

Hand­s-on lab

Lab 3 : Building an Adaptive UI

May 2015

Contents

[Overview 3](#_Toc420073842)

[Exercise 1: Adaptive Triggers 4](#_Toc420073843)

[Task 1 – Opening the starter project 4](#_Toc420073844)

[Task 2 – Building the initial UI 6](#_Toc420073845)

[Task 3 – Adding support for Adaptive UI 10](#_Toc420073846)

[Exercise 2: Custom Triggers 14](#_Toc420073847)

[Task 1 – Opening the starter project 14](#_Toc420073848)

[Task 2 – Add the orientation state trigger 15](#_Toc420073849)

[Task 3 – Network availability state trigger 20](#_Toc420073850)

[Task 4 – Device family and input type triggers 22](#_Toc420073851)

[Summary 26](#_Toc420073852)

Overview

In this lab we will begin to explore how we can create compelling user interfaces that are responsive to the device we are using and how it is oriented. We will show how some of the new capabilities in Windows 10 such as the **RelativePanel** and **AdaptiveTrigger** can be used to implement the desired layout. We will also show how you can create custom triggers and leverage them to update the UI.

To learn more about the design basics for Windows 10, you can view the resources [here](https://dev.windows.com/en-US/design/design-basics).

# Objectives

* 1. This lab will show you how to:
  + Build an adaptive UI
  + Respond to changes in the display height/width of your app
  + Build custom triggers to have further control over your ui.

# System requirements

* 1. You must have the following to complete this lab:
  + Microsoft Windows 10
  + Microsoft Visual Studio 2015

# Setup

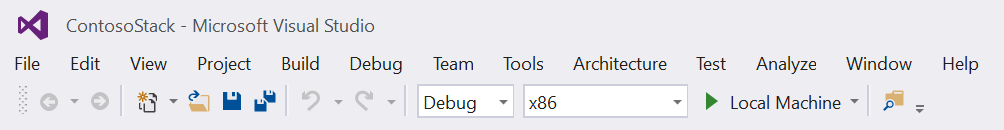
* 1. You must perform the following steps to prepare your computer for this lab:
  2. Install Microsoft Windows 10.
  3. Install Microsoft Visual Studio 2015.
  4. Install the Windows Universal Platform SDK.

# Exercises

* 1. This Hands-on lab includes the following exercises:
  2. Adaptive Triggers
  3. Custom Triggers
  4. Estimated time to complete this lab:  **45 to 60 minutes**.

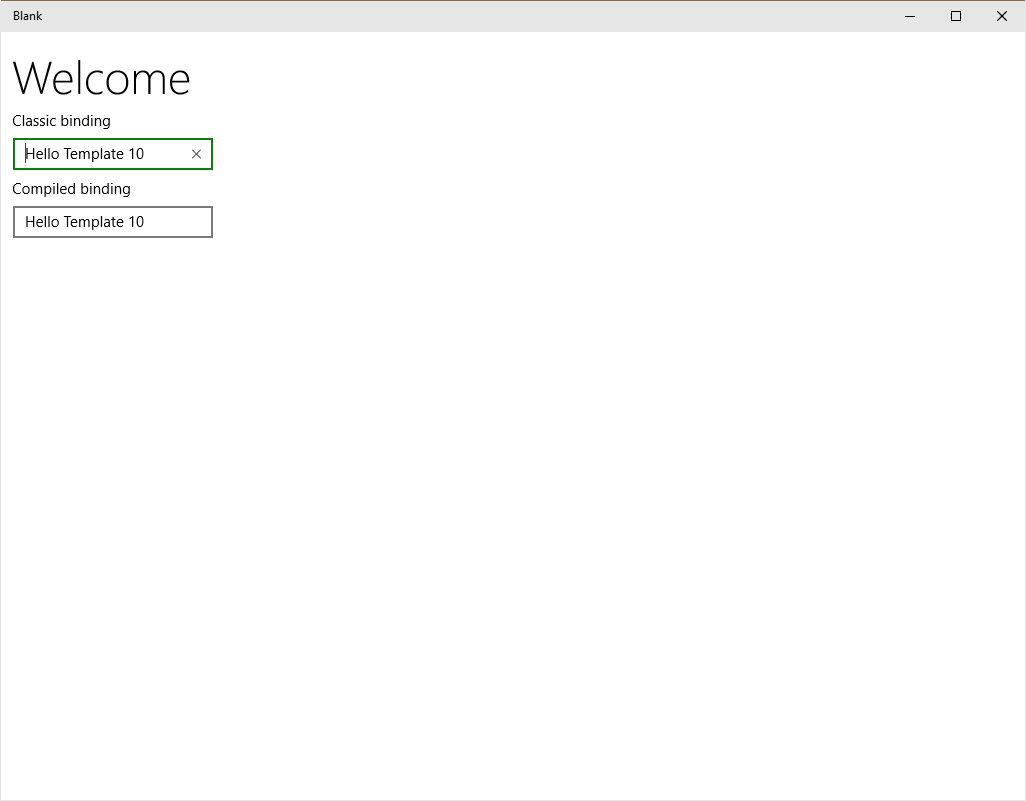
Exercise 1: Adaptive Triggers

1. This exercise will take you through the steps to implement an Adaptive UI using Adaptive triggers.
2. Task 1 – Opening the starter project
   1. The first step is to open the start project we have created for you.
3. Open the file location you installed the hands on labs into and navigate to the **\Lab 3\Solution\AdaptiveUI** folder and open **AdaptiveUI.sln.**
4. To prepare to build and run your app, use the Solution Configurations dropdown to choose the **Debug** configuration and use the Solution Platforms dropdown to target **x86 (Figure 8)**. To run the app on the local machine, select **Local Machine** from the drop-down list next to the Start Debugging button on the debugger Standard toolbar.
   1. Note:  is the Start Debugging button.



* + 1. **Figure 1**
    2. *Select options for building and running the app.*

1. Build and run the app:



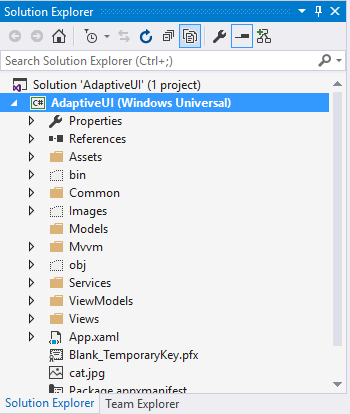
* + 1. **Figure 2**
    2. *The app running in desktop*

1. Return to Visual Studio and stop debugging.

Task 2 – Building the initial UI

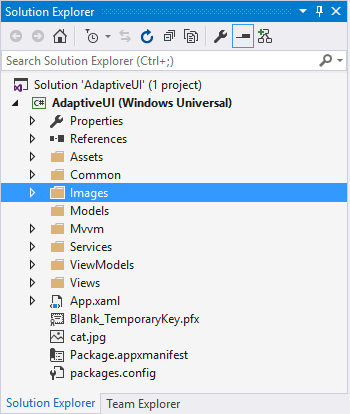
In this task, you will incorporate assets and build the initial, non-adaptive UI.

1. Let’s begin by adding an image to our project. In Visual Studio 2015, select the **Show All Files** option on the **Solution Explorer** toolbar. This option will refresh the **Solution Explorer** view with a list of all items in the **AdaptiveUI** project folder.



* + 1. **Figure 3**
    2. *The Solution Explorer view listing all files*

1. Right-click on the **Images** folder and select **Include In Project** from the context menu. The **Images** folder contains a single image. Deselect **Show All Files** in the **Solution Explorer** toolbar. Your **Solution Explorer** view will now look similar to this:



* + 1. **Figure 4**
    2. *The Solution Explorer view with the Images folder added*

1. You will now add the base line content to the **MainPageView.xaml** file. Note the use of the **RelativePanel** for positioning groups of elements.
   * 1. **XAML**

<Page

x:Class="AdaptiveUI.Views.MainPage"

xmlns:local="using:AdaptiveUI.Views"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:vm="using:AdaptiveUI.ViewModels"

mc:Ignorable="d">

<RelativePanel Background="{StaticResource ApplicationPageBackgroundThemeBrush}">

<TextBlock x:Name="Text00" Margin="12,12,12,0" Foreground="SteelBlue" Text="Screen size" />

<Image Source="ms-appx:///Images/cat.jpg" Margin="12,12,0,0" x:Name="MyImage" Stretch="Uniform" RelativePanel.Below="Text00" Width="225" RelativePanel.AlignLeftWith="Text00" />

<TextBlock FontSize="32" TextWrapping="Wrap" Foreground="Goldenrod" x:Name="MyText01" RelativePanel.RightOf="MyImage" Margin="12,12,12,0" RelativePanel.AlignTopWith="MyImage"> Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium</TextBlock>

<ScrollViewer x:Name="MyText02" VerticalScrollBarVisibility="Auto" RelativePanel.Below="MyText01" Margin="12" RelativePanel.AlignLeftWith="MyText01">

<RichTextBlock FontSize="18" TextWrapping="Wrap" Foreground="DimGray">

<Paragraph LineStackingStrategy="MaxHeight">

<InlineUIContainer>

<Grid Height="50" Width="50" Margin="0,0,12,0" Background="SteelBlue">

<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center" Foreground="White" FontSize="28">L</TextBlock>

</Grid>

</InlineUIContainer>

<Run>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.</Run>

</Paragraph>

<Paragraph/>

<Paragraph Margin="12,0,0,0" x:Name="SnappointText">

<Run Text="Snappoints" FontWeight="Black" />

<LineBreak />

<Run Text="&#x27A4; 0-500 epx" />

<LineBreak />

<Run Text="&#x27A4; 500-800 epx" />

<LineBreak />

<Run Text="&#x27A4; 800-1000 epx" />

<LineBreak />

<Run Text="&#x27A4; 1000+ epx" />

</Paragraph>

<Paragraph/>

<Paragraph>

<Run>Sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur? Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?</Run>

</Paragraph>

</RichTextBlock>

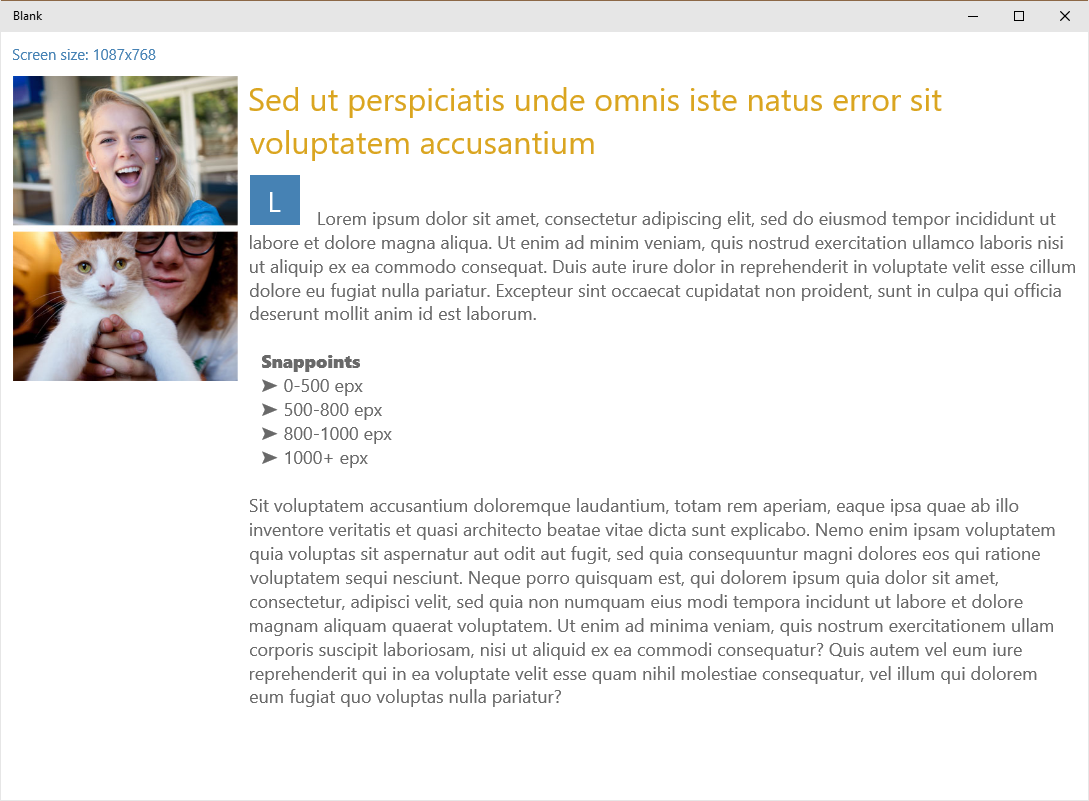
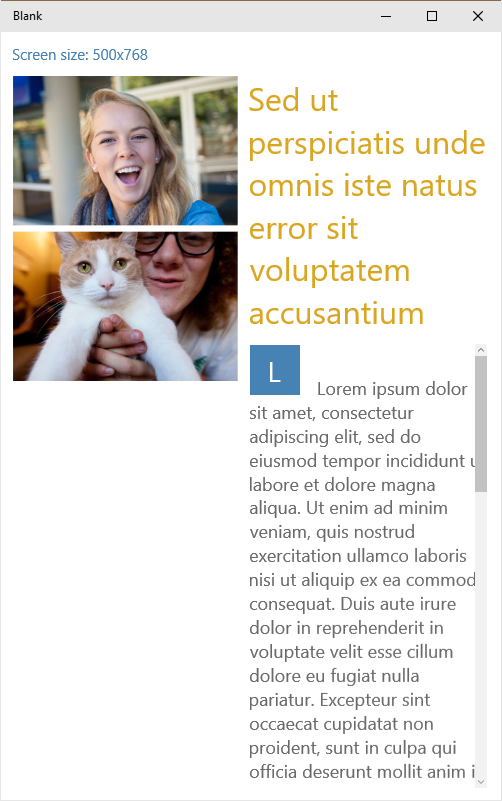
</ScrollViewer>

</RelativePanel>

</Page>

* 1. **Note:** The **RelativePanel** defines an area within which you can position and align child objects in relation to each other or the parent panel. Learn more here: <https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.relativepanel.aspx>

1. Build and run the app on the **Local Machine**.

* + 1. **Figure 5**
    2. *The desktop app resized*
  1. **Note:** the content is laid out reasonably well on a Desktop, with a **ScrollViewer** containing the large body of text. If you resize the app, the content on the right responds to the size change by filling the available space, however the app is not making the best use of the available space. Learn more about the **ScrollViewer** here: <https://msdn.microsoft.com/en-us/library/windows/apps/windows.ui.xaml.controls.scrollviewer.aspx>

1. Close the app and return to Visual Studio.
2. Select the Emulator device and build and run the app.

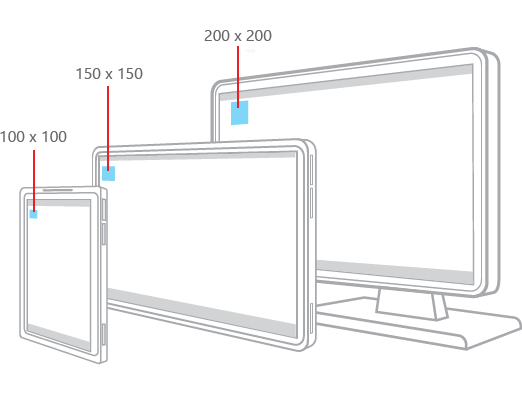
* + 1. **Figure 6**
    2. *The Phone Emulator views*
  1. **Note:** the UI does not handle the Phone portrait view at all well and the landscape view could make better use of the space.

1. Close the app and return to Visual Studio.
2. Task 3 – Adding support for Adaptive UI

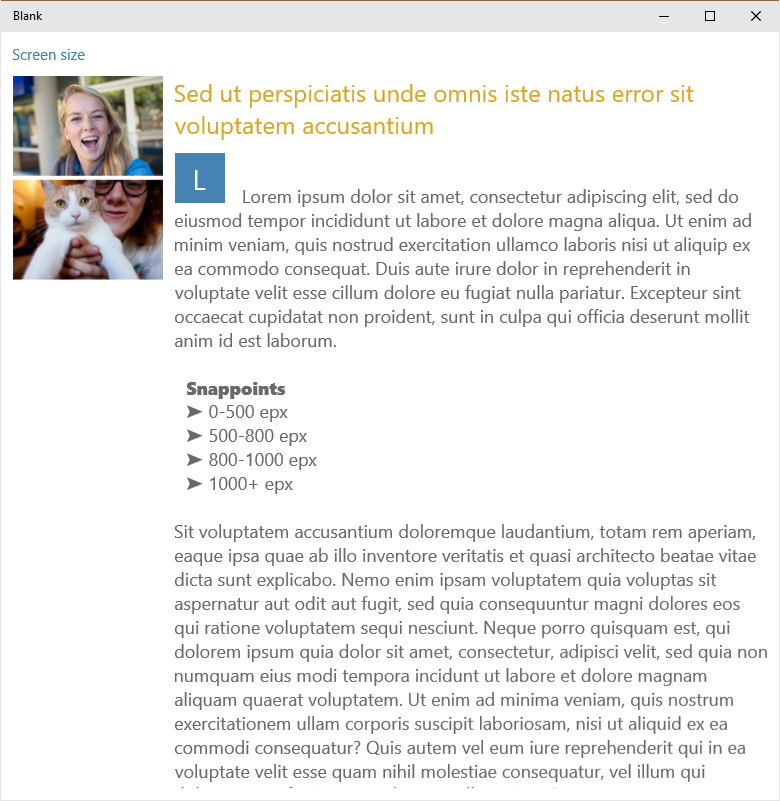
In this task, you will add XAML to update the layout based upon the view dimensions.

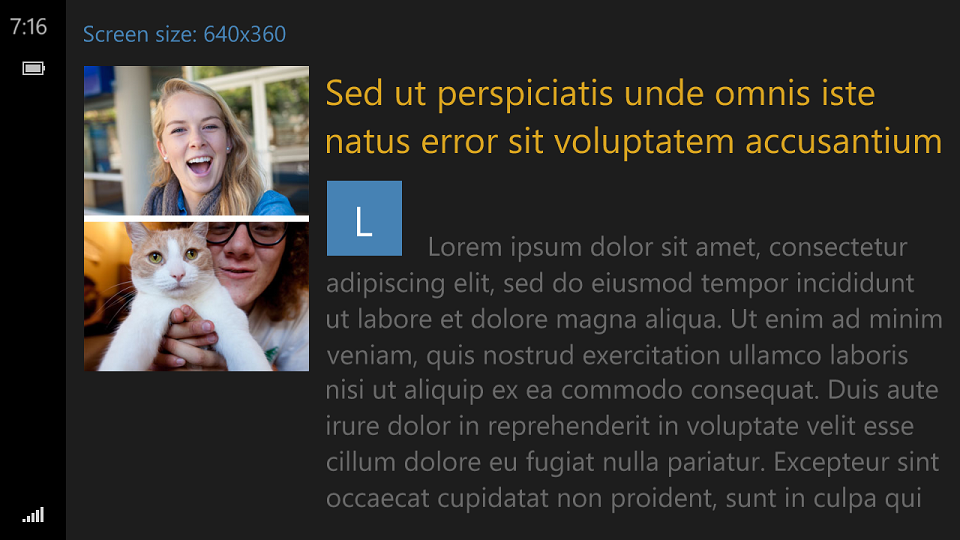
1. Open **MainPage.xaml** and insert the XAML shown below.
   * 1. **XAML**
     2. <Page x:Class="AdaptiveUI.Views.MainPage" xmlns:local="using:AdaptiveUI.Views" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:vm="using:AdaptiveUI.ViewModels" mc:Ignorable="d"> <RelativePanel Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
     3. <VisualStateManager.VisualStateGroups>
     4. <VisualStateGroup x:Name="VisualStateGroup">
     5. <VisualState x:Name="VisualState000min">
     6. <VisualState.StateTriggers>
     7. <AdaptiveTrigger MinWindowWidth="0" />
     8. </VisualState.StateTriggers>
     9. <VisualState.Setters>
     10. <Setter Target="MyText01.FontSize" Value="24" />
     11. <Setter Target="MyImage.Stretch" Value="UniformToFill" />
     12. <Setter Target="MyImage.Height" Value="150" />
     13. <Setter Target="MyText01.(RelativePanel.RightOf)" Value="{x:Null}" />
     14. <Setter Target="MyText01.(RelativePanel.AlignTopWith)" Value="{x:Null}" />
     15. <Setter Target="MyText01.(RelativePanel.Below)" Value="MyImage" />
     16. <Setter Target="MyText01.(RelativePanel.AlignLeftWith)" Value="MyImage" />
     17. </VisualState.Setters>
     18. </VisualState>
     19. <VisualState x:Name="VisualState500min">
     20. <VisualState.StateTriggers>
     21. <AdaptiveTrigger MinWindowWidth="501" />
     22. </VisualState.StateTriggers>
     23. <VisualState.Setters>
     24. <Setter Target="MyText01.FontSize" Value="24" />
     25. <Setter Target="MyText02.FontSize" Value="11" />
     26. <Setter Target="MyImage.Width" Value="150" />
     27. </VisualState.Setters>
     28. </VisualState>
     29. <VisualState x:Name="VisualState800min">
     30. <VisualState.StateTriggers>
     31. <AdaptiveTrigger MinWindowWidth="800" />
     32. </VisualState.StateTriggers>
     33. </VisualState>
     34. <VisualState x:Name="VisualState1000min">
     35. <VisualState.StateTriggers>
     36. <AdaptiveTrigger MinWindowWidth="1000" />
     37. </VisualState.StateTriggers>
     38. <VisualState.Setters>
     39. <Setter Target="MyImage.Width" Value="350" />
     40. </VisualState.Setters>
     41. </VisualState>
     42. </VisualStateGroup>
     43. </VisualStateManager.VisualStateGroups>
     44. <TextBlock x:Name="Text00" Margin="12,12,12,0" Foreground="SteelBlue" Text="Screen size" />
   1. **Note:** The XAML we have introduced leverages the **VisualStateManager** and the **AdaptiveTrigger** to modify properties on a number of UI elements. The **AdaptiveTrigger** is using the **MinWindowWidth** to change the layout at the following boundaries measured in **Effective pixels**:
   2. **0-500 epx**
   3. **500-800 epx**
   4. **800-1000 epx**
   5. **1000+ epx**
   6. **Effective pixels** enable you to focus on the actual perceived size of a UI element without having to worry about the pixel density or viewing distance of different devices. For example, when you design a 1'' by 1'' element, that element will appear to be approximately 1'' on all devices. On a very large screen with a high pixel density, the element might be 200 by 200 physical pixels, while on a smaller device like a phone, it might be 100 by 100 physical pixels.

Learn more here: <https://msdn.microsoft.com/en-us/library/windows/apps/dn958435.aspx>



* + 1. **Figure 7**
    2. *Effective Pixels scaling based upon perceived size*

1. Build and run the app on the **Local Machine**.
   * 1.   
     2. **Figure 8**
     3. *Different views for the various sizes of the desktop app*
   1. **Note:** As you resize the app, you will now notice 4 distinct layouts that are triggered as the width passes the boundaries defined in the **AdaptiveTrigger** XAML above.
2. Close the app and return to Visual Studio.
3. Select the Emulator device and build and run the app.

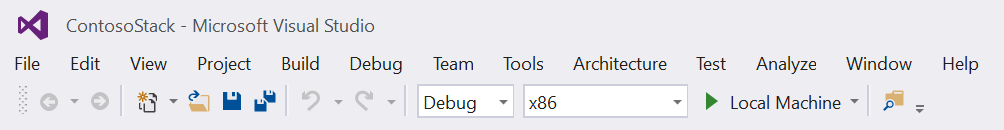
 

* + 1. **Figure 9**
    2. *Portrait and landscape views in the phone emulator.*
  1. **Note:** The layout of the content adapts as the device rotates between portrait and landscape.

1. Close the app and return to Visual Studio.

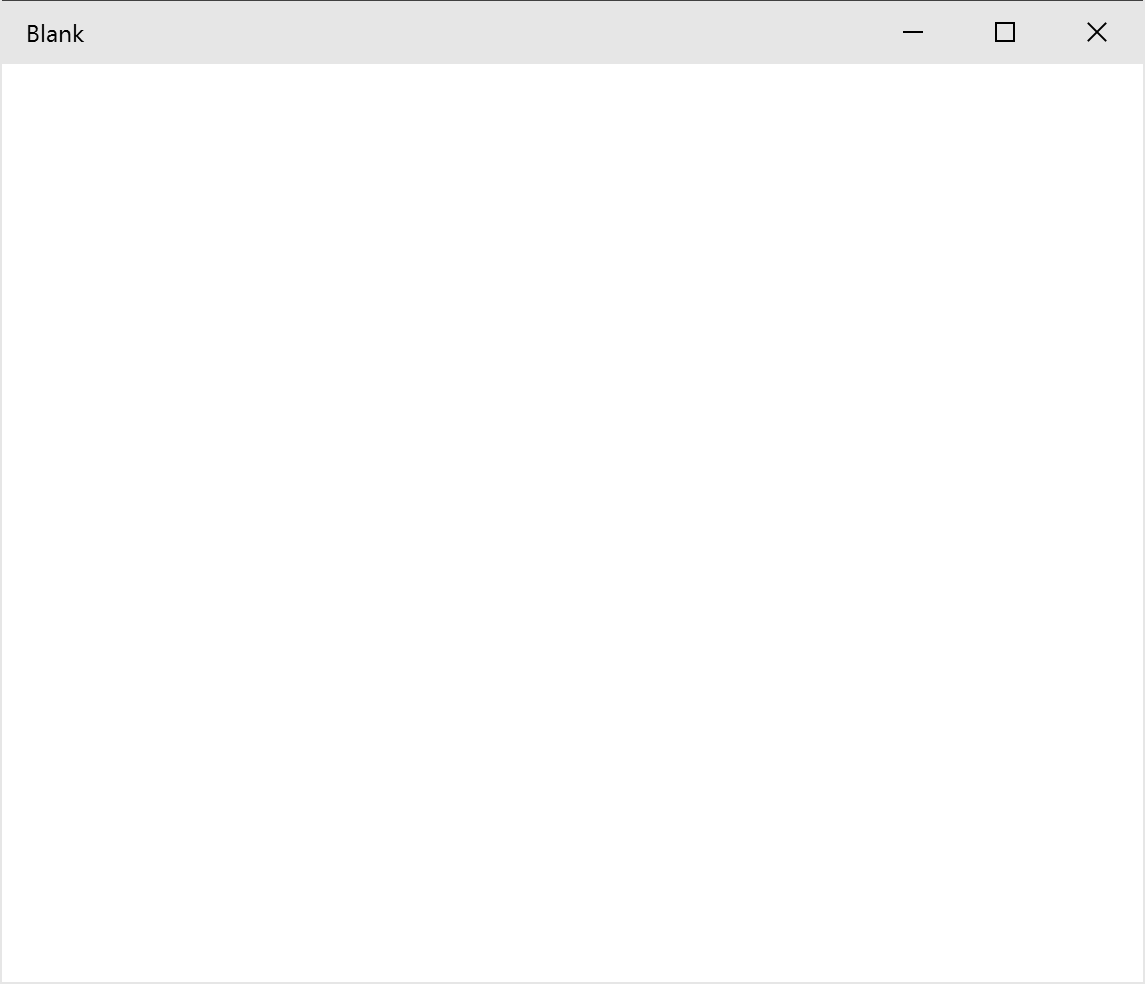
Exercise 2: Custom Triggers

1. In this exercise, you will implement custom triggers using the **Blank** template. You will test for network availability, orientation and input type changes.
2. Task 1 – Opening the starter project
   1. The first step is to open the start project we have created for you.
3. Open the file location where you installed the hands-on labs and navigate to the **\Lab 3\Solution\CustomTriggers** folder. Open **AdaptiveUI.sln.**
4. To prepare to build and run your app, use the Solution Configurations dropdown to choose the **Debug** configuration and use the Solution Platforms dropdown to target **x86 (Figure 8)**. To run the app on the local machine, select **Local Machine** from the drop-down list next to the Start Debugging button on the debugger Standard toolbar.
   1. Note:  is the Start Debugging button.



* + 1. **Figure 10**
    2. *Select options for building and running the app.*

1. Build and run the app. You should see a blank main page.



* + 1. **Figure 11**
    2. *The app running in desktop*

1. Return to Visual Studio and stop debugging.

Task 2 – Add the orientation state trigger

* 1. This task will show you how to check the current orientation of the device and change a visual state accordingly.

1. Create a new folder in the project where we will add our triggers. Right-click on the project and select **Add > New Folder.** Call the new folder **Triggers.**
2. Right click your new **Triggers** folder and select **Add > Class**. Name the class **OrientationTrigger.**
3. Delete the contents of the file and replace it with the code below.
   * 1. **C#**
   1. using CustomTriggers.Mvvm;
   2. using Windows.UI.ViewManagement;
   3. using Windows.UI.Xaml;
   4. namespace CustomTriggers.Triggers
   5. {
   6. public class OrientationTrigger : StateTriggerBase  
       {
   7. public OrientationTrigger()
   8. {
   9. }
   10. }  
       }
4. Now you have defined your trigger, but you need to make it do something. You need to create a function to determine if the orientation is landscape or portrait. Add the following function after the constructor.
   * 1. **C#**

private void CalculateState()  
{  
 var currentOrientation = ApplicationViewOrientation.Landscape;

var window = Window.Current;

currentOrientation = window.Bounds.Width >= window.Bounds.Height ? ApplicationViewOrientation.Landscape : ApplicationViewOrientation.Portrait;

SetActive(currentOrientation == orientation);

}   
  
private ApplicationViewOrientation orientation;

public ApplicationViewOrientation Orientation

{

get { return orientation; }

set

{

if (orientation != value)

{

orientation = value;

CalculateState();

}

}

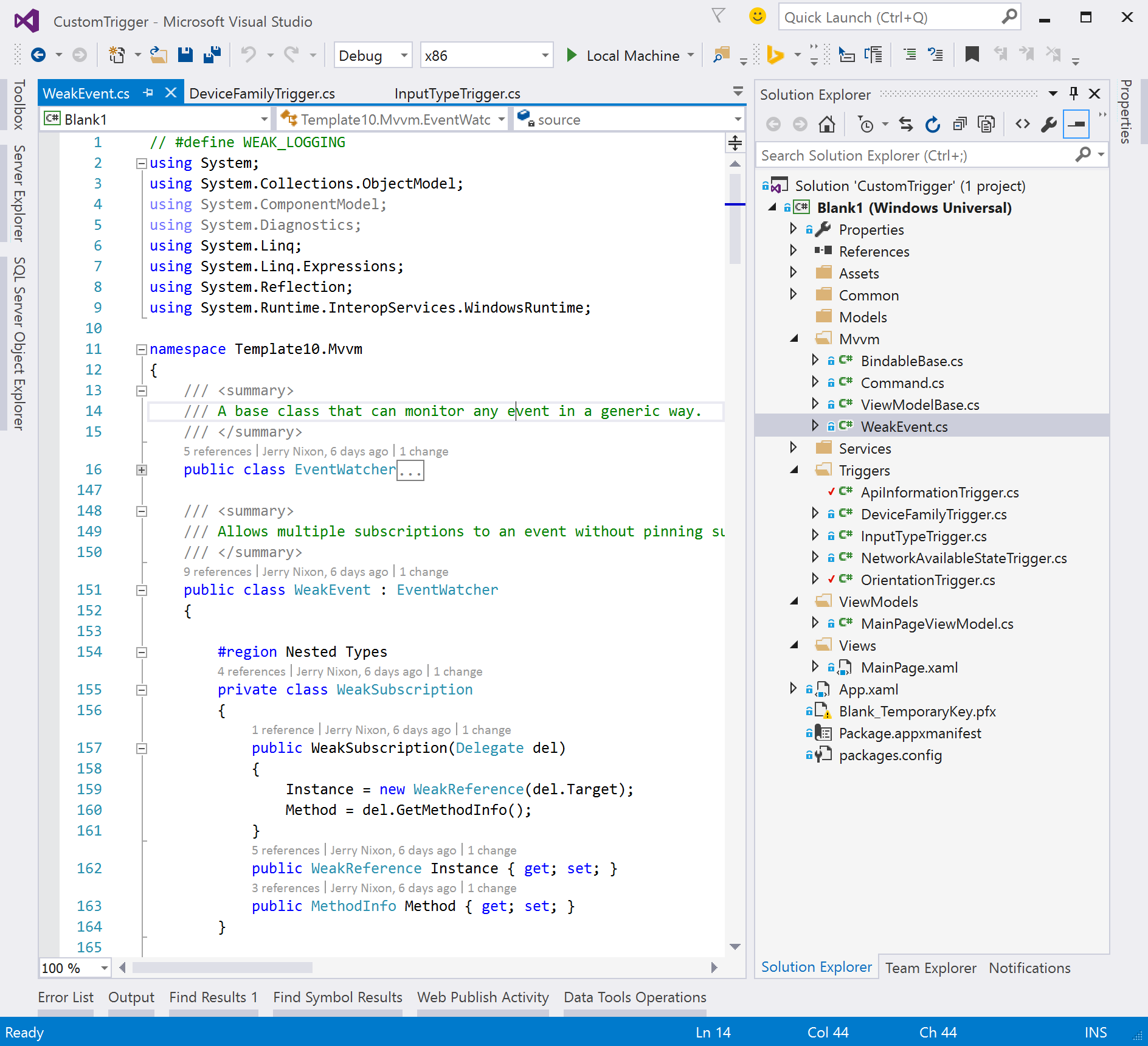
}

1. This function allows us to determine the orientation of the device, but we still need to update our *Orientation* parameter when the device is rotated. To do that, add the following lines to the constructor. This code adds an event handler to listen for the *SizeChanged* event on the window and calls our *CalculateState* function whenever that happens.  
   * 1. **C#**

var win = Window.Current;

WeakEvent.Subscribe<WindowSizeChangedEventHandler>(win, nameof(win.SizeChanged), (sender, e) => CalculateState());

CalculateState();

1. This code uses a class called **WeakEvent** in order to provide a weak reference to the *SizeChanged* event. You will need to add this class to your code. Right-click on your **Mvvm** folder and choose **Add > Existing Item**. Navigate to the \**Assets\Lab 3** folder in your Hands-on labs directory and select **WeakEvent.cs**. Click **Add** to import the class into your project. The WeakEvent class will appear in your **Mvvm** folder.
   * 1.   
        **Figure 12**
     2. Add the WeakEvent class to the project.

Note: The **WeakEvent**class helps to avoid a common memory leak with even handlers. For more information, visit <https://msdn.microsoft.com/en-us/library/aa970850(v=vs.110).aspx>.

1. Now you have the ability to detect when the user has changed the orientation of their device, but you need a way to use this ability in XAML to change the view. Open **MainPage.xaml** and add the Triggers namespace.
   * 1. XAML
   1. xmlns:vm="using:CustomTriggers.ViewModels"
   2. xmlns:t="using:CustomTriggers.Triggers"
   3. mc:Ignorable="d">
2. Add the code for the VisualStateManager inside the grid.
   * 1. **C#**
   1. <VisualStateManager.VisualStateGroups>
   2. <VisualStateGroup x:Name="OrientationTriggerVisualStateGroup">
   3. <VisualState x:Name="VisualStateLandscape">
   4. <VisualState.StateTriggers>
   5. <t:OrientationTrigger Orientation="Landscape" />
   6. </VisualState.StateTriggers>
   7. <Storyboard>
   8. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlock">
   9. <DiscreteObjectKeyFrame KeyTime="0">
   10. <DiscreteObjectKeyFrame.Value>
   11. <Visibility>Collapsed</Visibility>
   12. </DiscreteObjectKeyFrame.Value>
   13. </DiscreteObjectKeyFrame>
   14. </ObjectAnimationUsingKeyFrames>
   15. </Storyboard>
   16. </VisualState>
   17. <VisualState x:Name="VisualStatePortrait">
   18. <VisualState.StateTriggers>
   19. <t:OrientationTrigger Orientation="Portrait" />
   20. </VisualState.StateTriggers>
   21. <Storyboard>
   22. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlock1">
   23. <DiscreteObjectKeyFrame KeyTime="0">
   24. <DiscreteObjectKeyFrame.Value>
   25. <Visibility>Collapsed</Visibility>
   26. </DiscreteObjectKeyFrame.Value>
   27. </DiscreteObjectKeyFrame>
   28. </ObjectAnimationUsingKeyFrames>
   29. </Storyboard>
   30. </VisualState>
   31. </VisualStateGroup>
   32. </VisualStateManager.VisualStateGroups>
   33. <ScrollViewer HorizontalScrollMode="Auto">
   34. <StackPanel Orientation="Horizontal">
   35. <StackPanel.Resources>
   36. <Style TargetType="StackPanel">
   37. <Setter Property="VerticalAlignment" Value="Top" />
   38. <Setter Property="Margin" Value="12,12" />
   39. </Style>
   40. </StackPanel.Resources>
   41. <StackPanel>
   42. <TextBlock FontWeight="Black">Orientation trigger</TextBlock>
   43. <TextBlock x:Name="textBlock1">Landscape</TextBlock>
   44. <TextBlock x:Name="textBlock">Portrait</TextBlock>
   45. </StackPanel>

</StackPanel>

</ScrollViewer>

1. Your code is ready to run. If you are on a tablet and change orientation, you can run the application and notice the text change from *Landscape* to *Portrait*. If you are using a desktop or laptop, change your play mode to *Simulator*. In the simulator, you can rotate the orientation of the device. You can also run this app on the phone or phone simulator to view orientation behavior.
   1. **Note:** You can also grab the window of the application and resize the window until it is taller than it is wide to trigger your *Portrait* view state.



Figure

The app running in desktop with landscape orientation.



Figure

The app running in desktop with portrait orientation.

Task 3 – Network availability state trigger

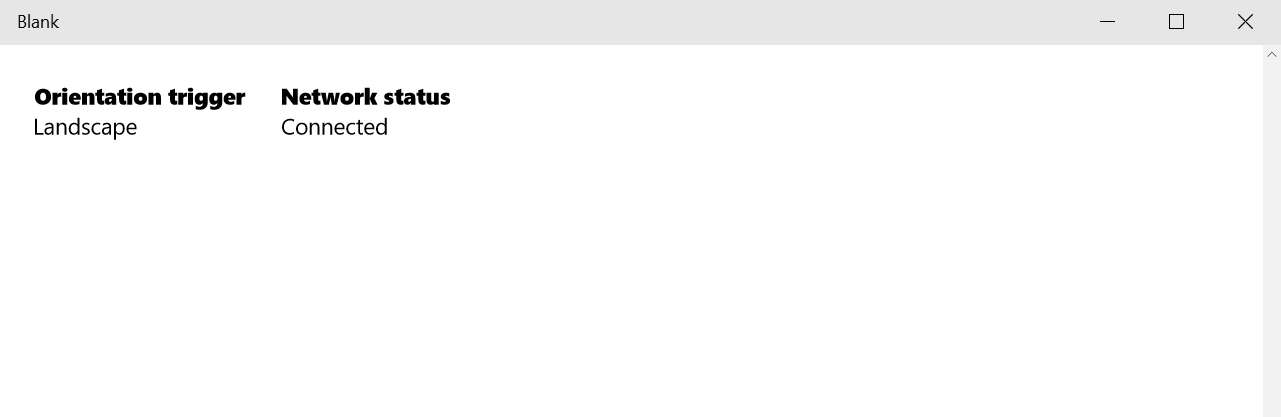
* 1. Next, you will create a trigger to notify you of changes in network availability.

1. Right-click on the **Triggers** folder and use **Add > Existing Item** to import the file **Assets\Lab 3\NetworkAvailableStateTrigger.cs**. This trigger monitors the *NetworkInformation* class for changes to the user’s internet access.
2. You will need to add some code into your viewto indicate the current network availability. Open **MainPage.xaml** and begin by adding code to display the current status.
   * 1. **C#**
   1. <StackPanel>
   2. <TextBlock FontWeight="Black">Orientation trigger</TextBlock>
   3. <TextBlock x:Name="textBlock1">Landscape</TextBlock>
   4. <TextBlock x:Name="textBlock">Portrait</TextBlock>
   5. </StackPanel>
   6. <StackPanel>
   7. <TextBlock FontWeight="Black">Network status</TextBlock>
   8. <TextBlock x:Name="textBlock8">Connected</TextBlock>
   9. <TextBlock x:Name="textBlock7">Disconnected</TextBlock>
   10. </StackPanel>
   11. </StackPanel>
3. While the code you added displays the status, it doesn’t change it. To change the status, you will need to add a trigger to hide and show textBlock8 and textBlock7 based on the current status of the connection. To add the trigger, you will need to create a new VisualStateGroup. Add this code after the *OrientationTriggerVisualStateGroup* you defined in the previous task.
   * 1. **C#**

<VisualStateGroup x:Name="NetworkAvailabilityVisualStateGroup">

</VisualStateGroup>

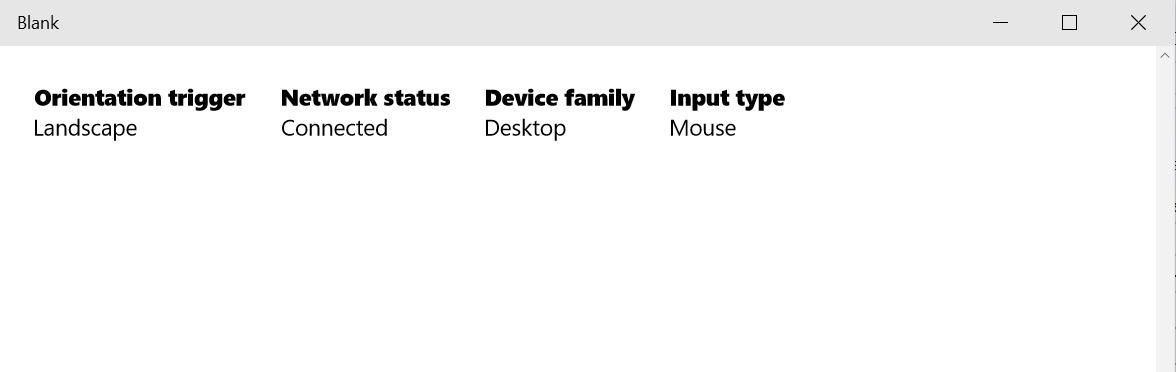
1. Once you have defined the VisualStateGroup, you can create the different states for network status. You’ll define two states: one for when the device is connected to the internet, and one for when it is disconnected.
   * 1. **C#**
   1. <VisualStateGroup x:Name="NetworkAvailabilityVisualStateGroup">
   2. <VisualState x:Name="VisualStateAvailable">
   3. <VisualState.StateTriggers>
   4. <t:NetworkAvailableStateTrigger ConnectionState="Connected" />
   5. </VisualState.StateTriggers>
   6. <Storyboard>
   7. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlock7">
   8. <DiscreteObjectKeyFrame KeyTime="0">
   9. <DiscreteObjectKeyFrame.Value>
   10. <Visibility>Collapsed</Visibility>
   11. </DiscreteObjectKeyFrame.Value>
   12. </DiscreteObjectKeyFrame>
   13. </ObjectAnimationUsingKeyFrames>
   14. </Storyboard>
   15. </VisualState>
   16. <VisualState x:Name="VisualStateUnavailable">
   17. <VisualState.StateTriggers>
   18. <t:NetworkAvailableStateTrigger ConnectionState="Disconnected" />
   19. </VisualState.StateTriggers>
   20. <Storyboard>
   21. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlock8">
   22. <DiscreteObjectKeyFrame KeyTime="0">
   23. <DiscreteObjectKeyFrame.Value>
   24. <Visibility>Collapsed</Visibility>
   25. </DiscreteObjectKeyFrame.Value>
   26. </DiscreteObjectKeyFrame>
   27. </ObjectAnimationUsingKeyFrames>
   28. </Storyboard>
   29. </VisualState>
   30. </VisualStateGroup>
2. Build and run your app. Notice that the network status changes from connected to disconnected if you go into airplane mode or disconnect your Wi-Fi.



Figure

The app running in desktop while connected to a network.

1. Task 4 – Device family and input type triggers
2. In this task, you will add two more triggers to the solution. The first will indicate if the device is *Desktop* vs *Mobile*. The second will show the current input type as either *Touch*, *Mouse* or *Pen.*
3. Right-click on the **Triggers** folder and use **Add > Existing Item** to import the files **Assets\Lab 3\DeviceFamilyTrigger.cs** and **Assets\Lab 3\InputTypeTrigger.cs**.
4. One the two triggers have been added, you can update **MainPage.xaml** to display the changing state of the triggers. Add the following *VisualStateGroups* and their associated states above the closing tag for the **VisualStateManager**.
   * 1. **C#**
   1. <VisualStateGroup x:Name="DeviceFamilyVisualStateGroup">
   2. <VisualState x:Name="VisualStateDesktop">
   3. <VisualState.StateTriggers>
   4. <t:DeviceFamilyTrigger DeviceFamily="Desktop" />
   5. </VisualState.StateTriggers>
   6. <Storyboard>
   7. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlock2">
   8. <DiscreteObjectKeyFrame KeyTime="0">
   9. <DiscreteObjectKeyFrame.Value>
   10. <Visibility>Collapsed</Visibility>
   11. </DiscreteObjectKeyFrame.Value>
   12. </DiscreteObjectKeyFrame>
   13. </ObjectAnimationUsingKeyFrames>
   14. </Storyboard>
   15. </VisualState>
   16. <VisualState x:Name="VisualStateMobile">
   17. <VisualState.StateTriggers>
   18. <t:DeviceFamilyTrigger DeviceFamily="Mobile" />
   19. </VisualState.StateTriggers>
   20. <Storyboard>
   21. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlock3">
   22. <DiscreteObjectKeyFrame KeyTime="0">
   23. <DiscreteObjectKeyFrame.Value>
   24. <Visibility>Collapsed</Visibility>
   25. </DiscreteObjectKeyFrame.Value>
   26. </DiscreteObjectKeyFrame>
   27. </ObjectAnimationUsingKeyFrames>
   28. </Storyboard>
   29. </VisualState>
   30. </VisualStateGroup>
   31. <VisualStateGroup x:Name="InputTypeVisualStateGroup1">
   32. <VisualState x:Name="VisualStateMouse">
   33. <VisualState.StateTriggers>
   34. <t:InputTypeTrigger PointerType="Mouse" />
   35. </VisualState.StateTriggers>
   36. <Storyboard>
   37. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlockMouse">
   38. <DiscreteObjectKeyFrame KeyTime="0">
   39. <DiscreteObjectKeyFrame.Value>
   40. <Visibility>Visible</Visibility>
   41. </DiscreteObjectKeyFrame.Value>
   42. </DiscreteObjectKeyFrame>
   43. </ObjectAnimationUsingKeyFrames>
   44. </Storyboard>
   45. </VisualState>
   46. </VisualStateGroup>
   47. <VisualStateGroup x:Name="InputTypeVisualStateGroup2">
   48. <VisualState x:Name="VisualStatePen">
   49. <VisualState.StateTriggers>
   50. <t:InputTypeTrigger PointerType="Pen" />
   51. </VisualState.StateTriggers>
   52. <Storyboard>
   53. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlockPen">
   54. <DiscreteObjectKeyFrame KeyTime="0">
   55. <DiscreteObjectKeyFrame.Value>
   56. <Visibility>Visible</Visibility>
   57. </DiscreteObjectKeyFrame.Value>
   58. </DiscreteObjectKeyFrame>
   59. </ObjectAnimationUsingKeyFrames>
   60. </Storyboard>
   61. </VisualState>
   62. </VisualStateGroup>
   63. <VisualStateGroup x:Name="InputTypeVisualStateGroup3">
   64. <VisualState x:Name="VisualStateTouch">
   65. <VisualState.StateTriggers>
   66. <t:InputTypeTrigger PointerType="Touch" />
   67. </VisualState.StateTriggers>
   68. <Storyboard>
   69. <ObjectAnimationUsingKeyFrames Storyboard.TargetProperty="(UIElement.Visibility)" Storyboard.TargetName="textBlockTouch">
   70. <DiscreteObjectKeyFrame KeyTime="0">
   71. <DiscreteObjectKeyFrame.Value>
   72. <Visibility>Visible</Visibility>
   73. </DiscreteObjectKeyFrame.Value>
   74. </DiscreteObjectKeyFrame>
   75. </ObjectAnimationUsingKeyFrames>
   76. </Storyboard>
   77. </VisualState>
   78. </VisualStateGroup>
5. Add stack panels to display the different possible values for *Device Family* and *Input Type*.
   * 1. **C#**
   1. <StackPanel>
   2. <TextBlock FontWeight="Black">Orientation trigger</TextBlock>
   3. <TextBlock x:Name="textBlock1">Landscape</TextBlock>
   4. <TextBlock x:Name="textBlock">Portrait</TextBlock>
   5. </StackPanel>
   6. <StackPanel>
   7. <TextBlock FontWeight="Black">Network status</TextBlock>
   8. <TextBlock x:Name="textBlock8">Connected</TextBlock>
   9. <TextBlock x:Name="textBlock7">Disconnected</TextBlock>
   10. </StackPanel>
   11. <StackPanel>
   12. <TextBlock FontWeight="Black">Device family</TextBlock>
   13. <TextBlock x:Name="textBlock3">Desktop</TextBlock>
   14. <TextBlock x:Name="textBlock2">Mobile</TextBlock>
   15. </StackPanel>
   16. <StackPanel>
   17. <TextBlock x:Name="textBlockInputType" FontWeight="Black">Input type</TextBlock>
   18. <TextBlock x:Name="textBlockMouse" Visibility="Collapsed">Mouse</TextBlock>
   19. <TextBlock x:Name="textBlockTouch" Visibility="Collapsed">Touch</TextBlock>
   20. <TextBlock x:Name="textBlockPen" Visibility="Collapsed">Pen</TextBlock>
   21. </StackPanel>
6. Build and run your application again. If you are using a touch-enabled screen, try tapping the application and watch the *InputType* trigger change. The *DeviceFamily* trigger should indicate what kind of device you are using.



Figure

The app running in desktop with mouse input.

1. Stop debugging and return to Visual Studio.

Summary

* 1. In this lab, we continued to use XAML controls in your application. Then we discovered how to tune our user interface to different screen sizes and form factors. Finally we walked through *Triggers* and how to use them to change how our UI reacts to different situations.
  2. Now we’ve got many of the tools to building great Windows Universal Apps. In the next lab, we’re going to dive even deeper. Let’s go!