Windows 10

Hands-on lab

Building an Adaptive UI

September 2015

Microsoft

CONTENTS

Overview

UWP apps may run on a number of device families that differ significantly in screen size and features. To give a great user experience on all devices, the design must adapt to the device. An adaptive UI can detect information about the device it is running on and deliver a layout based on the characteristics of that device.

An adaptive UI differs from a responsive UI, because it can deliver an individualized layout for each device family or screen size snap point. A responsive UI typically takes a single, flexible design and displays it gracefully on all devices. One drawback to responsive UI can be a slower load time – elements are loaded for all devices and resolutions even though they may not be needed. With adaptive UI, you can deliver a responsive design, but you also have the ability to deliver unique views to devices that have little in common with each other. For example, an Xbox view may be completely distinct from the desktop and mobile views for an app, because the device UI and interactions are so different.

In this lab, we will evolve a fixed layout into an adaptive UI. Initially, the UI will follow more responsive practices.

Objectives

This lab will show you how to:

- Evolve a fixed layout into an adaptive UI
- Create visual states to support narrow, medium, and wide screen widths
- Use a RelativePanel to reposition content in the different states
- Use Setters to better adapt styles to a smaller screen

System requirements

You must have the following to complete this lab:

- Microsoft Windows 10
- Microsoft Visual Studio 2015
- Windows 10 Mobile Emulator

Setup

You must perform the following steps to prepare your computer for this lab:

1. Install Microsoft Windows 10.
2. Install Microsoft Visual Studio 2015. Choose a custom install and ensure that the Universal Windows App Development Tools are selected from the optional features list.
3. Install the Windows 10 Mobile Emulator.

Exercises

This Hands-on lab includes the following exercises:

1. Transition to Adaptive UI
2. RelativePanels with Visual States

Estimated time to complete this lab: **30 to 45 minutes**.

Exercise 1: Transition to Adaptive UI

When starting to build an app, creating a fixed layout can be an expedient shortcut. Fixed designs are not ideal for UWP apps, however, which may display on many different screens. In this exercise, you will create a simple fixed layout to display an image and its associated metadata. You will then evolve the fixed layout into an adaptive one.

Task 1 – Create a blank Universal Windows app

We will begin by creating a project from the Blank App template.

1. In a new instance of Visual Studio 2015, choose **File > New> Project** to open the New Project dialog. Navigate to **Installed > Templates > Visual C# > Windows > Universal** and select the **Blank App (Universal Windows)** template.

2. Name your project **AdaptiveUI** and select the file system location where you will save your Hands-on Lab solutions. We have created a folder in our **C:** directory called **HOL** that you will see referenced in screenshots throughout the labs.

   Leave the options selected to **Create new solution** and **Create directory for solution**. You may deselect both **Add to source control** and **Show telemetry in the Windows Dev Center** if you don't wish to version your work or use Application Insights. Click **OK** to create the project.

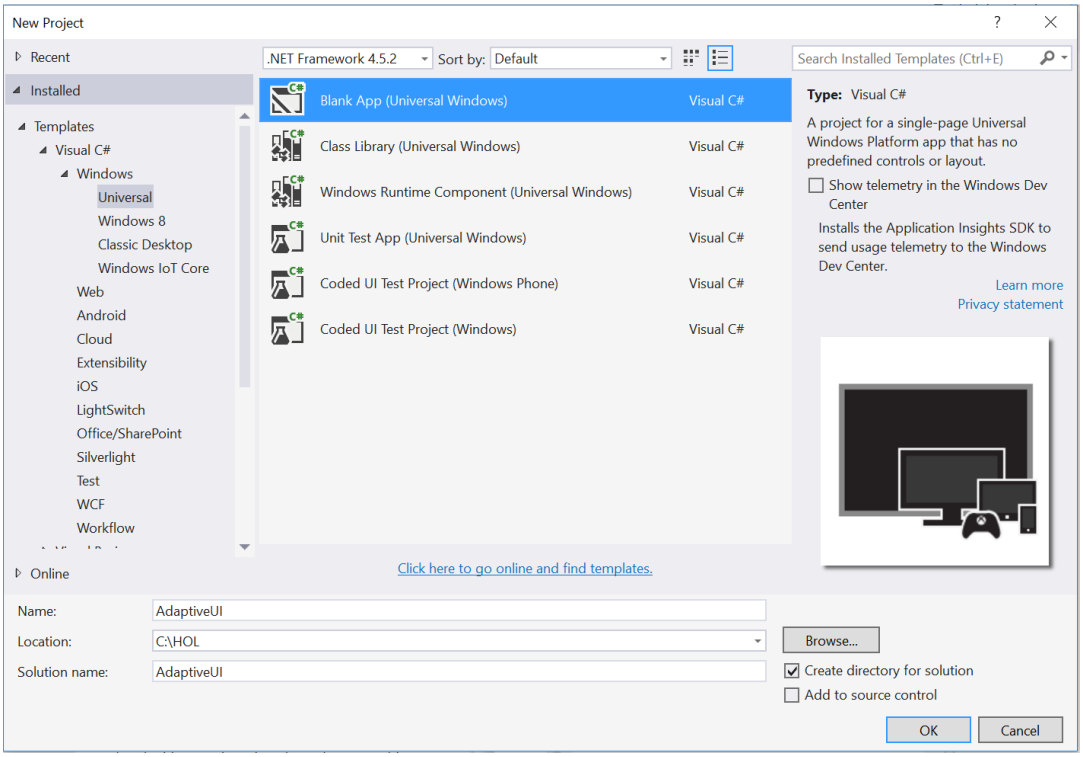


Figure 1

Create a new Blank App project in Visual Studio 2015.

3. Set your Solution Configuration to **Debug** and your Solution Platform to **x86**. Select **Local Machine** from the Debug Target dropdown menu.

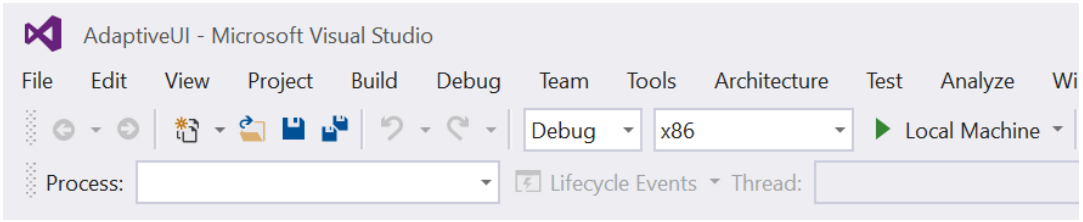


Figure 2

Configure your app to run on the Local Machine.

4. Build and run your app. You will see a blank app window with the frame rate counter enabled by default for debugging.

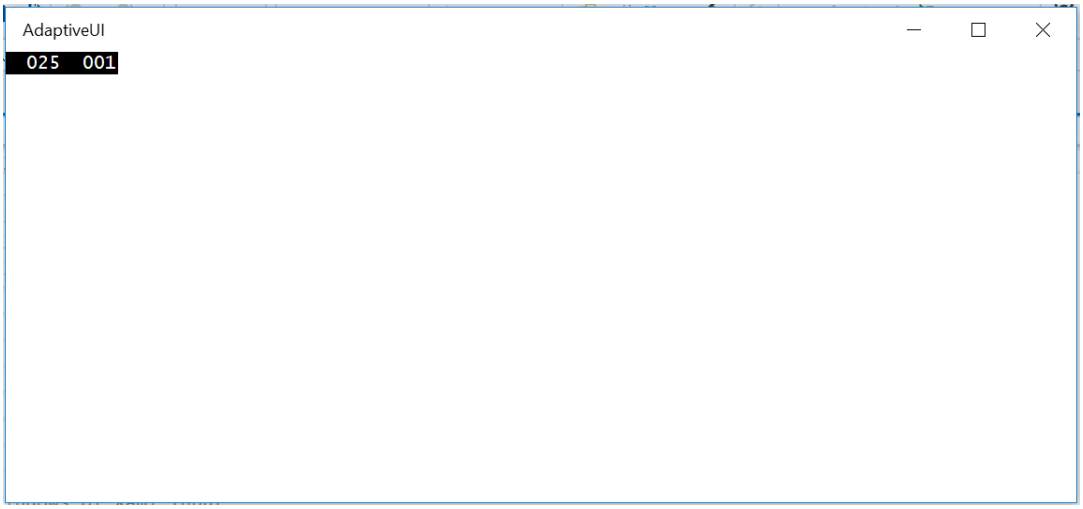


Figure 3

The blank universal app running in Desktop mode.

> **Note:** The frame rate counter is a debug tool that helps to monitor the performance of your app. It is useful for apps that require intensive graphics processing but unnecessary for the simple apps you will be creating in the Hands-on Labs.
>
> In the Blank App template, the preprocessor directive to enable or disable the frame rate counter is in **App.xaml.cs**. The frame rate counter may overlap or hide your app content if you leave it on. For the purposes of the Hands-on Labs, you may turn it off by setting **this.DebugSettings.EnableFrameRateCounter** to **false**.

5. Return to Visual Studio and stop debugging.

Task 2 – Create a fixed layout

As a starting point, we will create a basic fixed layout containing simple content. The content will consist of a large image and its associated metadata: username, avatar, image name, date, and description. We will format the content into two columns.

1. Open **MainPage.xaml**. Add two **ColumnDefinitions** to the Grid with fixed widths of **500 epx**.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
</Grid>
```

2. Right-click on the **Assets** folder and choose **Add > Existing Item**. Use the dialog to navigate to the **Lab Assets** folder at the location where you installed these Hands on Labs and select **airtime.jpg** and **avatar.jpg**. Add them to your project.

3. Return to **MainPage.xaml**. Add the **airtime** image to the first column.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
    <Image Grid.Column="0" Source="Assets/airtime.jpg" Stretch="UniformToFill"
      />
</Grid>
```

4. Add the avatar image and text metadata to the second column.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="500" />
        <ColumnDefinition Width="500" />
    </Grid.ColumnDefinitions>
    <Image Grid.Column="0" Source="Assets/airtime.jpg" Stretch="UniformToFill"
        />
    <StackPanel Grid.Column="1" Background="LightBlue">
        <Image Source="Assets/avatar.jpg" Width="100" Height="100"
            HorizontalAlignment="Left" />
        <TextBlock Text="phutureproof" />
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France." />
    </StackPanel>
</Grid>
```

5. Build and run your app. For certain aspect ratios, the content fits on the screen. Resize your window to see the behavior at smaller and larger window sizes. At smaller sizes, the second column is cut off, and at larger sizes, the content is surrounded by white space.

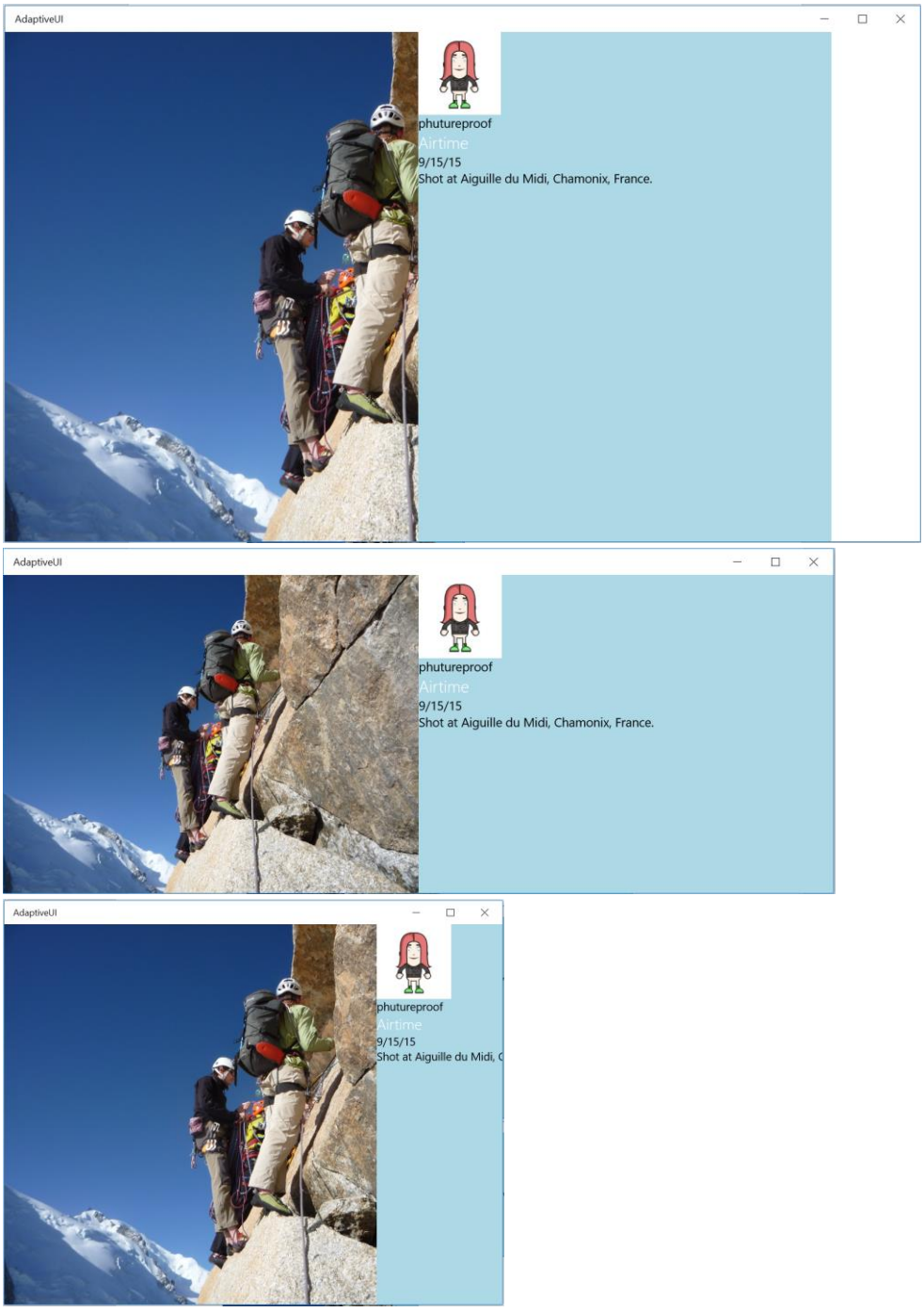


Figure 4

Fixed width columns at different window sizes.

6. Stop debugging and return to Visual Studio.

Task 3 – Adapt the layout for different screen sizes

Now that we've seen the drawbacks of a fixed layout UWP app, let's make it adaptive. We will follow the six principles of responsive design: Reposition, Resize, Reflow, Reveal, Replace, and Re-architect.

> **Note:** For more on responsive design in UWP apps, visit the Responsive Design 101 primer at https://msdn.microsoft.com/en-us/library/windows/apps/dn958435.aspx

1. Add row definitions to the grid. Remove the fixed width column attributes and give the columns x:Name attributes instead. We will use rows and columns to reposition the content in the grid.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition />
        <RowDefinition />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition x:Name="LeftCol" />
        <ColumnDefinition x:Name="RightCol" />
    </Grid.ColumnDefinitions>
```

2. Add Visual States based on the **MinWindowWidth** adaptive trigger.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="VisualStateGroup">
            <VisualState x:Name="VisualStateMin320">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="320"/>
                </VisualState.StateTriggers>
            </VisualState>
            <VisualState x:Name="VisualStateMin548">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="548"/>
                </VisualState.StateTriggers>
            </VisualState>
            <VisualState x:Name="VisualStateMin1024">
                <VisualState.StateTriggers>
                    <AdaptiveTrigger MinWindowWidth="1024"/>
                </VisualState.StateTriggers>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
```

> **Note:** Your UWP app will run on a spectrum of screen sizes. Snap points are a helpful tool when targeting groups of devices with similar, but not identical, characteristics. Some recommended snap

points are 320, 720, 1024 and 320, 548, 1024 depending on how your UI adapts through visual states. Which snap points you adopt in your app depends on your app and the user experience you are creating. You should always test your app on a variety of different screens.

Adaptive triggers in UWP include both **MinWindowWidth** and **MinWindowHeight**. We will use the MinWindowWidth trigger in this exercise.

3. Give **x:Name** directives to the hero image and the StackPanel containing the metadata content so that we can refer to them in VisualState Setters. Assign the hero image to the first row and column of the grid. The hero image will remain in the first column and row regardless of visual state, so we can leave it as a static setting. The metadata content will move depending on the screen width, however. Remove the **Grid.Column** assignment from the **Metadata** StackPanel,

XAML

```
<Image x:Name="Hero" Grid.Column="0" Grid.Row="0" Source="Assets/airtime.jpg"
    Stretch="UniformToFill" />
<StackPanel x:Name="Metadata" Background="LightBlue">
```

4. Add visual state setters to change the column behavior. For now, the two smaller visual states will share the same rules. We will differentiate them in the next exercise.

XAML

```
<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
```

```
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
    </VisualState.Setters>
</VisualState>
```

Note: The syntax used in the VisualState Setters to set properties such as (Grid.ColumnSpan) where the property is in brackets is the way to set attached properties of an element.

An attached property is a special kind of property inherited by children of certain elements, such as the Grid element, to enable the child to set properties on the parent. For instance, grid children can use attached properties to tell the parent in which Grid.Row and Grid.Column to position them within the grid structure.

It is best practice to set essential properties that define layout in each visual state. Although some values may carry over and not change from one visual state to the next as a user resizes the window, at app startup the window may be any size so the important properties must be present in every visual state to achieve the intended result.

5. Center the hero image with vertical and horizontal alignments. As the window is resized, it will hide and show the edges of the image depending on the aspect ratio. Centering the image makes it more likely that the subject of the image will remain in view.

XAML

```
<Image x:Name="Hero" Grid.Column="0" Grid.Row="0" Source="Assets/airtime.jpg"
    Stretch="UniformToFill" VerticalAlignment="Center"
    HorizontalAlignment="Center" />
```

6. Add text wrapping to the description field.

XAML

```
<TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France "
    TextWrapping="WrapWholeWords" />
```

7. Add padding to the Metadata panel.

XAML

```
<StackPanel x:Name="Metadata" Background="LightBlue" Padding="12">
```

8. Without a fixed width, the columns default to 50% width. Set the left column to 66.6% and the right column to 33.3% for the 1024 visual state to give more room to the image.

XAML

```
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
    </VisualState.Setters>
</VisualState>
```

Note: The asterisk in the width assignment tells the Grid that the width value is expressed as a weighted proportion of available space. Your content spans both columns in the smaller visual states, so it is not necessary to set width rules for the columns in those states.

9. Build and run your app. When you shrink the window, the metadata will move into the first column underneath the hero image. At larger sizes, the metadata will display in the second column.

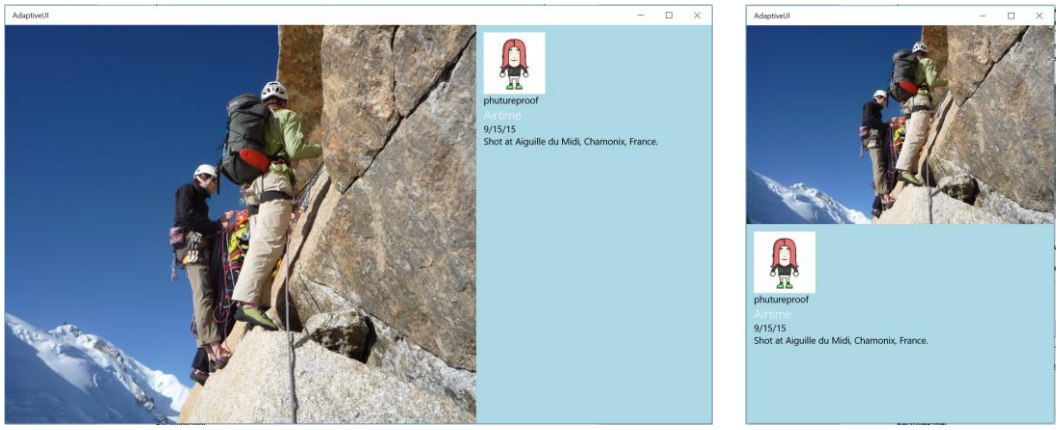


Figure 5

The metadata content moves below the image for smaller window sizes.

10. Stop debugging and return to Visual Studio.

Exercise 2: RelativePanels with Visual States

In this exercise, you will add a RelativePanel to handle semantically grouped content and further refine the responsiveness of the design.

Task 1 – Add the RelativePanel

The metadata content is grouped together in all states, which makes it a good candidate for a RelativePanel.

> **Note:** A RelativePanel allows you define the layout of its children elements in relation to each other. While those relationships may stay the same across visual states, you may also modify them per state. For more on the RelativePanel, visit https://msdn.microsoft.com/library/windows/apps/windows.ui.xaml.controls.relativepanel.aspx

1. Change the Metadata **StackPanel** to a **RelativePanel** and keep its attributes the same.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image Source="Assets/avatar.jpg" Width="100" Height="100"
HorizontalAlignment="Left" />
    <TextBlock Text="phutureproof" />
    <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
        Text="Airtime" />
    <TextBlock Text="9/15/15" />
    <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
        TextWrapping="WrapWholeWords" />
</RelativePanel>
```

2. Enclose the image title, date, and description **TextBlocks** in a **StackPanel**. These TextBlocks will remain stacked together, and it will be easier to position the StackPanel than each item individually.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image Source="Assets/avatar.jpg" Width="100" Height="100"
HorizontalAlignment="Left" />
    <TextBlock Text="phutureproof" />
    <StackPanel>
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
```

```
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

3. Give **x:Name** attributes to the immediate children of the **RelativePanel**. These items will be positioned relative to each other.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100" Height="100"
        HorizontalAlignment="Left" />
    <TextBlock x:Name="Username" Text="phutureproof" />
    <StackPanel x:Name="Description" >
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text="Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

4. Use **RelativePanel.Below** and **RelativePanel.AlignHorizontalCenterWith** to position the username relative to the avatar. This relationship will stay the same across visual states.

XAML

```
<RelativePanel x:Name="Metadata" Background="LightBlue" Padding="12">
    <Image x:Name="Avatar" Source="Assets/avatar.jpg" Width="100" Height="100"
        HorizontalAlignment="Left" />
    <TextBlock x:Name="Username" RelativePanel.Below="Avatar"
        RelativePanel.AlignHorizontalCenterWith="Avatar" Text="phutureproof"
        />
    <StackPanel x:Name="Description" >
        <TextBlock Foreground="White" FontSize="20" FontWeight="Light"
            Text="Airtime" />
        <TextBlock Text="9/15/15" />
        <TextBlock Text=" Shot at Aiguille du Midi, Chamonix, France."
            TextWrapping="WrapWholeWords" />
    </StackPanel>
</RelativePanel>
```

5. Add visual state setters to target the placement of the description relative to the avatar. For the **VisualStateMin1024** and **VisualStateMin548** states, there is room to display the description to the right of the avatar. For the **VisualStateMin320** state, position the description below the avatar. The description margin will also need to adapt to its placement.

XAML

```
<VisualState x:Name="VisualStateMin320">
```

```
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="" />
        <Setter Target="Description.(RelativePanel.Below)" Value="Username" />
        <Setter Target="Description.Margin " Value="0,12,0,0" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar" />
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar" />
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
    </VisualState.Setters>
```

```
</VisualState>
```

Note: In some visual states, the attached RelativePanel properties are set to empty string values. The empty string effectively erases the settings that may otherwise carry over from the other visual states as the window is resized and different visual states are applied. Without these setters, some unwanted relative positions may persist across states.

6. In addition to making layout changes, visual state setters can redefine styles like font size and margin for better display on different devices. Remove the font attribute from the TextBlock representing the image name and add an **x:Name** directive to allow you to target that element.

XAML

```
<TextBlock x:Name="ImageName" Foreground="White" FontWeight="Light"
    Text="Airtime" />
```

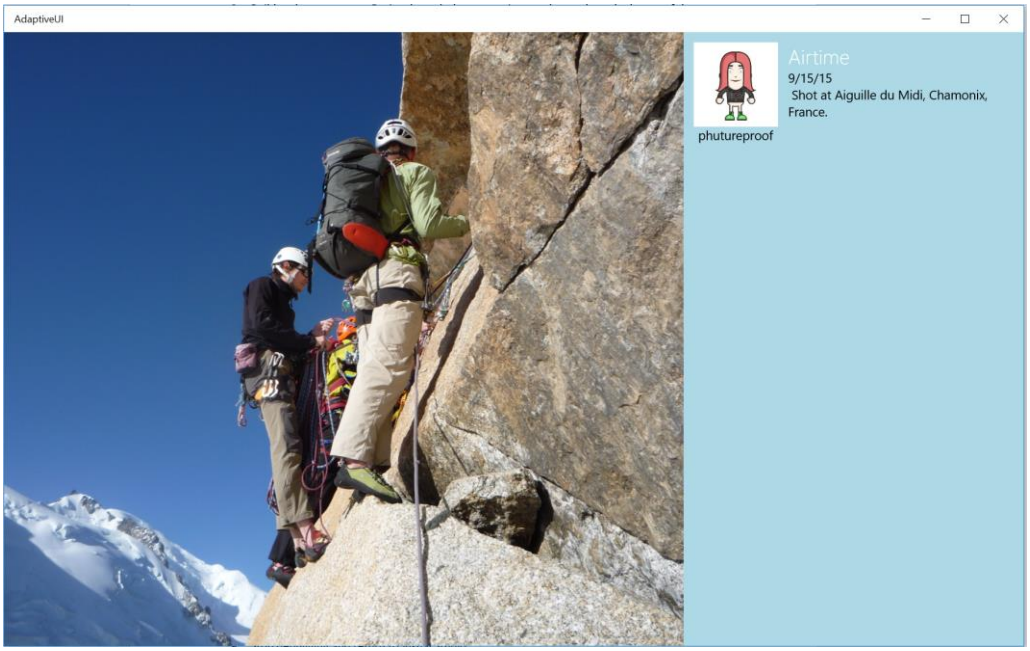
7. Create setters to customize the font size based on the visual state.

XAML

```
<VisualState x:Name="VisualStateMin320">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="320"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="" />
        <Setter Target="Description.(RelativePanel.Below)" Value="Username" />
        <Setter Target="Description.Margin " Value="0,12,0,0" />
        <Setter Target="ImageName.FontSize" Value="20" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin548">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="548"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.Column)" Value="0" />
        <Setter Target="Metadata.(Grid.Row)" Value="1" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="1" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar" />
```

```
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
        <Setter Target="ImageName.FontSize" Value="20" />
    </VisualState.Setters>
</VisualState>
<VisualState x:Name="VisualStateMin1024">
    <VisualState.StateTriggers>
        <AdaptiveTrigger MinWindowWidth="1024"/>
    </VisualState.StateTriggers>
    <VisualState.Setters>
        <Setter Target="Hero.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Hero.(Grid.RowSpan)" Value="2" />
        <Setter Target="Metadata.(Grid.Column)" Value="1" />
        <Setter Target="Metadata.(Grid.Row)" Value="0" />
        <Setter Target="Metadata.(Grid.ColumnSpan)" Value="1" />
        <Setter Target="Metadata.(Grid.RowSpan)" Value="2" />
        <Setter Target="LeftCol.Width" Value="2*" />
        <Setter Target="RightCol.Width" Value="1*" />
        <Setter Target="Description.(RelativePanel.RightOf)" Value="Avatar" />
        <Setter Target="Description.(RelativePanel.Below)" Value="" />
        <Setter Target="Description.Margin " Value="12,0,0,0" />
        <Setter Target="ImageName.FontSize" Value="24" />
    </VisualState.Setters>
</VisualState>
```

8. Build and run your app. Resize through the snap points to observe how the layout of the RelativePanel changes.



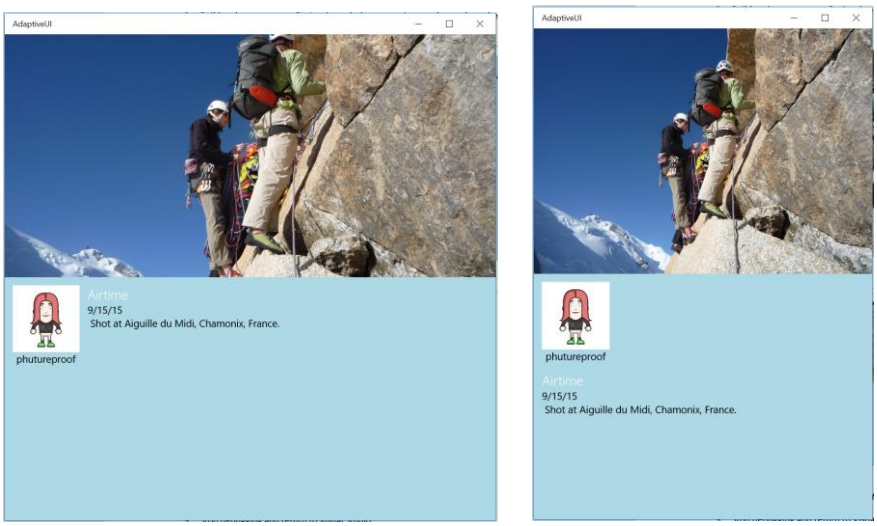


Figure 6

The RelativePanel content is repositioned for the different visual states.

9. Stop debugging and return to Visual Studio.

Summary

Adaptive UI is a essential part of UWP app design. In this lab, we followed the principles of responsive design to reposition and reflow content to better suit smaller screens.