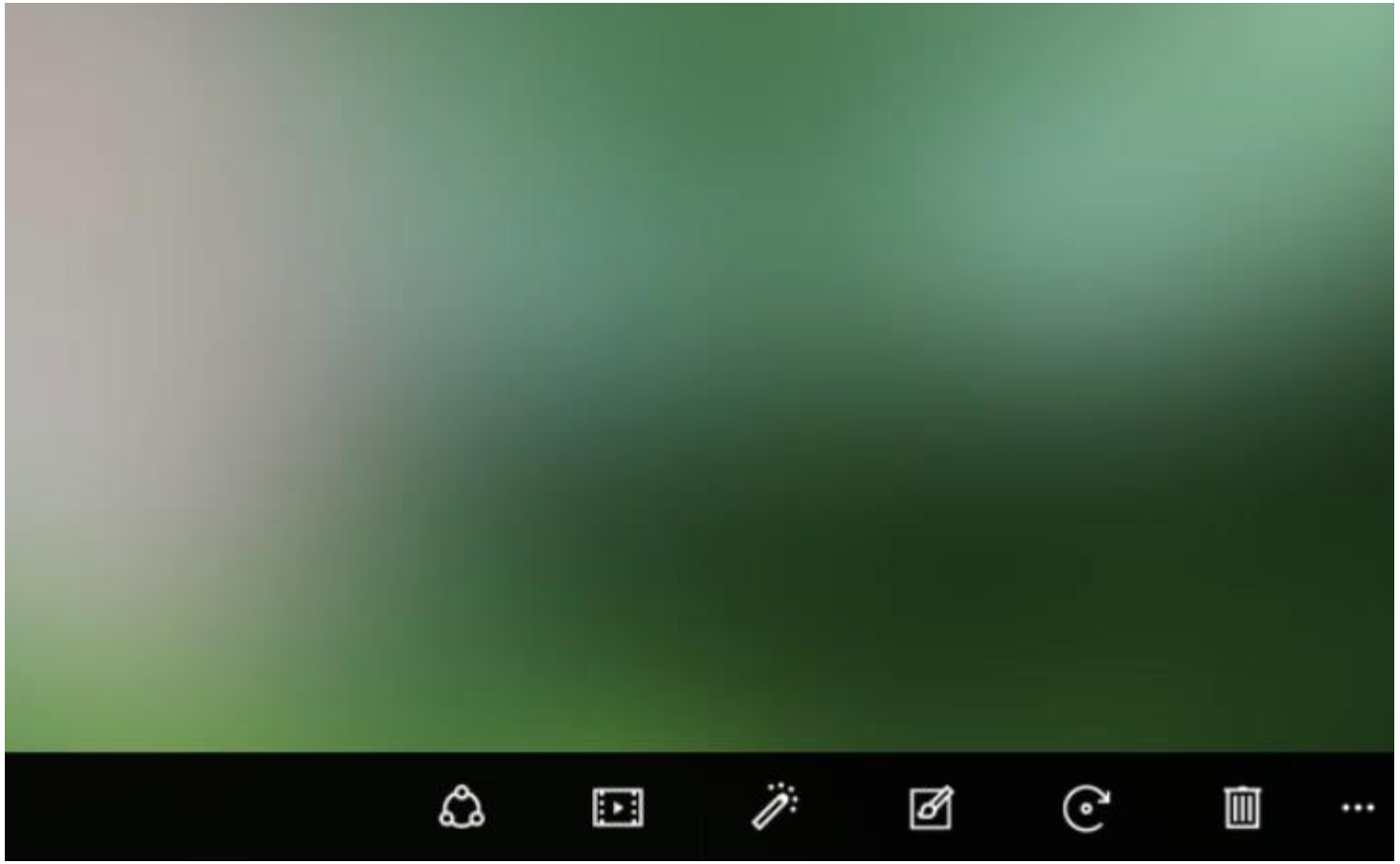
Adaptive Apps

Windows

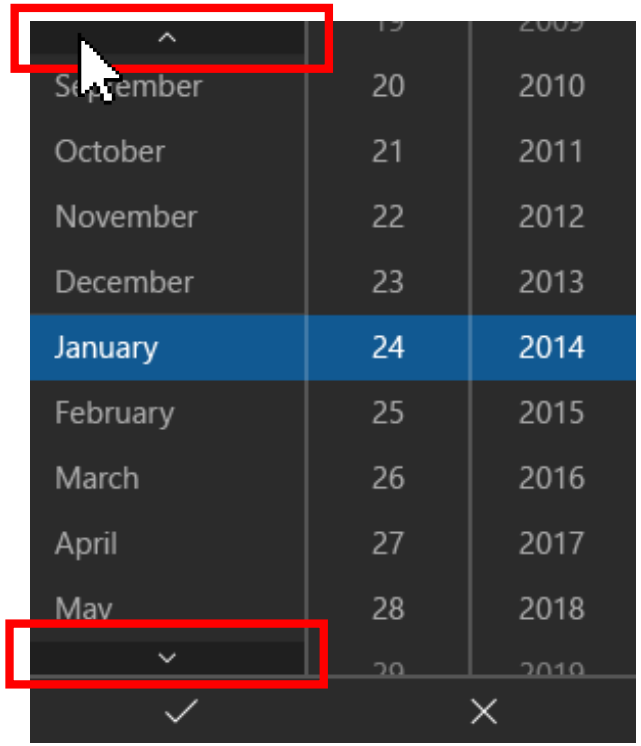
Adaptive Design

Windows

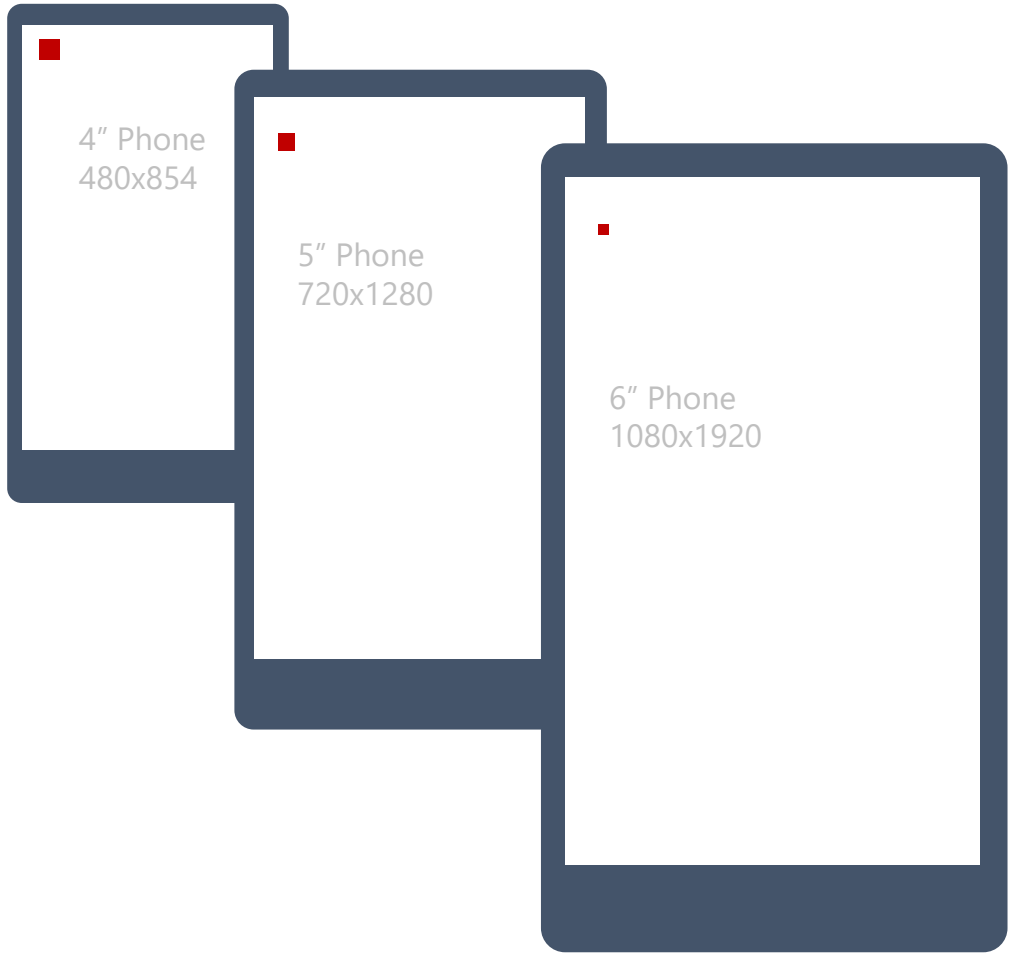
Adaptive controls

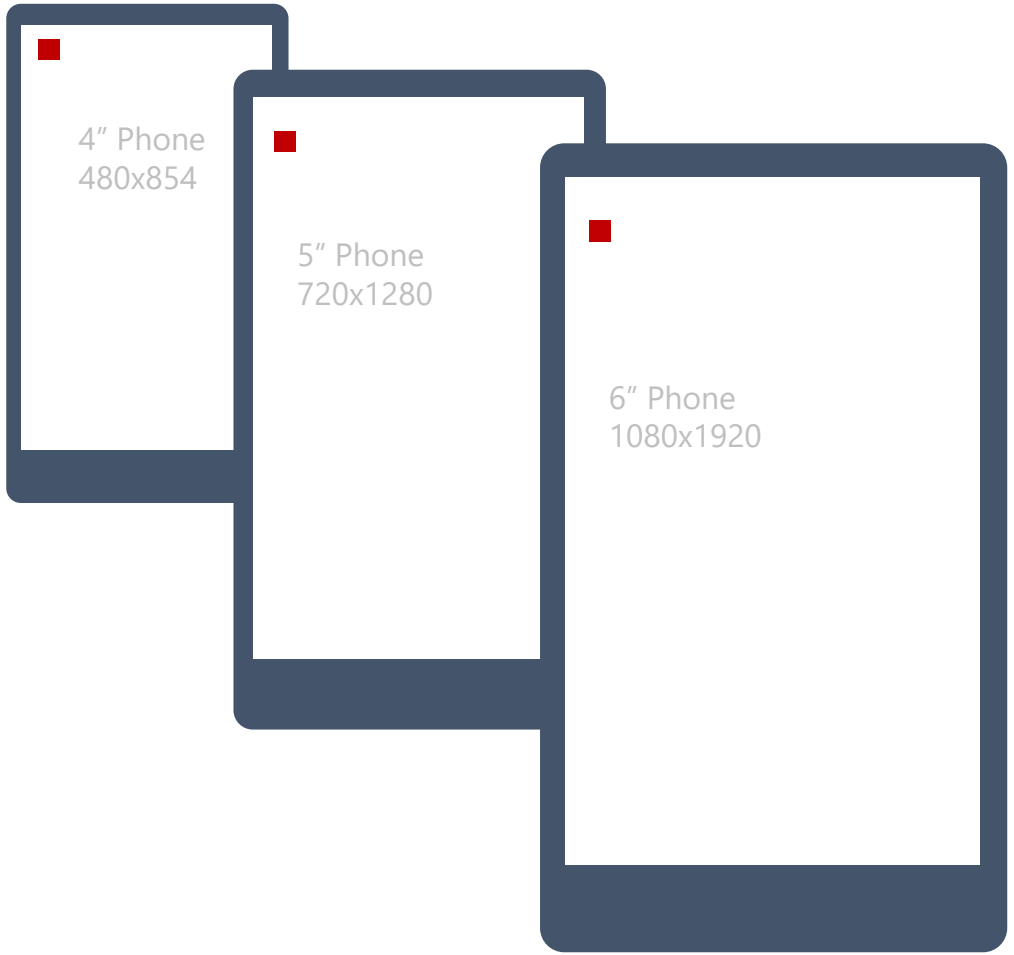


Input intelligence

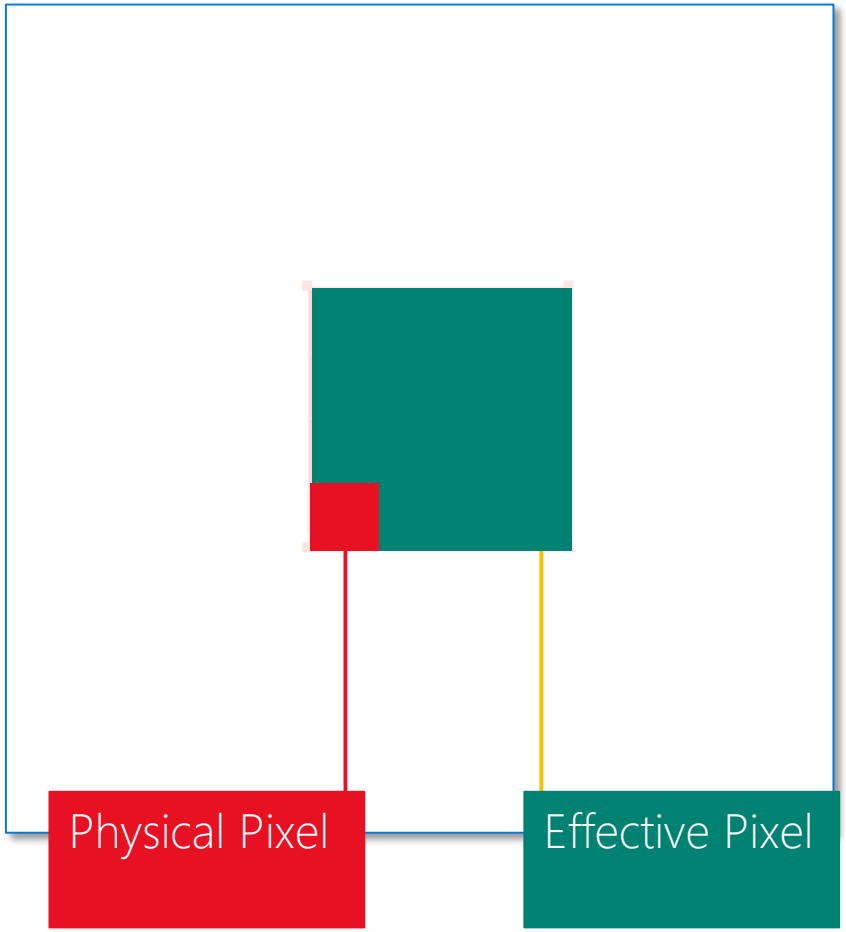


Physical versus effective pixel

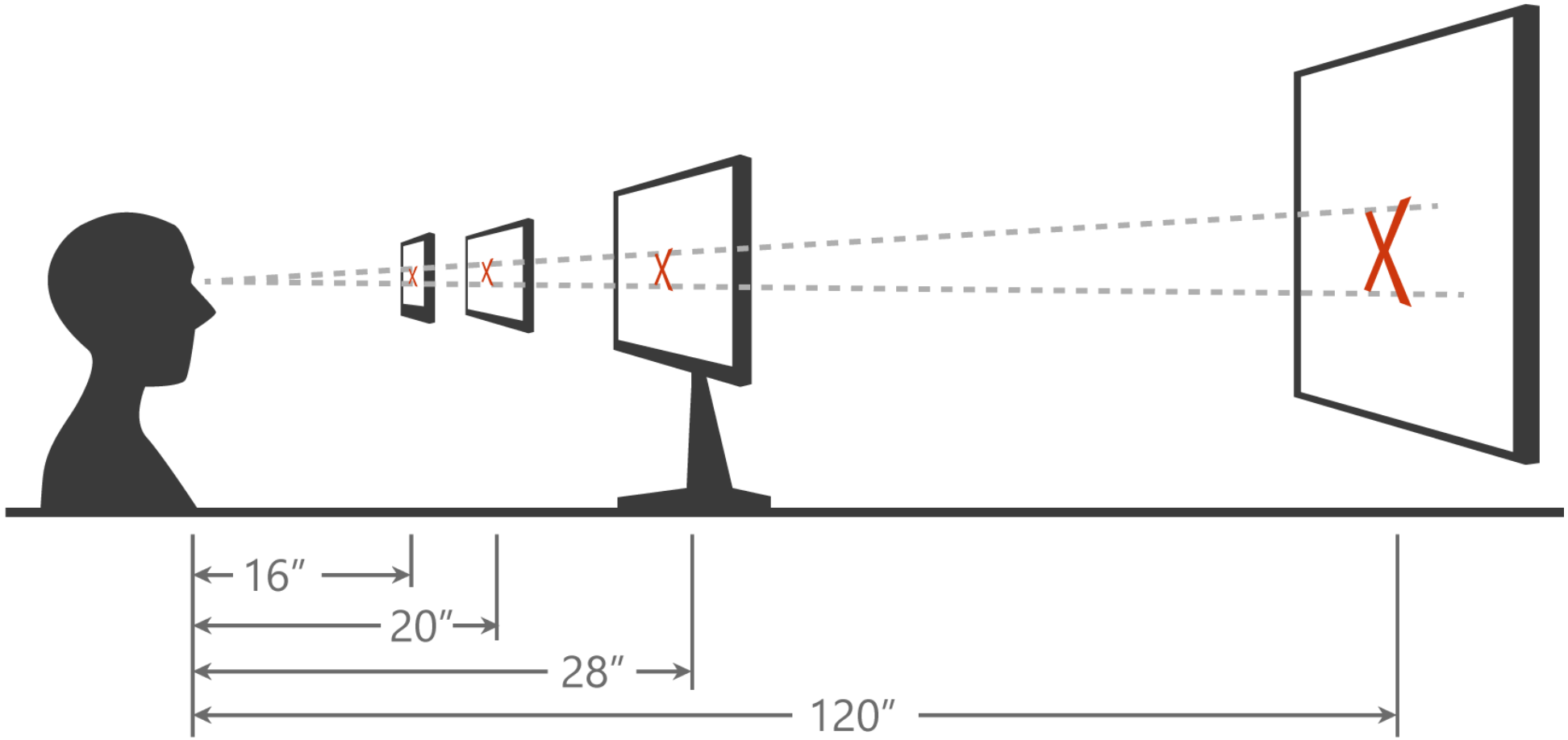




Effective pixel

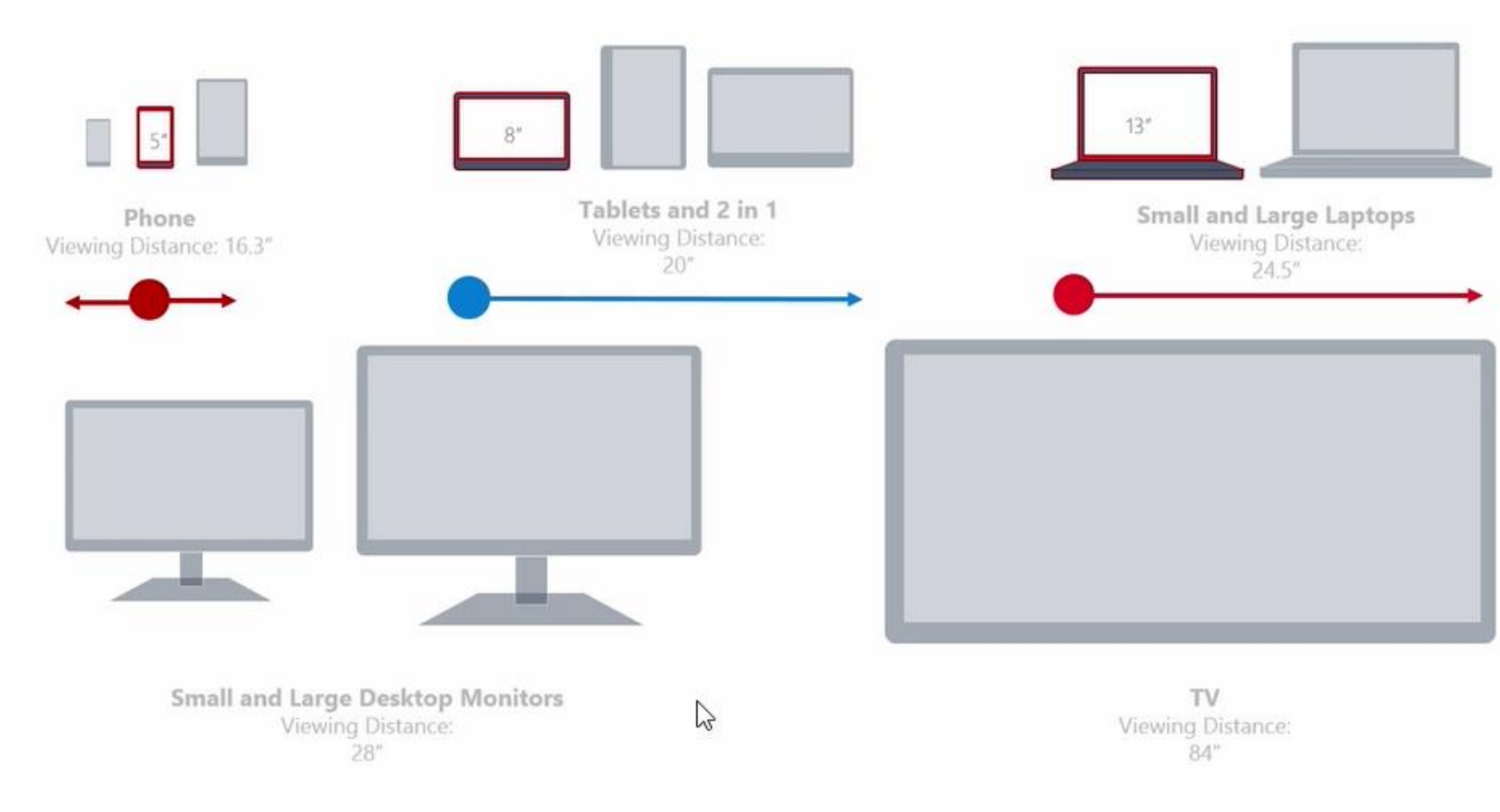


Scaling algorithm



Automatic Scaling

Design for...


5"
Phone
Viewing Distance: 16.3"
8"
Tablets and 2 in 1
Viewing Distance:
20"
13"
Small and Large Laptops
Viewing Distance:
24.5"
Small and Large Desktop Monitors
Viewing Distance:
28"
TV
Viewing Distance:
84"

Three core adaptive approaches

Responsive design

Resize your content

Reflow your content

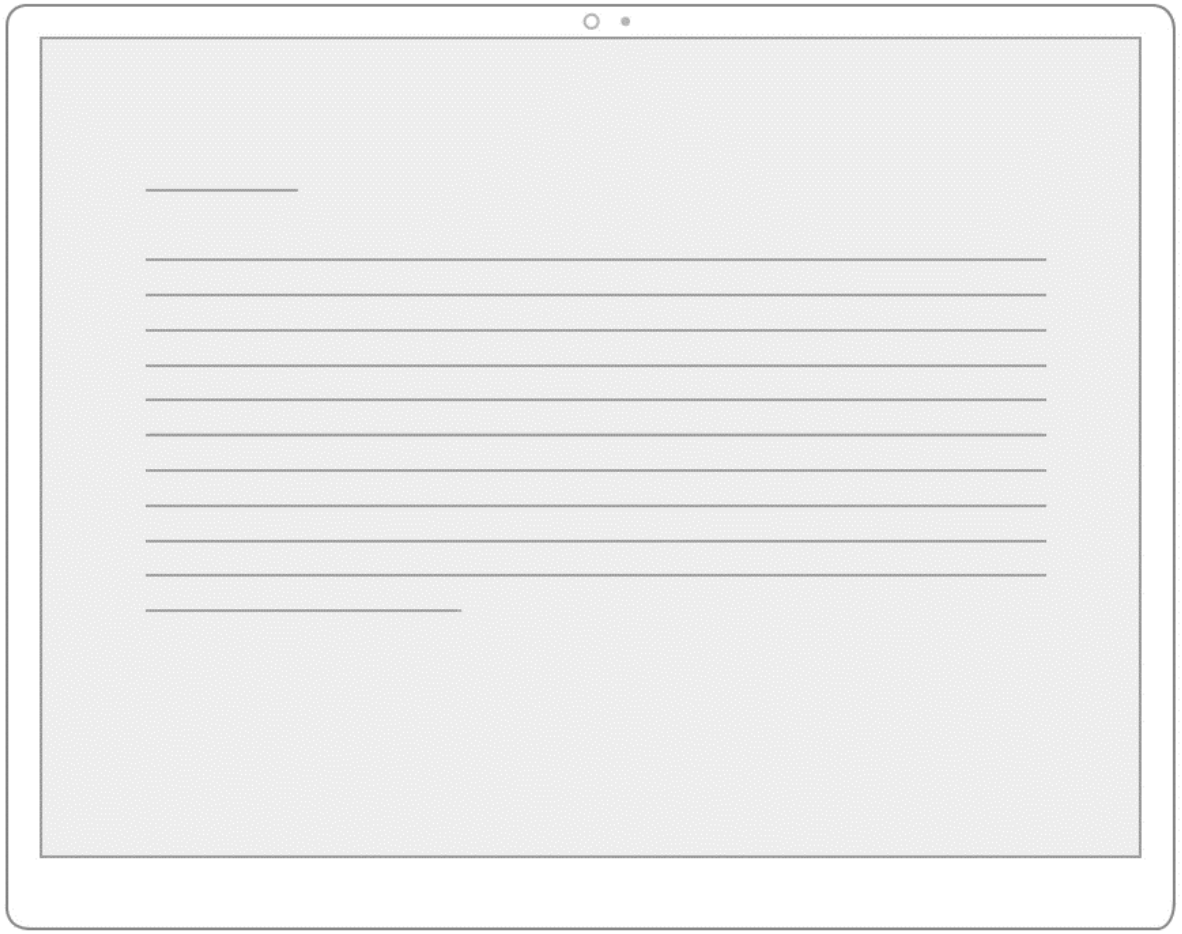
Adaptive design

Reposition your content

Redesign your content

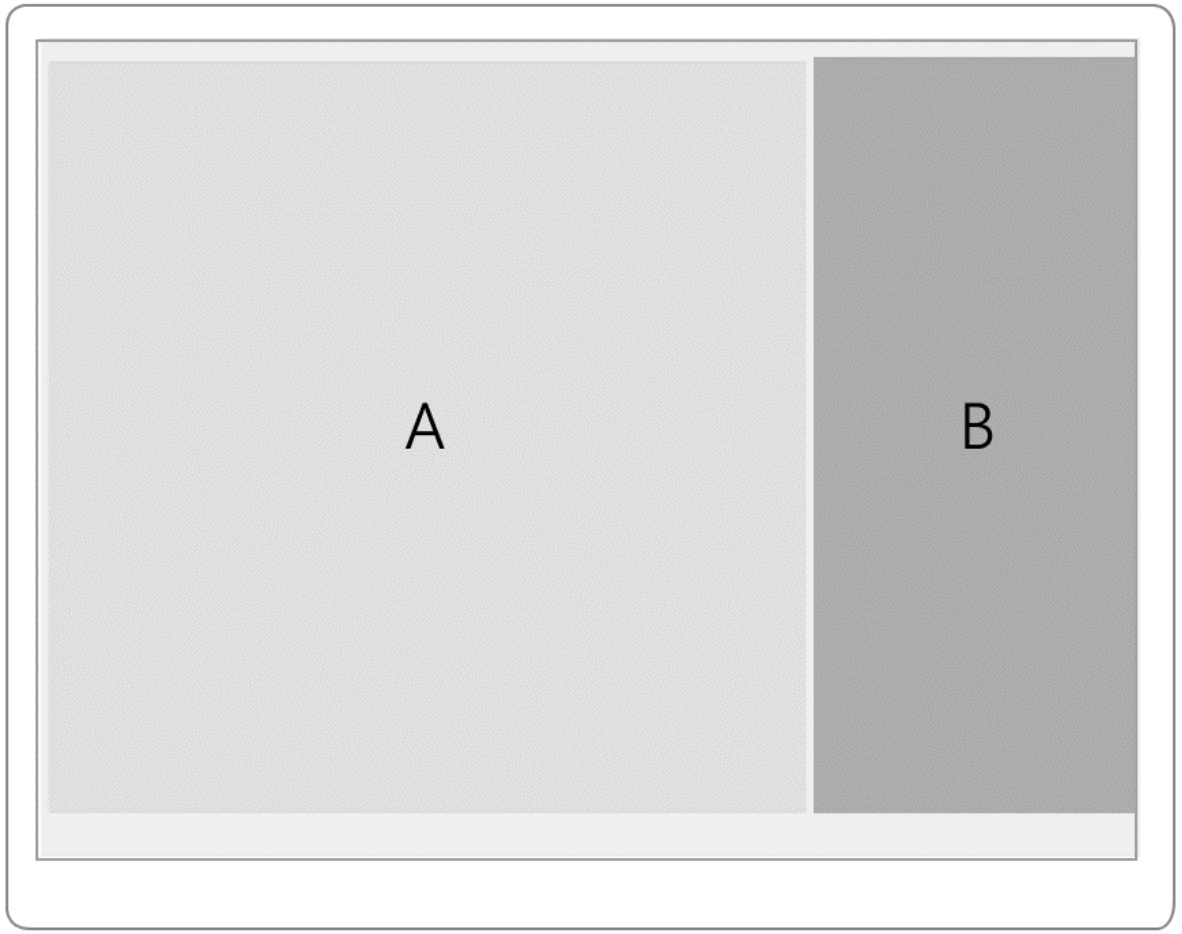
Tailored design

Resize

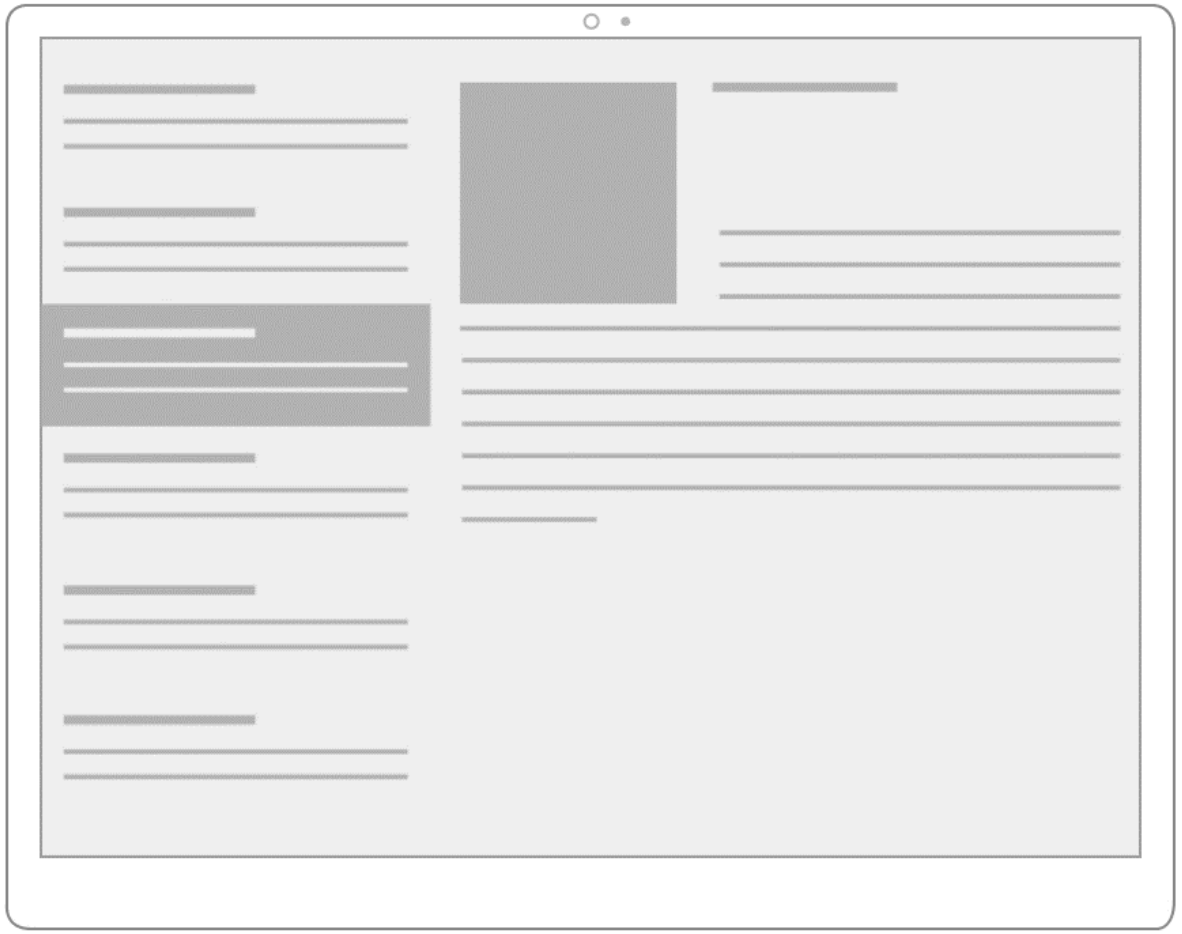


Reflow

Reposition



Redesign



How to set the visual state

VisualStateManager.Goto(element, state, transition)

```
public MainPage()
{
    this.InitializeComponent();
    this.SizeChanged += (s, e) =>
    {
        var state = "VisualState000min";
        if (e.NewSize.Width > 500)
            state = "VisualState500min";
        else if (e.NewSize.Width > 800)
            state = "VisualState800min";
        else if (e.NewSize.Width > 1000)
            state = "VisualState1000min";
        VisualStateManager.GoToState(this, state, true);
    };
}
```

Adaptive triggers

Code-free state transition

Two built-in triggers

MinWindowHeight (Taller than this)

MinWindowWidth (Wider than this)

```
<VisualState x:Name="VisualState500min">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="501" />
    </VisualState.StateTriggers>
</VisualState>
```

Custom adaptive triggers

Build to handle special cases.

```
public class DeviceFamilyTrigger : StateTriggerBase
{
    private string _deviceFamily;
    public string DeviceFamily
    {
        get { return _deviceFamily; }
        set
        {
            var qualifiers = Windows.ApplicationModel.Resources.Core
                .ResourceContext.GetForCurrentView().QualifierValues;
            if (qualifiers.ContainsKey("DeviceFamily"))
                SetActive(qualifiers["DeviceFamily"] == (_deviceFamily = value));
            else
                SetActive(false);
        }
    }
}
```

Custom adaptive triggers

also via NuGet

e.g.

https://github.com/Herdo/AdaptiveTriggerLibrary

Visual state setters

Setting a simple, scalar value

Great when you think of ENUM values like Visibility, Stretch, etc

Does not invoke a storyboard

Does not require ObjectAnimationUsingKeyFrames

```
<VisualState.Setters>
    <Setter Target="MyText01.FontSize" Value="24" />
    <Setter Target="MyImage.Stretch" Value="UniformToFill" />
    <Setter Target="MyImage.Height" Value="150" />
</VisualState.Setters>
```

XAML's RelativePanel control

A child or two act as a anchor elements

They are relative to the panel

Other children are relative to the anchors

RelativePanel.Above = "ElementName"
RelativePanel.RightOf = "ElementName"
RelativePanel.Below = "ElementName"
RelativePanel.LeftOf = "ElementName"

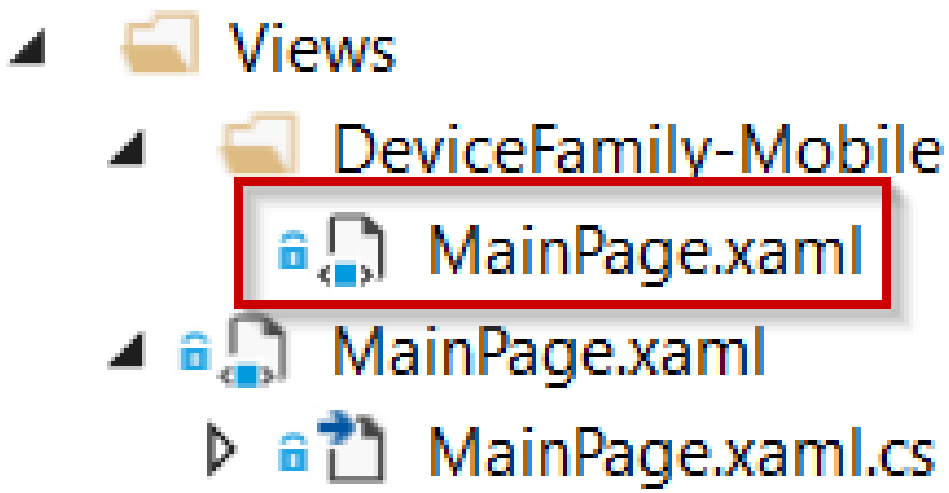
RelativePanel simplifies adaptive UI

A simple Visual State setter can rearrange the UI
One element can move a family of related elements

Device Family specific

File with same name as original XAML Page in device specific Folder

Codebehind is shared with original Page



Adaptive Code

Windows

Compilation directives

C# Syntax

```
#if WINDOWS_PHONE_APP
        Windows.Phone.UI.Input.HardwareButtons
                .BackPressed += this.HardwareButtons_BackPressed;
#endif
```

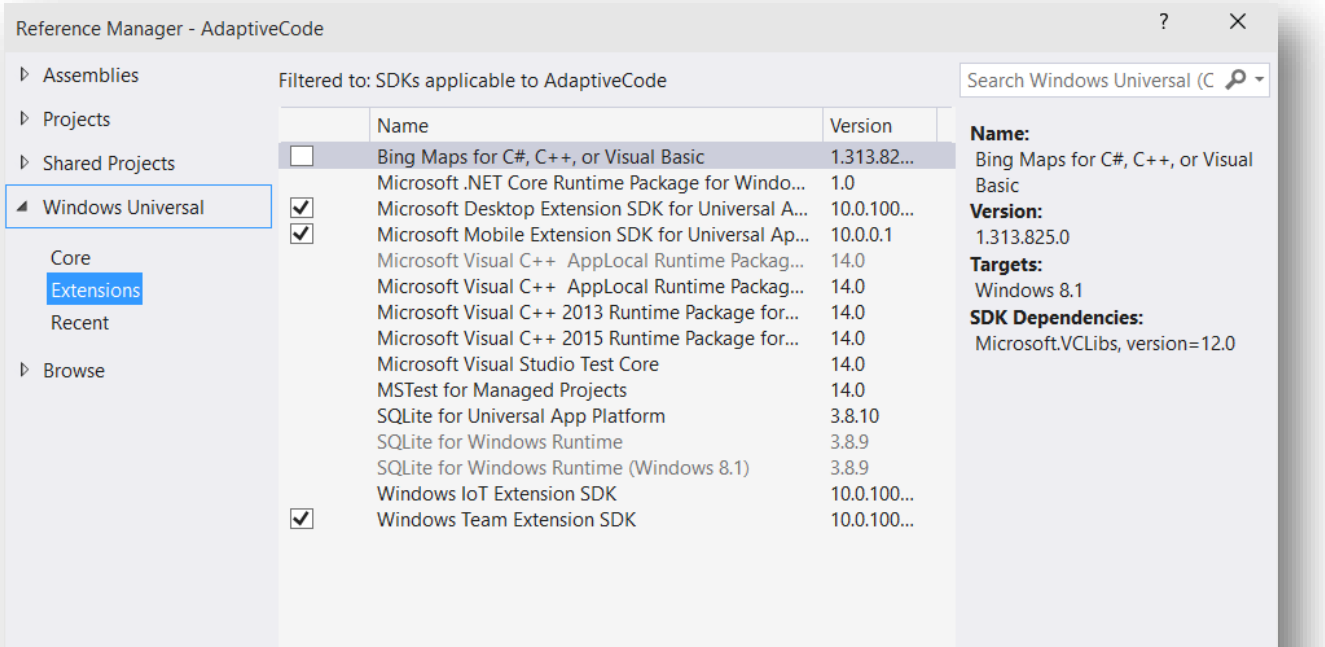
Platform extensions

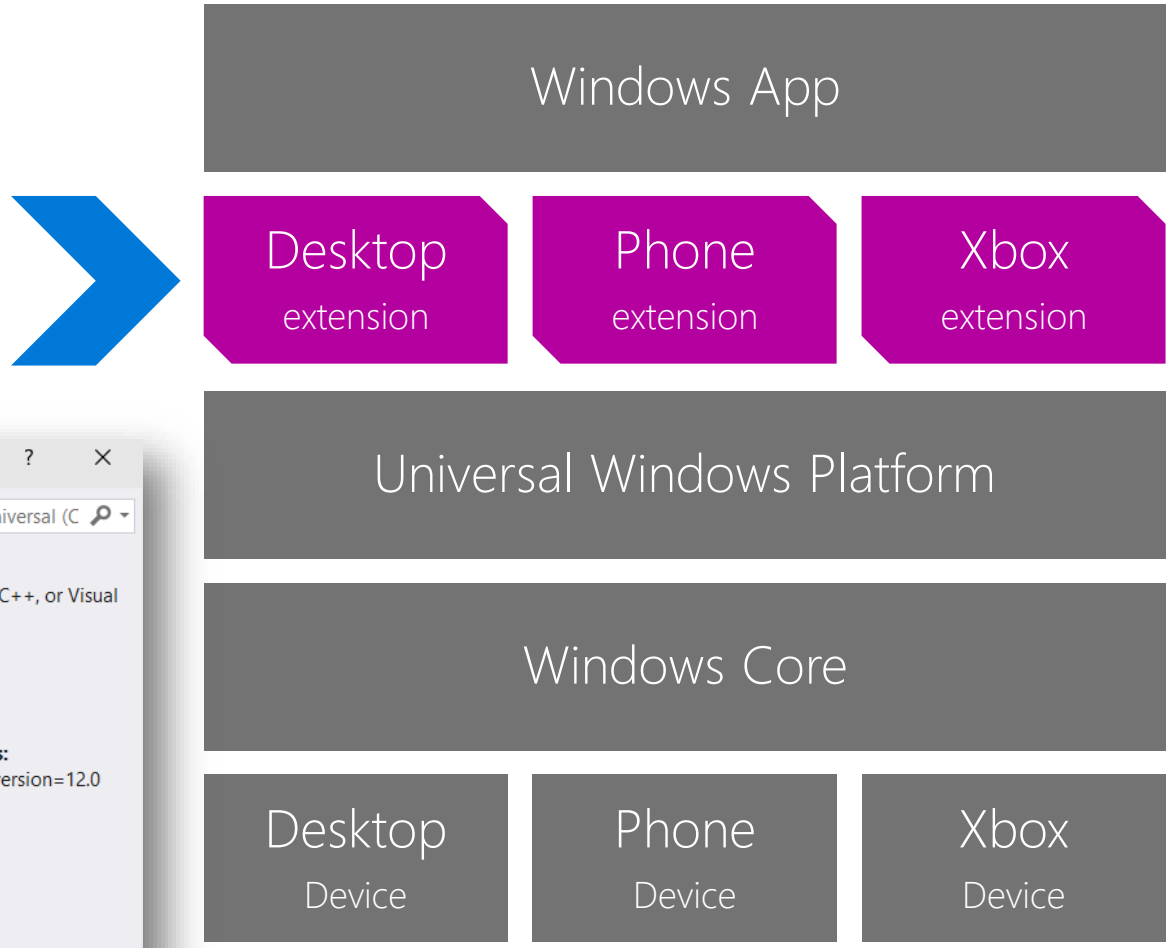
Device-specific API

Family-specific capabilities

Compatible across devices

Unique update cadence





Test capabilities at runtime

```
var api = "Windows.Phone.UI.Input.HardwareButtons";
if (Windows.Foundation.Metadata.ApiInformation.IsTypePresent(api))
{
   Windows.Phone.UI.Input.HardwareButtons.CameraPressed
        += CameraButtonPressed;
}
```

What about shared projects?

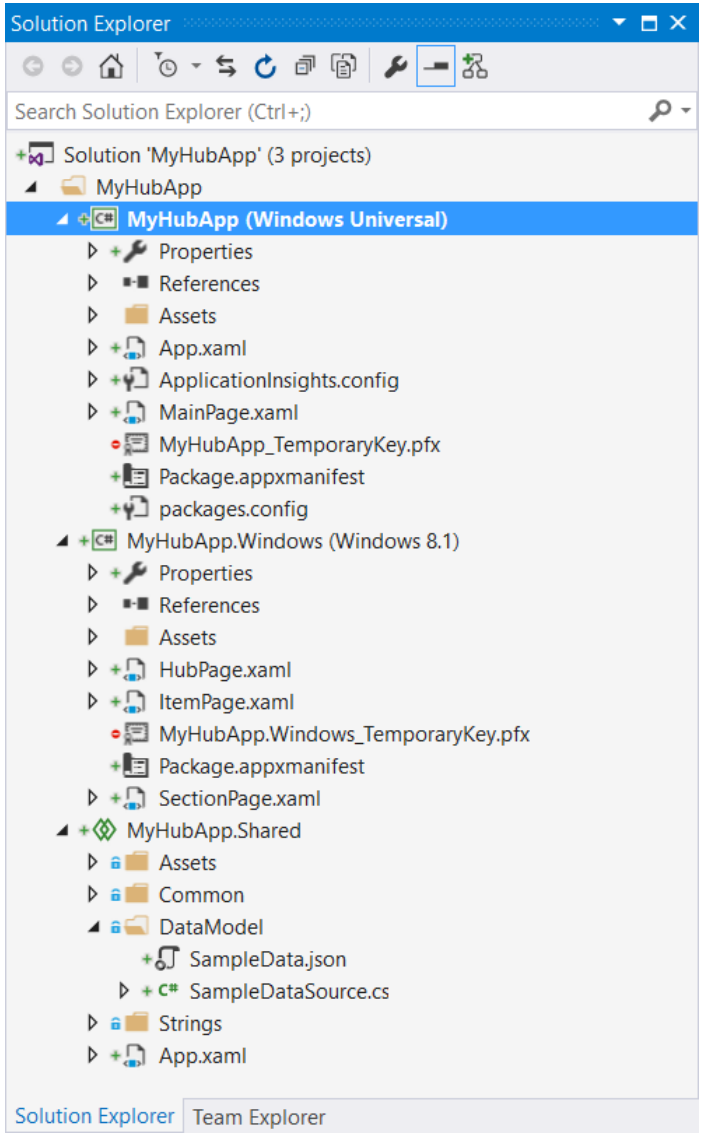
Shared Projects share code at the code level, pre-compilation

Share with Windows 8.1 projects, Xamarin projects, anything…

Still completely supported

`#if` Compilation directives used to conditionally include code by those projects referencing the Shared Project

1. WINDOWS_APP
2. WINDOWS_PHONE_APP
3. WINDOWS_UAP (new)



Use Adaptive Code in Shared too!

```
#if WINDOWS_PHONE_APP
     // Processing for Windows Phone 8.1 only
      Windows.Phone.UI.Input.HardwareButtons.CameraPressed
                += this.Camera_Pressed;

#elif WINDOWS_UAP
    // Processing for Windows UWP – Desktop AND Mobile
    if (Windows.Foundation.Metadata.ApiInformation
          .IsTypePresent("Windows.Phone.UI.Input.HardwareButtons";))
    {
       // But this is only for UWP AND Mobile
        Windows.Phone.UI.Input.HardwareButtons.CameraPressed
                 += this.Camera_Pressed;
    }
#endif
```

Compiled Binding

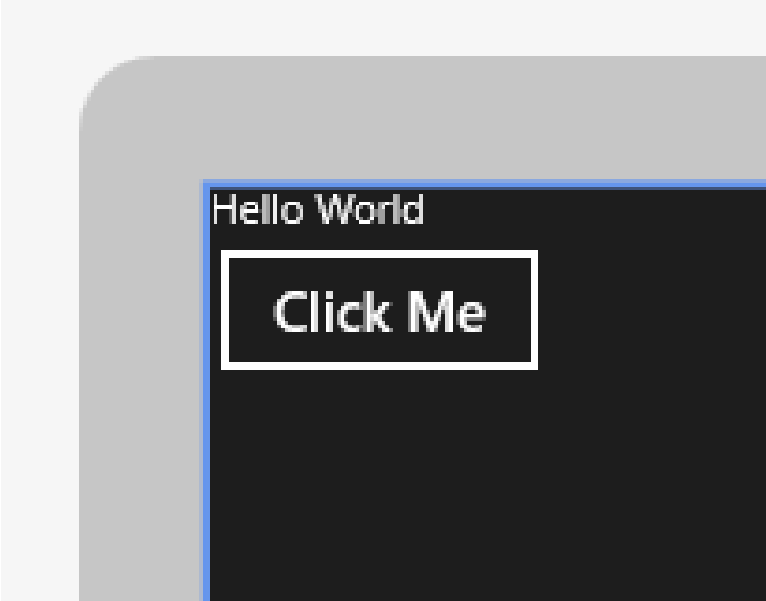
Windows

Data binding basics

Literal Data

You could hard code everything, but…

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <TextBlock>Hello World</TextBlock>
        <Button>Click Me</Button>
    </StackPanel>
</Grid>
```



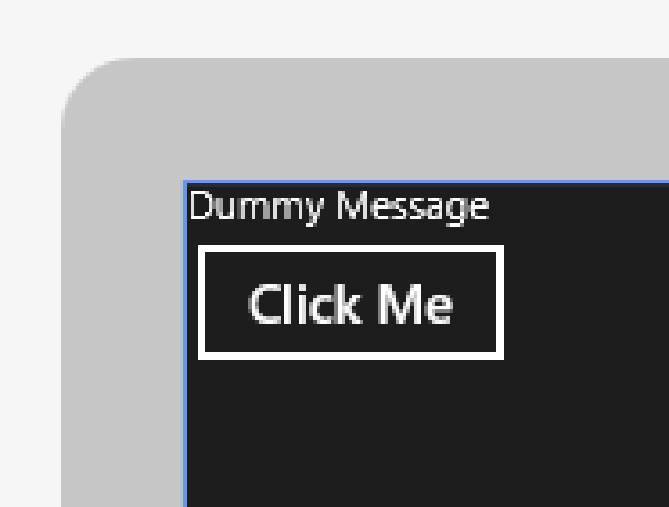
Not very dynamic.

Dynamic Data

Use data binding to connect to a data source

Typical data source would be a view model

```
<Page.Resources>
    <local:ViewModel x:Key="Vm"/>
</Page.Resources>

<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel DataContext="{StaticResource Vm}">
        <TextBlock Text="{Binding Message}"/>
        <Button Content="{Binding ButtonText}"/>
    </StackPanel>
</Grid>
```



Updating the UI

INotifyPropertyChanged

Implement in a view model class

Raised by View Model when property value changes

INotifyCollectionChanged

Implemented in *ObservableCollection<T>* and *ReadOnlyObservableCollection<T>*

Raised by the collection when the collection is modified

(also IObservableVector)

Compiled binding

Introducing compiled binding

Premise:

How do we keep the power of {Binding} but make it faster?

New mechanism for data binding in Xaml Apps

Heavy lifting is done at project build time rather than at runtime

Declarative bindings are converted into generated code behind

Eliminates need for slow runtime “reflection” operations

Code can be inspected and debugged

x:Bind bindings are validated at build time

x:Bind

Compiled binding

Bindings are committed at compile-time

Strongly-typed binding

Duck binding is not supported

Default mode is OneTime

OneWay and TwoWay are still available

Data source is Codebehind

Not DataContext anymore

Standard binding approaches

INotifyPropertyChanged, IObservableVector, INotifyCollectionChanged

Syntax

```
<TextBox Text="{Binding
      Converter
      ConverterLanguage
      ConverterParameter
      ElementName
      FallbackValue
      Mode
      Path
      RelativeSource
      Source
      TargetNullValue
      UpdateSourceTrigger}
```

```
<TextBox Text="{x:Bind
      Converter
      ConverterLanguage
      ConverterParameter
      ~~ElementName~~
      FallbackValue
      Mode
      Path
      ~~RelativeSource~~
      ~~Source~~
      TargetNullValue
      ~~UpdateSourceTrigger~~}
```

Data Templates

```
<ListView ItemsSource="{x:Bind ViewModel.Employees}">
    <ListView.ItemTemplate>
      <DataTemplate x:DataType="model:Employee">
          <Grid>
            <TextBlock Text="{x:Bind Name}"/>
          </Grid>
      </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Syntax differences

```
<ListView ItemsSource="{Binding Items}" Header="Classic" Grid.Column="0">
  <ListView.ItemTemplate>
    <DataTemplate>
      <TextBlock Text="{Binding Title}" />
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>

<ListView ItemsSource="{x:Bind ViewModel.Items}" xmlns:m="using:Blank3.Models"
          Header="Compiled" Grid.Column="1">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="m:TodoItem">
            <TextBlock Text="{x:Bind Title}" />
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Binding for Events

```
<Button Click="PokeEmployee">Poke Employee</Button>
```

```
<Button Click="{x:Bind Employee.Poke}">Poke Employee</Button>
```

Signature

Have no parameters - `void Poke()`

Match event parameters - `void Poke(object sender, RoutedEventArgs e)`

Match event base types - `void Poke(object sender, object e)`

Overloading is not supported

Because all events are eligible:

This may replace ICommand & EventToCommand

Note: this does not include parameter or CanExecute

Page.ViewModel

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        InitializeComponent();
        this.DataContextChanged += (s, e) =>
        {
            ViewModel = DataContext as ViewModels.MainPageViewModel;
        };
    }

    // strongly-typed view models enable x:bind
    public ViewModels.MainPageViewModel ViewModel { get; set; }
}
```

Page.ViewModel

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        InitializeComponent();
    }

    // strongly-typed view models enable x:bind
    public ViewModels.MainPageViewModel ViewModel => this.DataContext
                                                     as MainPageViewModel;
}
```

DANKE

Windows