Lesson 4 Guide:
UI Design with XAML

Table of Contents

Creating the XAML Interface 2

The XAML Code Editor 2

Pages 3

Examining XAML Controls (Part 1) 6

Grid 6

Text Controls - Overview 12

The TextBlock Control 12

The RichTextBlock Control 21

RichTextBoxOverflow 24

The TextBox Control 26

Changing the Text Display Programmatically 26

Other Text Controls 30

Image Control 30

Buttons 35

RadioButtons 36

CheckBoxes 38

ToggleSwitches 41

ToggleButtons 45

Creating and Applying Styles 49

Implicit Styles 49

Explicit Styles 54

Creating the XAML Interface

The XAML Code Editor

There are three formats that an element's XAML code may take.

Consider a Button element. An element can consist of a single tag with a concluding backslash to close it.

```
<Button x:Name="btnSMCC" Content="SMCC" Margin="140,100,0,0" Height="40" Width="100" />
```

Note that each attribute is specified in a property="value" format. The button above is named 'btnSMCC', is located at 140,100 in the parent and is 40x100 pixels in size.

The closing can be separated into a separate closing tag:

```
<Button x:Name="btnSMCC" Content="SMCC" Margin="140,100,0,0" Height="40" Width="100">
</Button>
```

The individual attributes as separate pair tags. Note that in this case, the values are not enclosed in quotes.

```
    <Button x:Name="btnSMCC">
        <Button.Content>SMCC</Button.Content>
        <Button.Margin>40,100,0,0</Button.Margin>
        <Button.Height>40</Button.Height>
        <Button.Width>100</Button.Width>
    </Button>
```

Where you want an element inside a parent container element, the parent tags are separated and they bookend the child elements. In the following block the parent Grid element houses a Button and a TextBlock element.

```
<Grid>
     <Button x:Name="btnSMCC" Content="SMCC" Margin="140,100,0,0"
                Height="40" Width="100" />
     <TextBlock x:Name="tbCampus" Content="South Mountain Community College"
                Margin="140,160,0,0" Height="40" Width="300" />
</Grid>
```

The XAML code can be entered directly in the XAML editor.

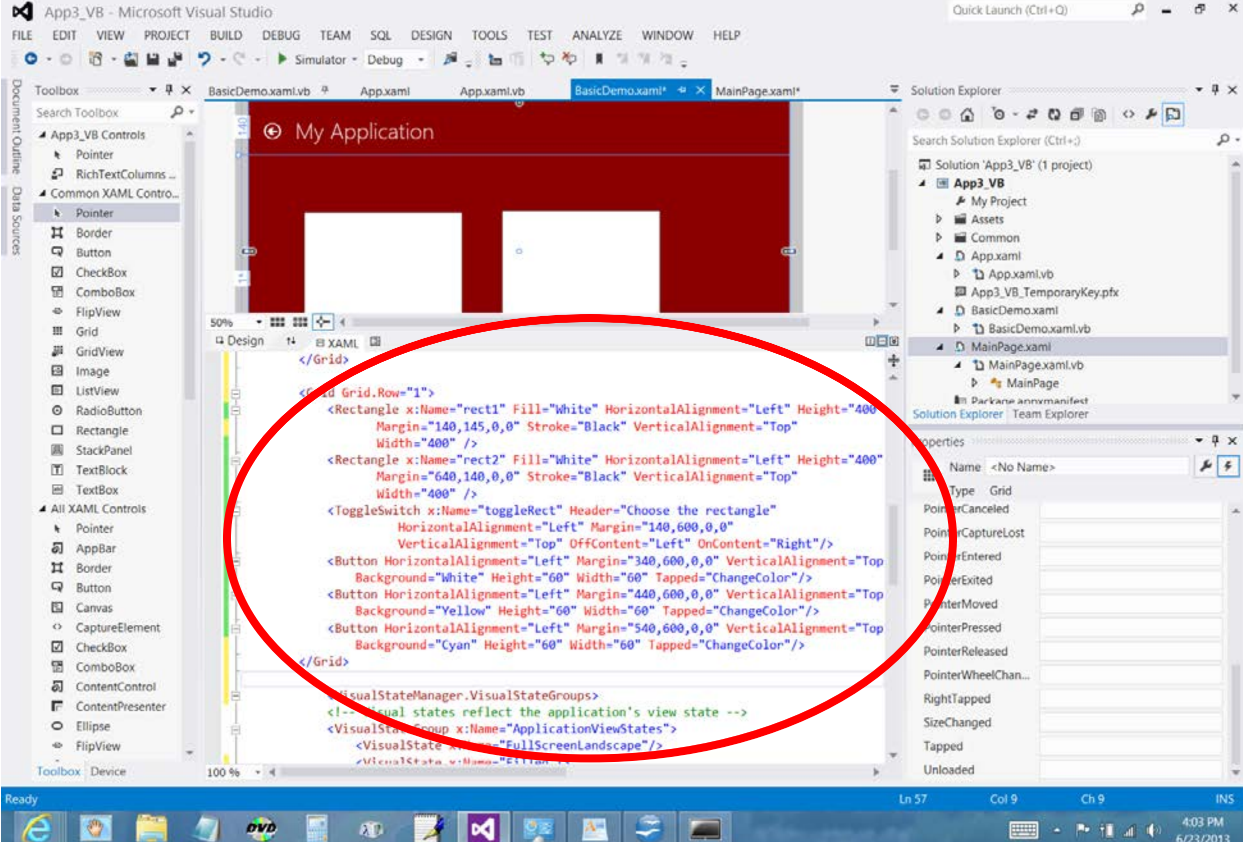


Figure 1 –The XAML Editor is underneath the visual Designer panel.

Pages

Windows Store apps allow you to navigate between multiple pages. You can add several types of pages, depending on which option is best suited to your needs.

To add pages, use the following steps:

1. Select **Add New Item** from the Project menu.
2. In the dialog, expand either the Visual C# or Visual Basic listings in the Installed pane as necessary and choose Windows Store App as the template type.
3. Choose the page of your choice (see table below), provide a name for the page, and click **Add**.

Page Type	Description	Thumbnail Image

Blank Page	As the name implies, this is a blank page with a <Canvas> element. No prebuilt capabilities. This is the starting page if you begin with a Blank App (XAML) project template.	
Basic Page	Similar to the Blank Page, but has a prebuilt Back button and page title, along with recommended margins and styles and a structure for handling various view states. This is not part of a project template, but can be added to a Blank template as a new starting point.	
Grouped Items Page	One of three pages included in the Grid App project template. This is the starting page containing grid view of grouped data items (broadest view). Tapping or clicking on a group navigates to the Group Detail Page for the data of that group.	
Group Detail Page	The middle detail view of a Grid App project template. This presents the data of a single group consisting of multiple items. On the left is information about the group as a whole and on the right are the various items of the group with additional (broad) info.	

Item Detail Page	This is the detailed view of items in a Grid App project. It presents detailed information about a specific item and is typically accessed by clicking an item listing in a Group Detail Page. Thus, the flow of drilling down from general to specific is from Grouped Items Page (broad) to Group Details Page (middle) to Item Detail Page (narrow).	
Items Page	This is the starting point of the Split App project template. It presents groups of items at the broadest level.	
Split Page	This is the detailed page of the Split App project template. It presents the broadest level of dual-layer data in a page split with two elements. On the left, there is a list of items. Details about the selected item is presented on the right.	

For each page you add, you will see in the Solution Explorer that a XAML file has been added to the project with an underlying .cs or .vb code page.

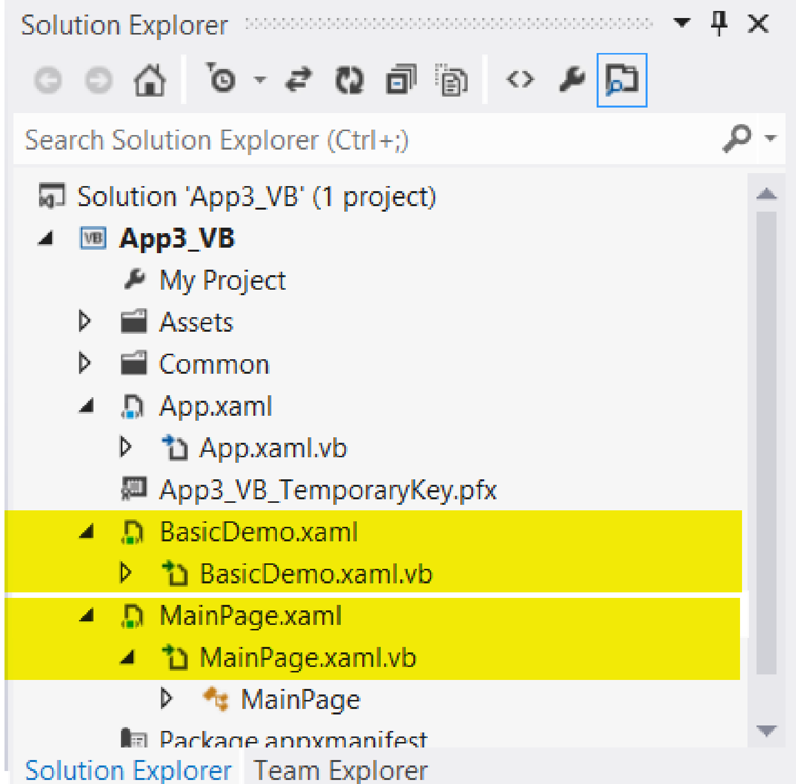


***Figure 2** – Each page in an app consists of a XAML interface definition and a code-behind Visual Basic or C# document.*

Examining XAML Controls (Part 1)

Grid

In choosing a Blank App (XAML) project template, the app in Lesson 3 began with the MainPage as a Blank page. But in reality, it had an element– the Grid element. Grids are often nested inside other grids to create very flexible layouts.

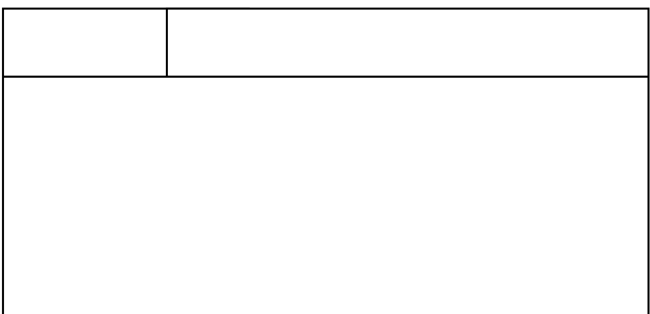
When we added a Basic page to the Lesson 3 app, it was predefined with a grid of two rows:

```
<Grid Style="{StaticResource LayoutRootStyle}">
     <Grid.RowDefinitions>
         <RowDefinition Height="140"/>
         <RowDefinition Height="*"/>
     </Grid.RowDefinitions>

     <Grid>
         <Grid.ColumnDefinitions>
             <ColumnDefinition Width="Auto"/>
             <ColumnDefinition Width="*"/>
          </Grid.ColumnDefinitions>
     </Grid>

</Grid>
```

This set up a grid of two rows. The first row (row 0) had a fixed height of 140 pixels, while the second row took up the remaining height of the display (using the asterisk wildcard).

The first row contains another grid of two columns. The left column (0) is autosized to display the content of the column. The right column takes up the remaining space (the * wildcard again).

You could display this graphically as:



Grids can be divided into rows and columns by defining RowDefinitions and/or ColumnDefinitions. The following block of code establishes a grid of four equal rows. For demonstration purposes, rectangles of different Fill colors are added to distinguish each of the rows. Note that the first row is row 0.

```
        <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
            <Grid.RowDefinitions>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
            </Grid.RowDefinitions>
            <Rectangle Grid.Row="0" Fill="Red"/>
            <Rectangle Grid.Row="1" Fill="Green"/>
            <Rectangle Grid.Row="2" Fill="Blue"/>
            <Rectangle Grid.Row="3" Fill="Yellow"/>
        </Grid>
```

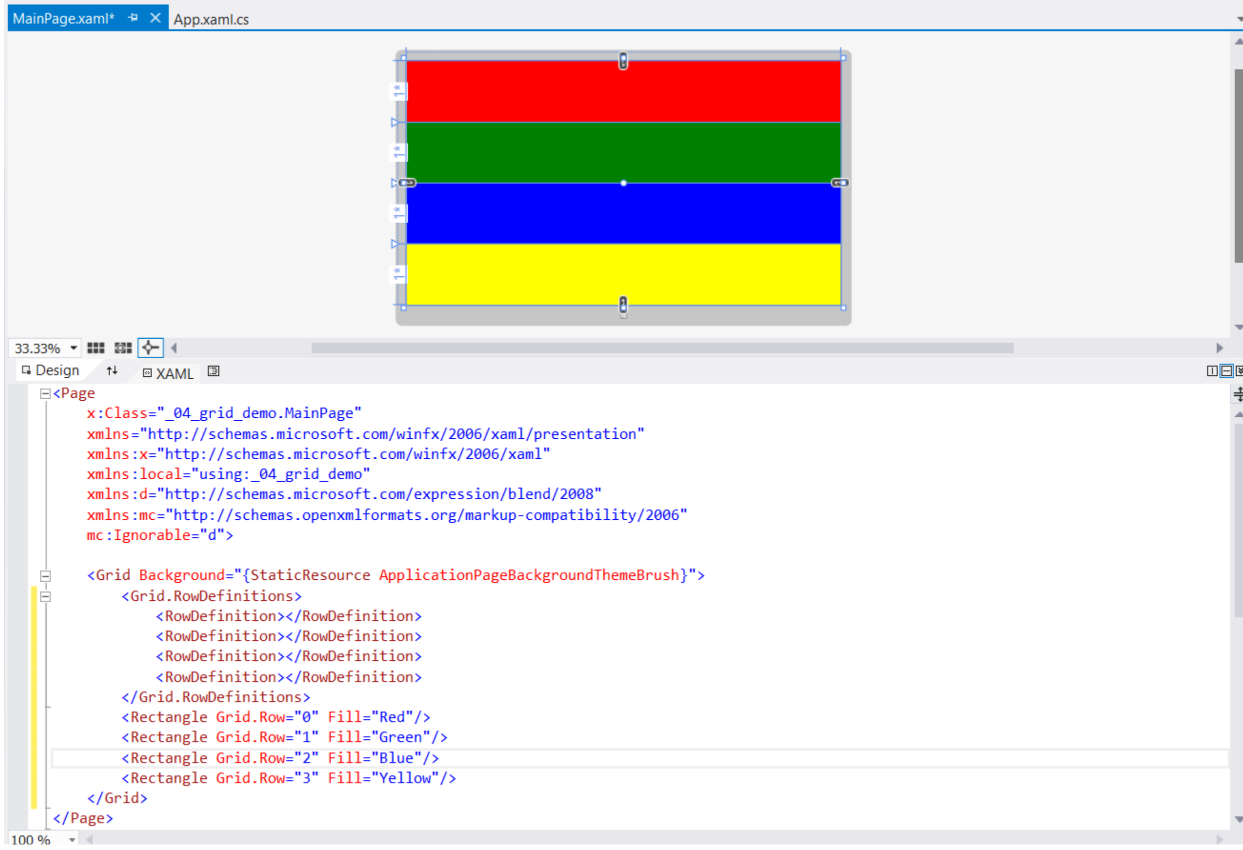


Figure 3 *– A Grid control filling the screen and consisting of four equally sized rows.*

The following code further divides the Grid element to have two columns in addition to the four rows, thus having eight divisions. Each resulting cell in this case is the same size. The last three Rectangle elements specify both a row value and a column value. The default on each is "0", thus the first four rectangle elements do not specify a Grid.Column value and column 0 is assumed. No rectangle is created in row 3, column 1, and the cell remains the page fill of black as specified by the applied Page style.

```
        <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
            <Grid.RowDefinitions>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Rectangle Grid.Row="0" Fill="Red"/>
            <Rectangle Grid.Row="1" Fill="Green"/>
            <Rectangle Grid.Row="2" Fill="Blue"/>
            <Rectangle Grid.Row="3" Fill="Yellow"/>
            <Rectangle Grid.Row="0" Grid.Column="1" Fill="Magenta"/>
            <Rectangle Grid.Row="1" Grid.Column="1" Fill="Cyan"/>
            <Rectangle Grid.Row="2" Grid.Column="1" Fill="White"/>
        </Grid>
```

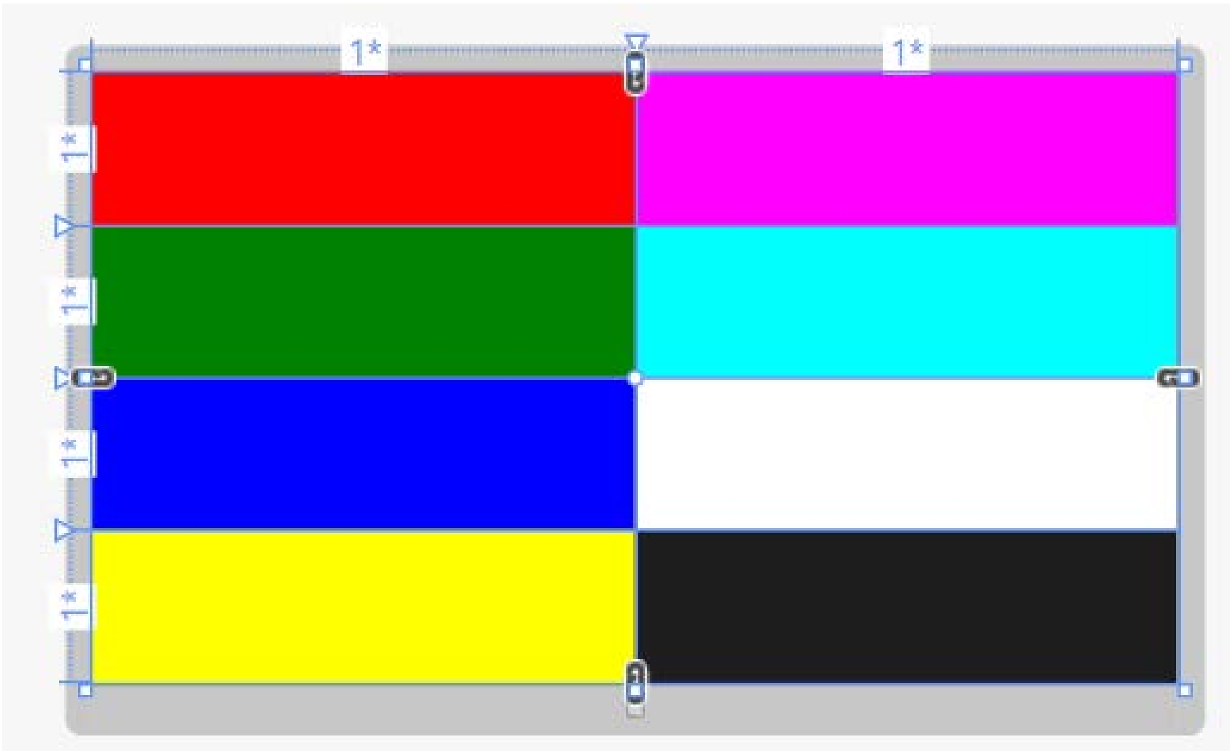


Figure 4 *– The above code listing results in a display of eight cells – four equally divided rows and two columns.*

Grids can be nested. In the next block of code, a second grid is placed inside the first column (default 0) of the third row (row value 2). This grid subdivides the cell into three columns, the middle one of which is filled with HotPink.

```
        <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
            <Grid.RowDefinitions>
                <RowDefinition>
                </RowDefinition>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
                <RowDefinition></RowDefinition>
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Rectangle Grid.Row="0" Fill="Red"/>
            <Rectangle Grid.Row="1" Fill="Green"/>
            <Rectangle Grid.Row="2" Fill="Blue"/>
            <Rectangle Grid.Row="3" Fill="Yellow"/>
            <Rectangle Grid.Row="0" Grid.Column="1" Fill="Magenta"/>
            <Rectangle Grid.Row="1" Grid.Column="1" Fill="Cyan"/>
            <Rectangle Grid.Row="2" Grid.Column="1" Fill="White"/>
            <Grid Grid.Row="2">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                    <ColumnDefinition/>
```

```
                </Grid.ColumnDefinitions>
                <Rectangle Grid.Column="1" Fill="HotPink"/>
            </Grid>
        </Grid>
```

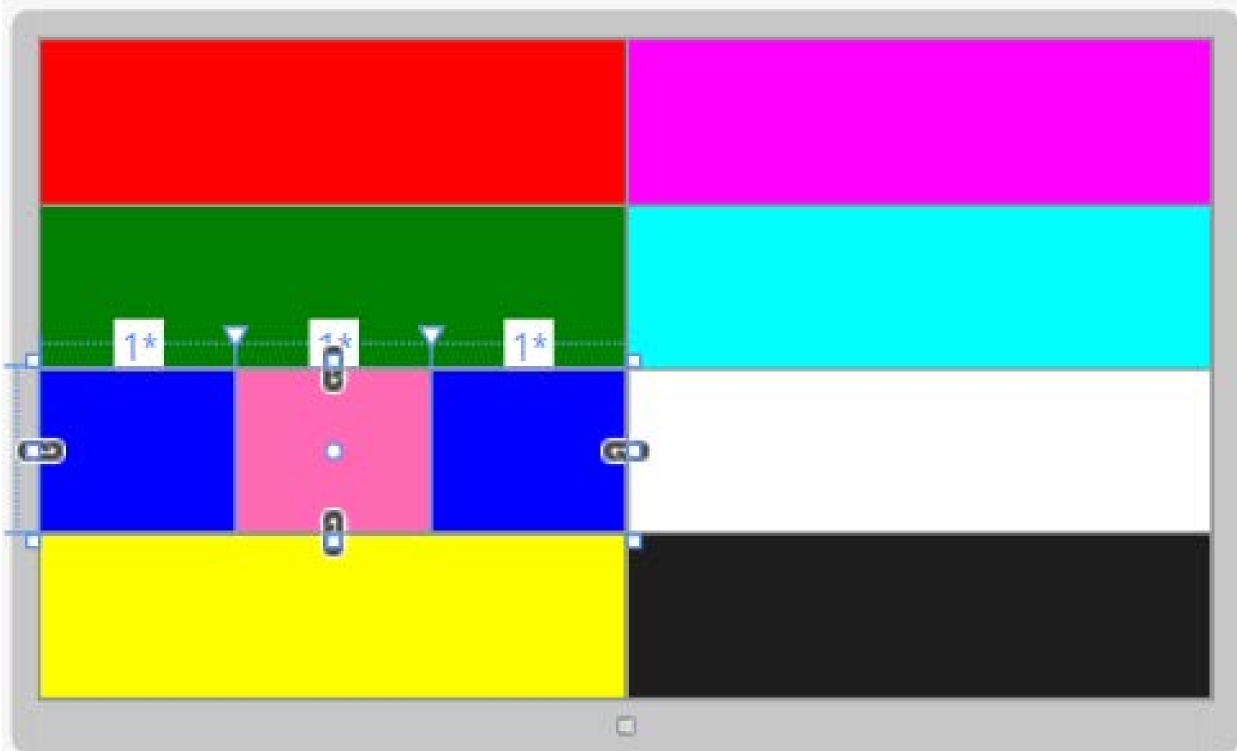


Figure 5 *– Row 2 and Column 0 has been subdivided by another Grid control of three columns with its center column (Column 1) shaded with a hot pink fill.*

Rows need not be equal height. A height attribute can be set and measured in pixels. In the next block of code, the first row is set to a height of 400 pixels with the bottom row (row 3) set to 50 pixels. The two middle rows have no height values set, and thus take up equal heights of the remaining pixels.

```
        <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
            <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
                <Grid.RowDefinitions>
                    <RowDefinition Height="400"/>
                    <RowDefinition/>
                    <RowDefinition/>
                    <RowDefinition Height="50"/>
                </Grid.RowDefinitions>
                <Rectangle Grid.Row="0" Fill="Red"/>
                <Rectangle Grid.Row="1" Fill="Green"/>
                <Rectangle Grid.Row="2" Fill="Blue"/>
                <Rectangle Grid.Row="3" Fill="Yellow"/>
            </Grid>
        </Grid>
```

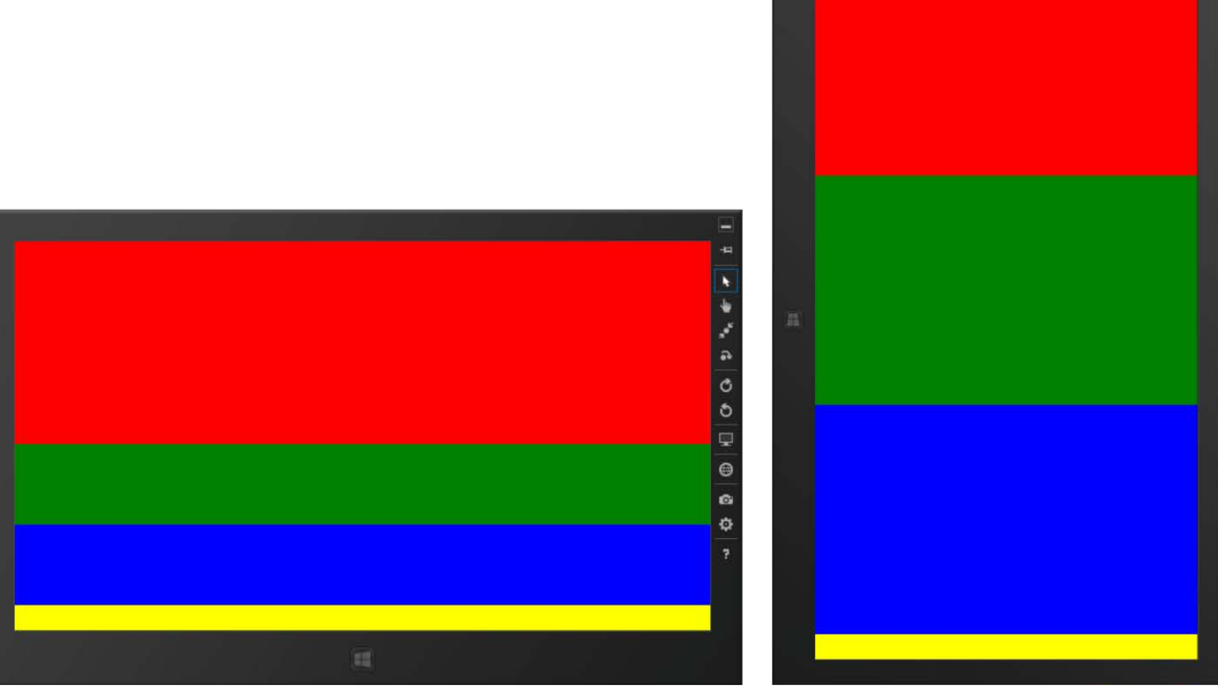


Figure 6 *– The top cell of the grid is 400 pixels and the bottom cell is 50 pixels. The middle two cells split the remaining pixels. In landscape view on a 1366x768 tablet these cells would be 159 pixels in height, but 458 pixels in portrait mode.*

Similarly, you can divide a grid into columns of varying widths by specifying a Width attribute value for each individual column. The second row in the next block is divided into three columns, the first two columns measuring 150 and 300 pixels respectively, with the third column consuming the remaining horizontal space. Note the use of the optional wildcard `"*"` value for the second row and the third column specifications.

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="140"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Rectangle Grid.Row="0" Fill="Olive"/>
    <Rectangle Grid.Row="1" Fill="White"/>
    <Grid Grid.Row="1">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="150"/>
            <ColumnDefinition Width="300"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Rectangle Grid.Column="0" Fill="Aquamarine"/>
        <!-- Row 1 is assumed in this block -->
        <Rectangle Grid.Row="1" Grid.Column="1" Fill="Maroon"/>
        <Rectangle Grid.Row="1" Grid.Column="2" Fill="Goldenrod"/>
    </Grid>
</Grid>
```

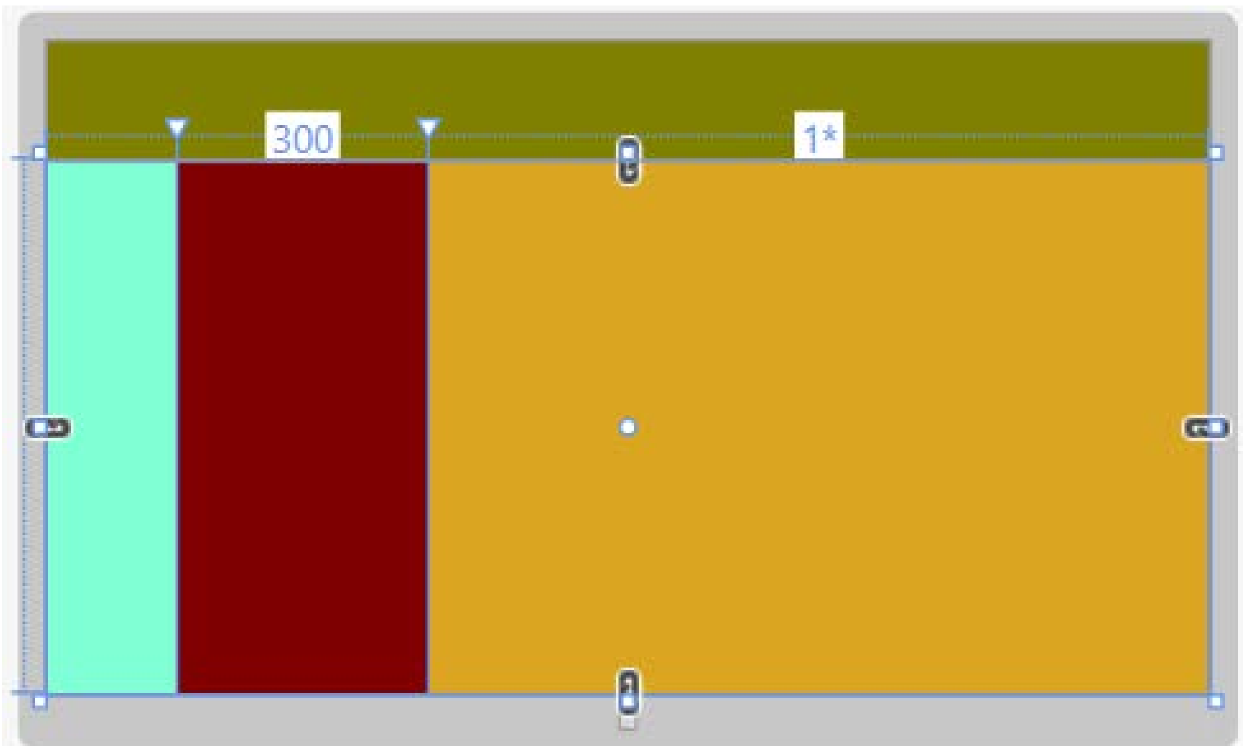


Figure 7 *– Row 1 and Column 2 of row 1 of this grid display are both set to the wildcard "*" value, which takes up the remaining vertical and horizontal spaces respectively.*

Text Controls - Overview

There are six primary controls for displaying text. The TextBlock, RichTextBlock, and RichTextBlockOverflow controls are static controls for displaying read-only values. The remaining three text controls are the TextBox, RichEditBox, and PasswordBox. These controls are for gathering user input.

The TextBlock Control

The TextBlock control is like the Label control of desktop applications in the Visual Studio IDE, but with greater formatting capabilities. The displayed content (Text attribute) is static and cannot be altered directly by the user. The font face can be set using the FontFamily and FontSize attributes. For demonstration purposes, a large TextBlock is created in the following code.

```
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
                                    FontFamily="Segoe UI" FontSize="48">
               South Mountain Community College
        </TextBlock>
```

The same display of text could be achieved using the Text attribute:

```
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
             FontFamily="Segoe UI" FontSize="48"
             Text="South Mountain Community College" />
```

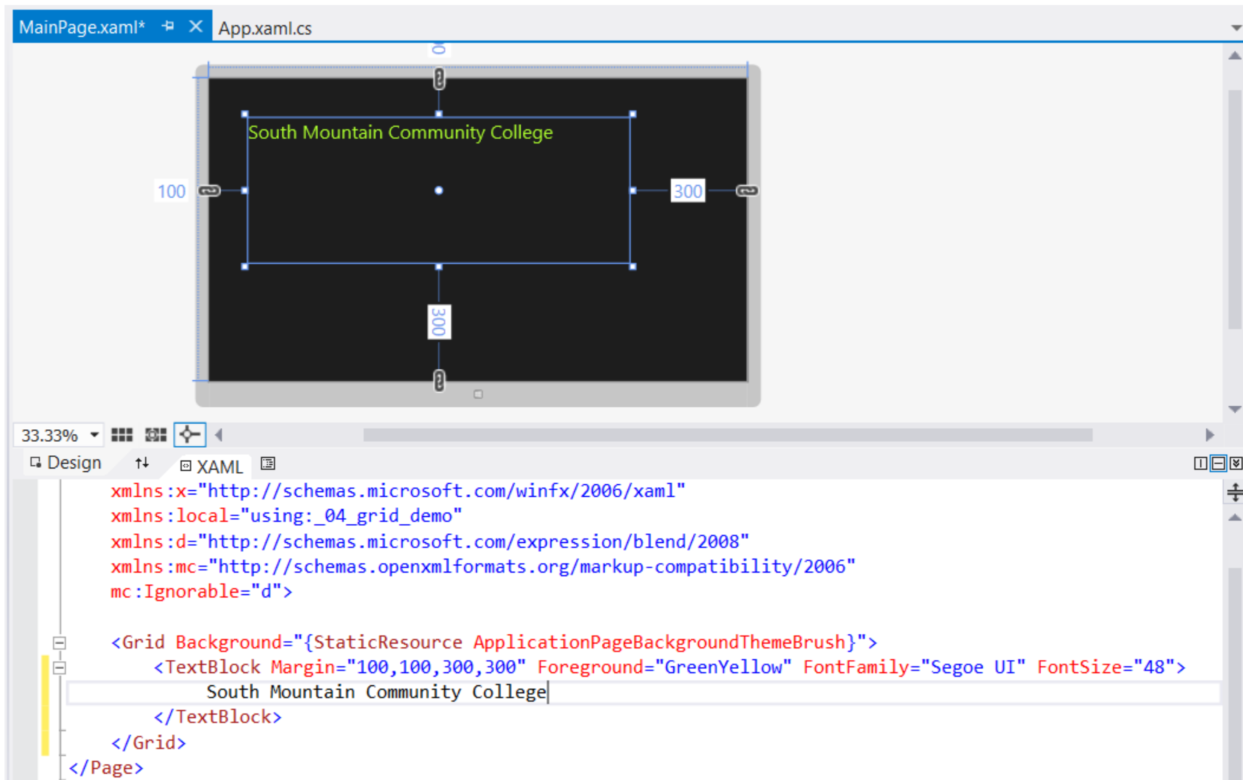


Figure 8 *– The font values are assigned as attributes in the TextBlock control.*

As with HTML, Bold and Italic tags can be applied for inline formatting.

```
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
                   FontFamily="Segoe UI" FontSize="48">
            <Bold>South Mountain </Bold> <Italic> Community College</Italic>
        </TextBlock>
```

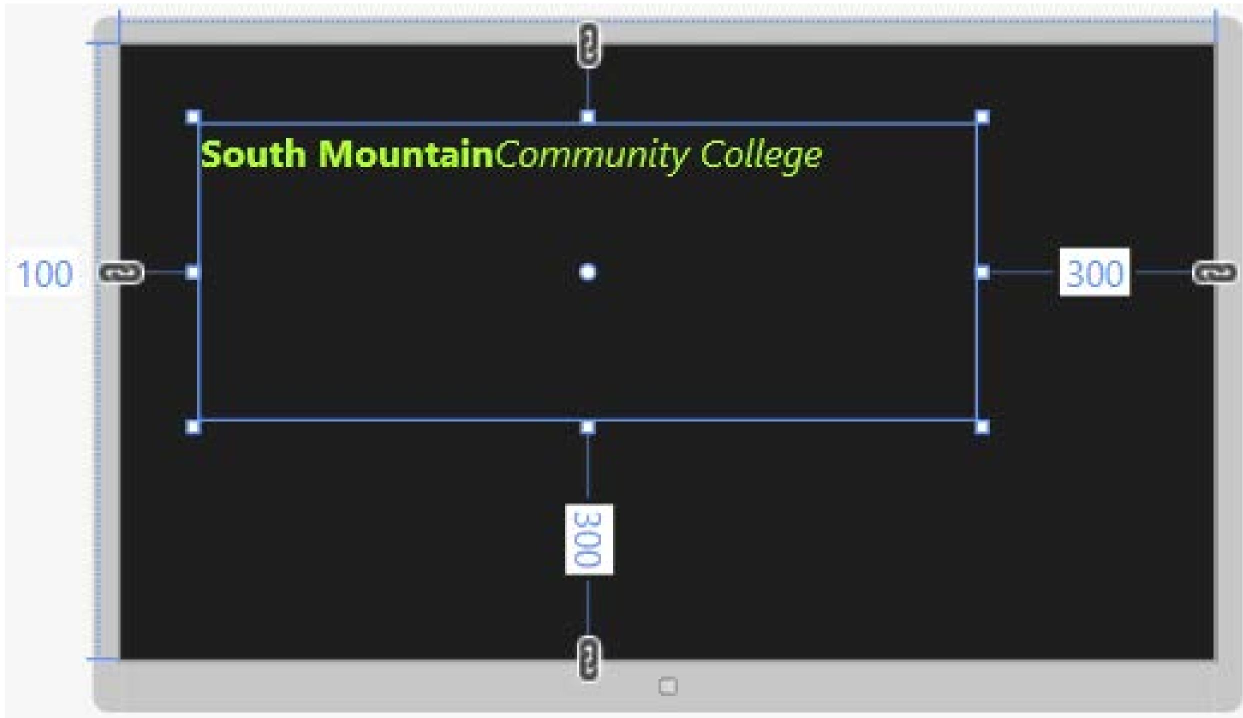


Figure 9 *– Individual phrases, words, or even letters can be formatted with tags such as <Bold> and <Italic>.*

Note that the white space is ignored trailing the "South Mountain" text and preceding the "Community College" text; the two text values are concatenated without a space between them in the Design panel preview. If you test this in the Simulator, however, you will notice that the space does appear. It is wise to test your XAML code before assuming that it is wrong.

Note that the Font family and sizes cannot be changed within the Text property. If you try the following, you will get an error:

```
<TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
                 FontFamily="Segoe UI" FontSize="48">
         South Mountain <FontSize="32"> Community College</FontSize>
</TextBlock>
```

MainPage.xaml* App.xaml.cs

Invalid Markup

Check the Error List for more information.

View code

Design XAML

```
<Page
    x:Class="_04_grid_demo.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_04_grid_demo"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow" FontFamily="Segoe UI" FontSize="48">
            South Mountain <FontSize="32"> Community College</FontSize>
        </TextBlock>
    </Grid>
</Page>
```

Figure 10 *– Errors in the XAML code will result in suspension of the visual display in the Design panel. In this case, FontSize tags are not allowed.*

You can fix this font altering issue by using Runs (or Spans). The TextBlock automatically concatenates Runs and allows font values to be set in each run. White space is still ignored in the Design panel preview but looks fine in the Simulator.

```
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
                FontFamily="Segoe UI" FontSize="48">
            <Run>South Mountain </Run>
            <Run FontSize="32" > Community College</Run>
            <!-- Runs allow for formatting changes -->
        </TextBlock>
```

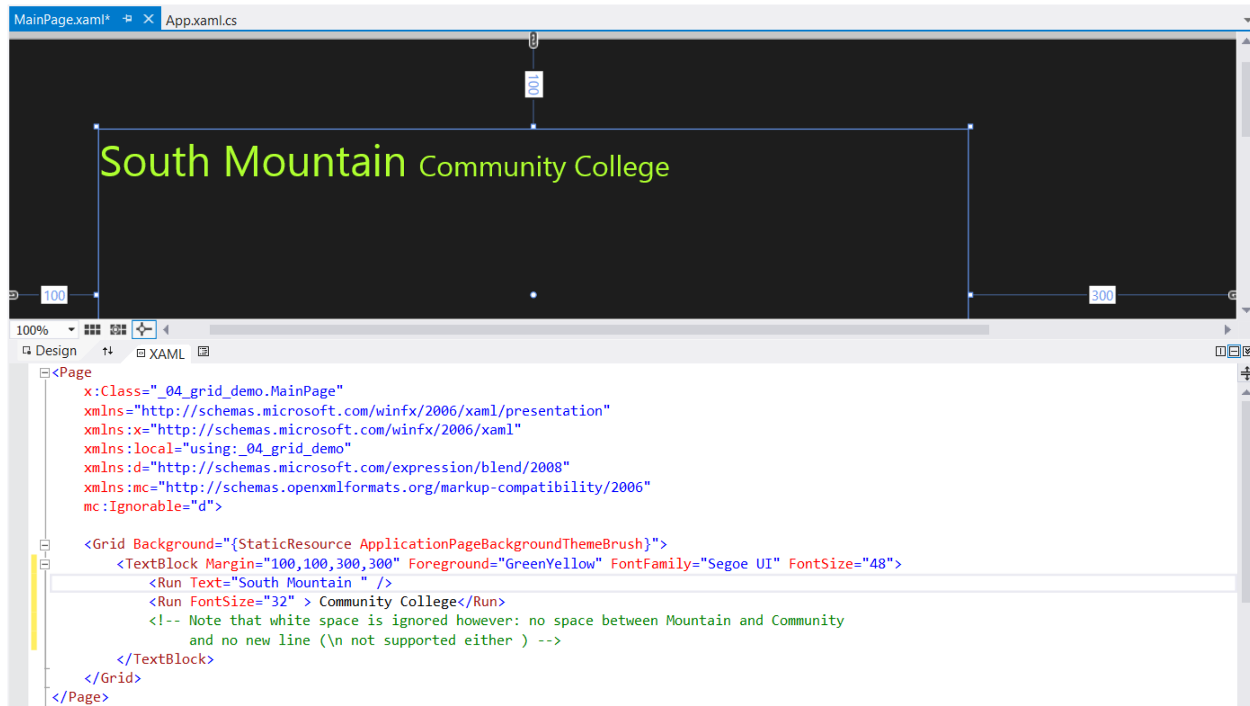


Figure 11*- Font attributes such as FontFamily, FontSize, and Foreground color can be specified within individual Run or Span tags.*

CharacterSpacing is another TextBlock attribute. It sets the tracking of the text. It is measured in thousandths of an em space (the width of the letter m in the particular font/size). A negative value will draw the text tighter together and a positive value will space text further apart. <LineBreak/> tags can be employed to begin a new line.

```
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
                   FontFamily="Segoe UI" FontSize="48">
            <Run FontWeight="Bold" Text="South Mountain " />
                  <Italic>Community College</Italic><LineBreak/>
            <Run CharacterSpacing="100"> South Mountain Community College</Run>
            <LineBreak/>
            <Run> South Mountain Community College</Run> <LineBreak/>
            <Run CharacterSpacing="-100"> South Mountain Community College</Run>
        </TextBlock>
```

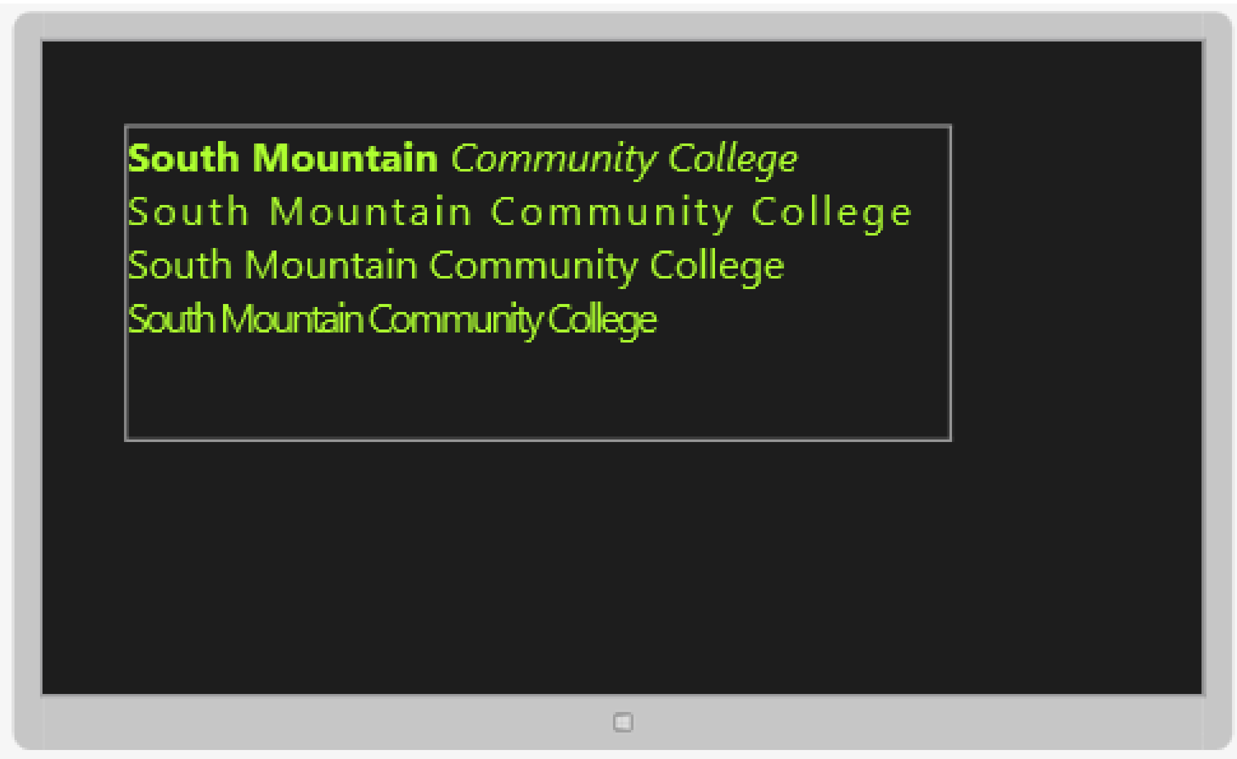


***Figure 12** – Text with CharacterSpacing attributes set for the second and fourth lines and LineBreak tags to create new lines.*

The FontWeight attribute sets the relative weight of the active font. The possible values are as follows (from the thinnest to the heaviest):

- Thin
- ExtraLight
- Light
- SemiLight
- Normal
- SemiBold
- Bold
- ExtraBold
- Black
- ExtraBlack

The application of the FontWeight will however vary from font family to font family. For instance, with the Segoe UI font, Thin, Light, and SemiLight all map to Light, while Bold, ExtraBold, and Black map to Bold.

```
<TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
           FontFamily="Segoe UI" FontSize="48">
    <Bold>South Mountain </Bold> <Italic> Community College</Italic>
    <LineBreak/>
    <Run Foreground="Tomato" FontWeight="Light">
       South Mountain Community College</Run>
</TextBlock>
```

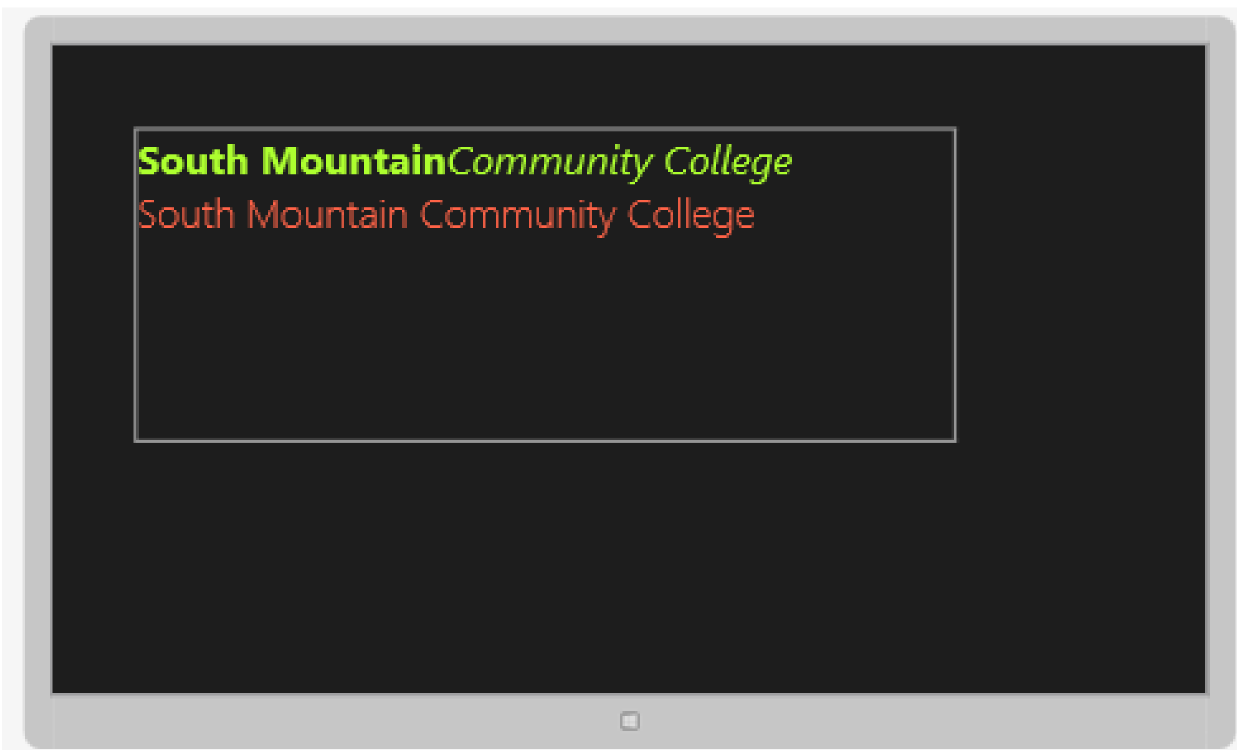


Figure 13 *– A FontWeight attribute of 'Light' was set for the second line of text.*

The text in the following TextBlock definition does not wrap.

```
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
                                    FontFamily="Segoe UI" FontSize="48">
            South Mountain Community College is located in Phoenix, Arizona.
        </TextBlock>
```

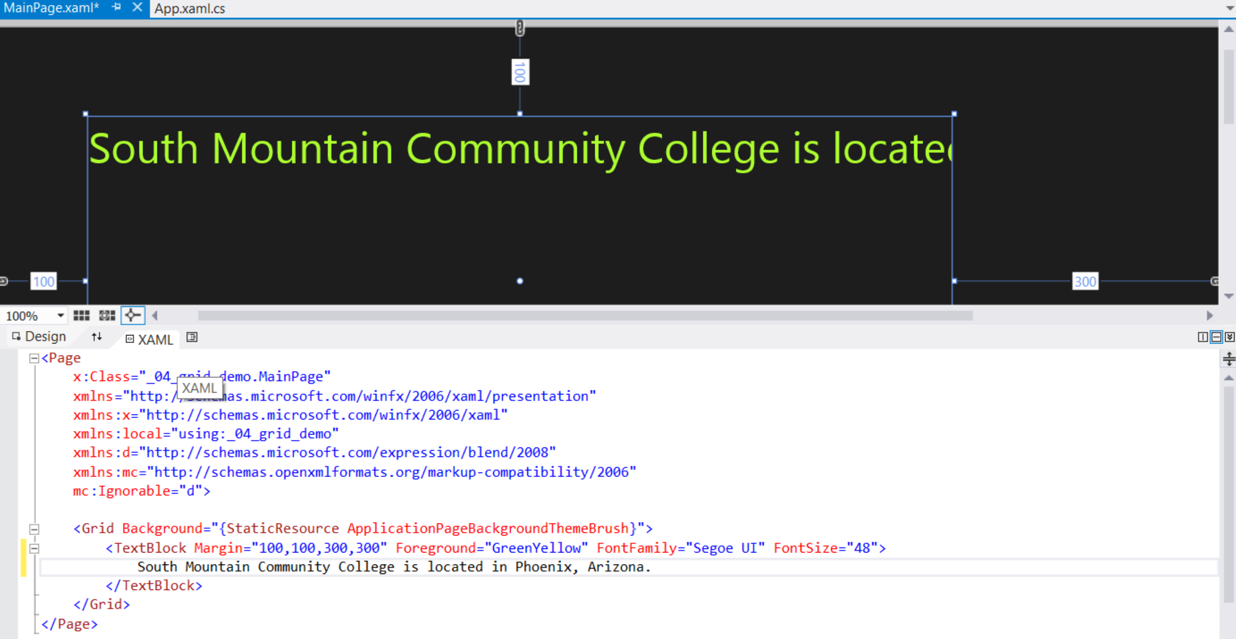


Figure 14 *– By default, TextBlocks do not wrap their text.*

You can change this behavior by specifying a TextWrapping attribute of "Wrap". The default is "NoWrap".

```
        <TextBlock Margin="100,100,300,300" Foreground="GreenYellow"
                   FontFamily="Segoe UI" FontSize="48" TextWrapping="Wrap">
            South Mountain Community College is located in Phoenix, Arizona.
        </TextBlock>
```

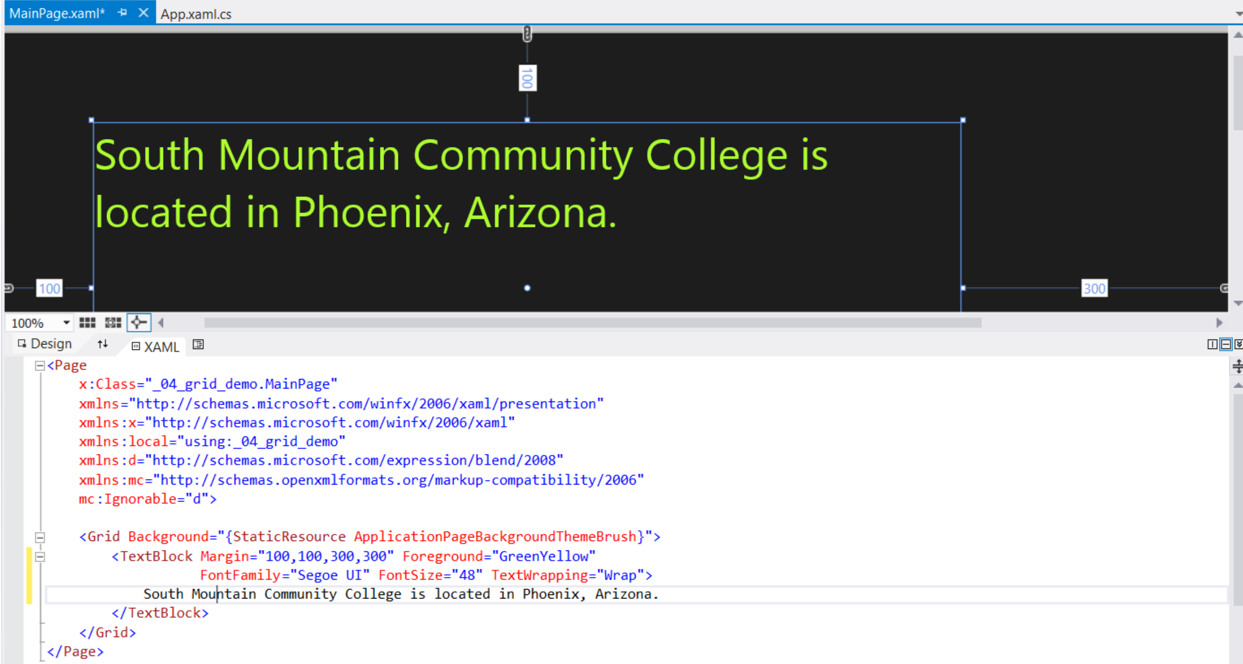


Figure 15 *– A "Wrap" setting for the TextWrapping attribute results in the text being displayed on multiple lines if necessary.*

The leading of text in a TextBlock may be modified using a LineHeight attribute.

```
        <TextBlock Margin="200,200,800,100" FontFamily="Segoe UI" FontSize="42"
                   Foreground="Yellow">
              LineHeight not set<LineBreak/>
              South<LineBreak/>Mountain<LineBreak/>Community<LineBreak/>College
        </TextBlock>
        <TextBlock Margin="700,200,300,100" FontFamily="Segoe UI" FontSize="42"
                   Foreground="Yellow" LineHeight="100" >
              LineHeight="100"<LineBreak/>
              South<LineBreak/>Mountain<LineBreak/>Community<LineBreak/>Colle
        </TextBlock>
```

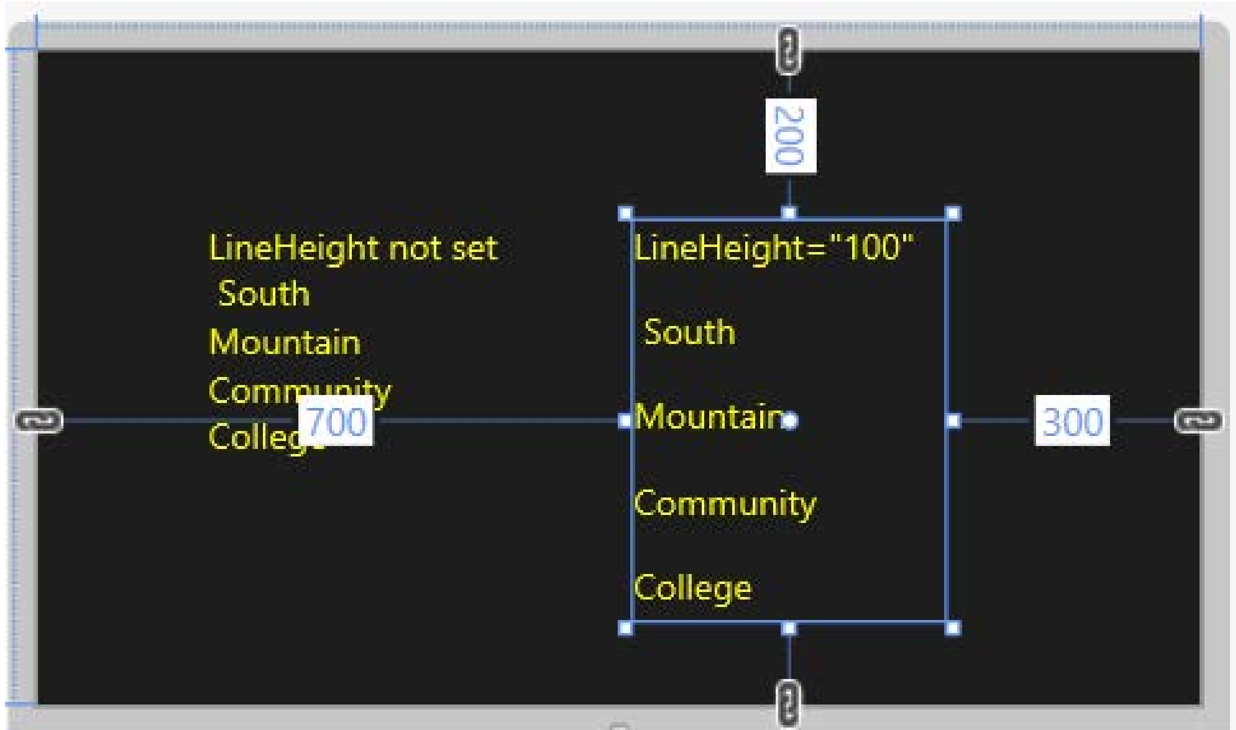


***Figure 16** – Two contrasting TextBlocks with the only difference being that the one on the right has a LineHeight attribute setting of 100 points.*

Adding a ScrollViewer Element

The TextBlock control does not have a ScrollBar attribute, but a TextBlock might be placed inside a pair of ScrollViewer element tags.

```
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="150"/>
            <RowDefinition Height="550"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="50"/>
            <ColumnDefinition Width="700"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <ScrollViewer Grid.Row="1" Grid.Column="1"
                      VerticalScrollBarVisibility="Visible">
            <TextBlock Margin="0,0,0,0" Foreground="GreenYellow"
                             FontFamily="Segoe UI" FontSize="48"
                             TextWrapping="Wrap" >
                South Mountain Community College offers courses in:
                <LineBreak/>Adobe Flash
                <LineBreak/>Adobe Illustrator
                <LineBreak/>Adobe Photoshop
                <LineBreak/>Cisco (CCNA; CCNP; Security, etc.)
                <LineBreak/>HTML5/CSS/JavaScript
                <LineBreak/>Microsoft Access<LineBreak/>Microsoft Excel
                <LineBreak/>Microsoft PowerPoint
                <LineBreak/>Microsoft PowerPoint
```

```
                <LineBreak/>Microsoft Visual Studi0 - c#
                <LineBreak/>Visual Studio - VB.NET
                <LineBreak/>Android programming (Java)
                <LineBreak/>iOS/iPhone/iPad programming (Objective C)
                <LineBreak/>Windows 8 App Porgramming (C#/VB with XAML)
            </TextBlock>
        </ScrollViewer>
    </Grid>
```

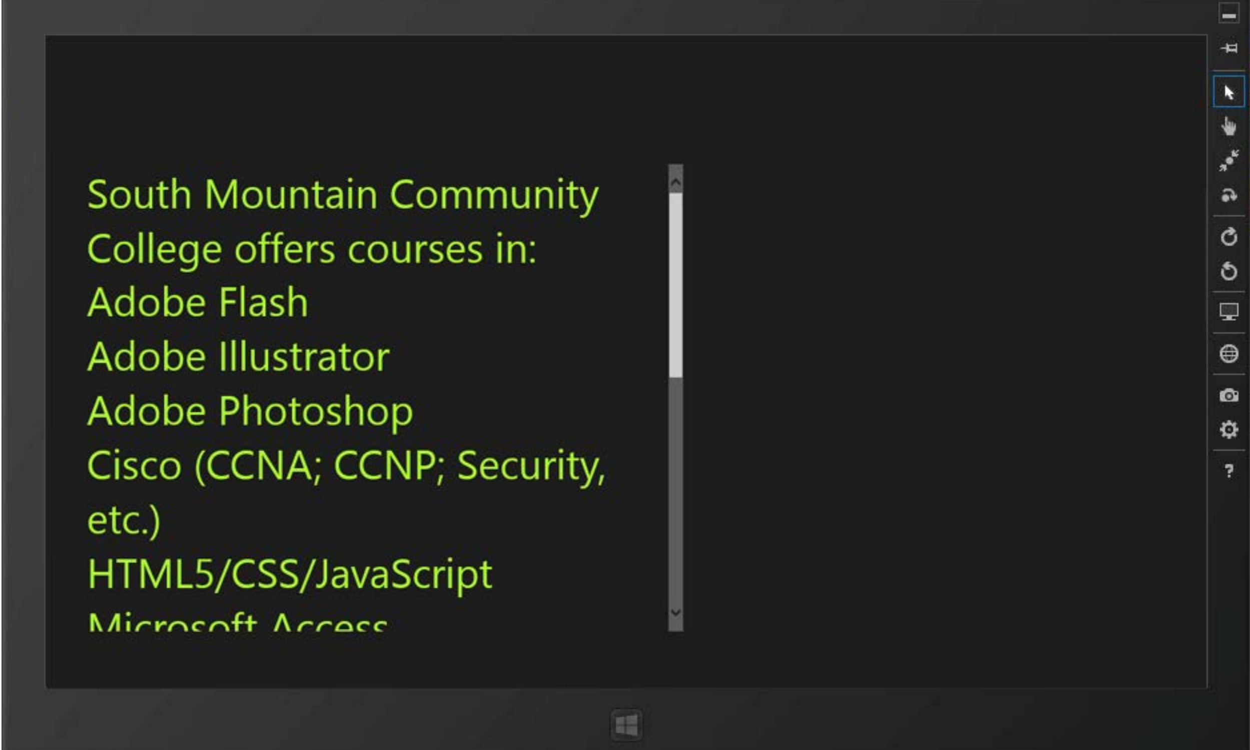


Figure 17 *– A ScrollViewer control added to a Grid cell with a TextBlock inside the ScrollViewer.*

The RichTextBlock Control

The RichTextBlock is very similar to the TextBlock, but with some added formatting capabilities.

While Runs may be used in RichTextBlocks, the display in a RichTextBlock is usually comprised of blocks. RichTextBlocks support <Paragraph> and </Paragraph> tags to create the blocks. Unlike TextBlocks, RichTextBlocks automatically implement text wrapping.

```
        <RichTextBlock Margin="200,200,200,200">
            <Paragraph FontFamily="Segoe UI" FontSize="42" Foreground="Yellow">
                SMCC to Offer WIndows 8 Programming Class in Spring 2013
            </Paragraph>
            <Paragraph/>
            <!--This blank paragraph provides a bit of extra
                                                    spacing.-->
            <Paragraph FontFamily="Segoe UI" FontSize="22" Foreground="White">
                South Mountain Community College will offer a Windows 8 App
            Programming course for students who wish to build upon their
```

```
            knowledge either C# or VB desktop programming to create apps
            for the Windows Store. A key new skill that will be learned
            in the course will be interface creation with XAML elements.
            </Paragraph>
        </RichTextBlock>
```

Figure 18 *– RichTextBlocks incorporate paragraphs into their display.*

Most of the same attributes available to formatting TextBlocks are also pertinent to RichTextBlocks. Fonts are set using FontFamily, FontSize, FontStyle, FontWeight, FontStretch. You can further specify formatting by setting TextAlignment, TextWrapping, LineHeight, and CharacterSpacing attributes.

```
        <RichTextBlock Margin="200,200,200,200">
            <Paragraph FontFamily="Segoe UI" FontSize="42" Foreground="Yellow" >
                SMCC to Offer
                <Run FontFamily="Courier New" Foreground="DarkOrange">
                    Window 8
                </Run> Programming Class in Spring 2013
            </Paragraph>
            <Paragraph/>
            <!--This blank paragraph provides a bit of extra spacing.-->
            <Paragraph FontFamily="Segoe UI" FontSize="22" Foreground="White"
                       LineHeight="50">
                South Mountain Community College will offer a
                <Italic>Windows 8 App Programming</Italic> Windows 8 App
            Programming course students who wish to build upon their
            knowledge of either C# or VB desktop programming to create
            apps for the
                <Bold>Windows Store</Bold>.
                <LineBreak/>
                <LineBreak/>
```

```
                <Run Foreground="Red">A key new skill that will be learned in the
                Course will be interface creation with XAML elements.</Run>
            </Paragraph>
        </RichTextBlock>
```

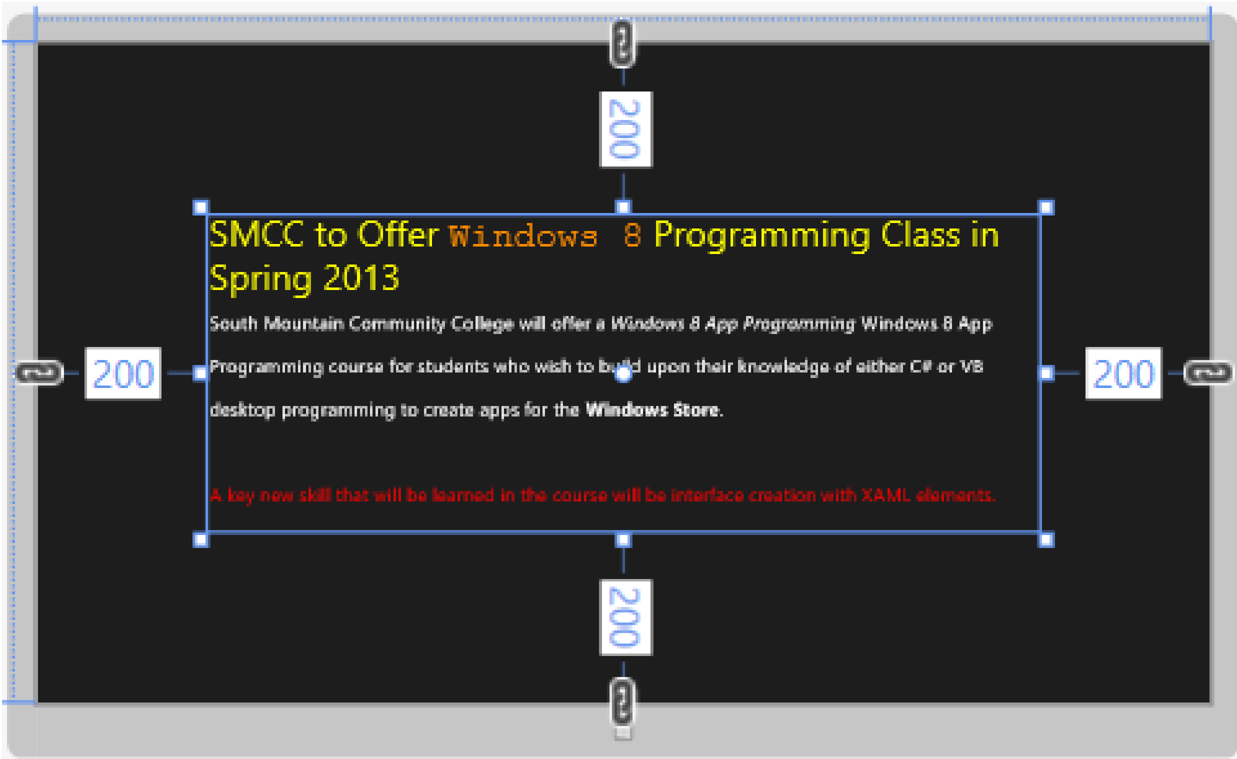


Figure 19 *– Formatting can be set within individual <Paragraph> tags of a RichTextBlock. Inline tags such as <Bold> and <Italic> may also be applied.*

One of the unique features of a RichTextBlock is the InlineUIContainer element, which you can use to embed other controls inline in the RichTextBlock's text. These controls might include one or more Buttons, Images, RadioButtons, or other controls.

```
        <RichTextBlock Margin="100,200,100,200">
CP: Please add photo and caption.        <Paragraph FontFamily="Segoe UI" FontSize="22"
Foreground="Yellow" >
                SMCC is located in [
                <InlineUIContainer >
                    <StackPanel Orientation="Horizontal">
                        <RadioButton Content="Flagstaff"/>
                        <RadioButton Content="Phoenix"/>
                        <RadioButton Content="Tucson"/>
                    </StackPanel>
                </InlineUIContainer>
                ] in the state of Arizona.
            </Paragraph>
        </RichTextBlock>
```

SMCC is located in [Flagstaff Phoenix Tucson] in the state of Arizona.

***Figure 20** – The <InlineUIContainer > element allows other controls to be embedded within the text of a RichTextBlock.*

The contents of both RichTextBlocks and TextBlocks are selectable by the user. Selections can be handled by a SelectionChanged event and using the SelectedText property of the control. The ability to select the contents of either can be modified by setting the IsTextSelectionEnabled attribute to False. IsTapEnabled, IsDoubleTapEnabled, and IsRightTapEnabled properties (set to true by default) also allow the text of the TextBlock or RichTextBlock to respond to other touch events.

RichTextBoxOverflow

The RichTextBoxOverflow control is to display additional text from a RichTextBox control (or another RichTextBlockOverflow control) that is unable to display all of its contents within its boundaries.

In the following example, text is assigned to the RichTextBlock named rtbColumn1. If it cannot display all the text, the remainder is displayed in its OverflowContentTarget, which is specified as the RichTextBlockOverflow control named ofColumn2. It in turn has an overflow target of the ofColumn3 RichTextBoxOverflow control. And the ofColumn3 control overflows to the RichTextBoxOverflow control named ofColumn4.

Note that the text property of the first column control (the RichTextBlock named rtbColumn1) is coming from the value of a defined StaticResource string named BodyText. BodyText is the text content of the Declaration of Independence.

```
        <RichTextBlock x:Name="rtbColumn1" HorizontalAlignment="Left" Height="587"
                   Margin="50,5,0,0" Grid.Row="1" Style="{StaticResource
                   ItemRichTextStyle}" TextWrapping="Wrap" VerticalAlignment="Top"
                   Width="275"  OverflowContentTarget="{Binding ElementName=ofColumn2}" >
            <Paragraph >
                <Run Text="{StaticResource BodyText}"/>
            </Paragraph>
        </RichTextBlock>
        <RichTextBlockOverflow x:Name="ofColumn2" HorizontalAlignment="Left"
                Height="588" Margin="375,5,0,0" Grid.Row="1" VerticalAlignment="Top"
                Width="275"  OverflowContentTarget="{Binding ElementName=ofColumn3}"/>
        <RichTextBlockOverflow x:Name="ofColumn3" HorizontalAlignment="Left"
                Height="588" Margin="701,4,0,0" Grid.Row="1" VerticalAlignment="Top"
                Width="275" OverflowContentTarget="{Binding ElementName=ofColumn4}"/>
        <RichTextBlockOverflow x:Name="ofColumn4" HorizontalAlignment="Left"
                Height="588" Margin="1025,4,0,0" Grid.Row="1" VerticalAlignment="Top"
                Width="275"/>
```

***Figure 21** – RichTextBlockOverflow controls are utilized to have text flow from one control to the other.*

The BodyText resource was defined in the PageResources.

```
    <Page.Resources>
        <x:String x:Key="AppName">Declaration of Independence</x:String>
        <x:String x:Key="BodyText" >IN CONGRESS, July 4, 1776.
    The unanimous Declaration of the thirteen united States of America
    When in the Course of human events, it becomes necessary for one people to dissolve
the political bands which have connected them with another, and to assume among the
powers of the earth, the separate and equal station to which the Laws of Nature and of
. . .
of this Declaration, with a firm reliance on the protection of divine Providence, we
mutually pledge to each other our Lives, our Fortunes and our sacred Honor.
        </x:String>
    </Page.Resources>
```

Note that white spaces are ignored in the above String resource. To have the white space honored in the text, an xml:space="preserve" attribute can be added to the String element.

```
    <Page.Resources>
        <x:String x:Key="AppName">Declaration of Independence</x:String>
        <x:String x:Key="BodyText" xml:space="preserve">IN CONGRESS, July 4, 1776.
    The unanimous Declaration of the thirteen united States of America
    When in the Course of human events, it becomes necessary for one people to dissolve
the political bands which have connected them with another, and to assume among the
powers of the earth, the separate and equal station to which the Laws of Nature and of
. . .
```

```
of this Declaration, with a firm reliance on the protection of divine Providence, we
mutually pledge to each other our Lives, our Fortunes and our sacred Honor.
        </x:String>
    </Page.Resources>
```

The TextBox Control

The TextBox control is used for gathering input from the users and functions as the TextBox controls did in developing desktop applications. It shares the FontFamily, FontSize, FontStyle, FontWeight, FontStretch, TextAlignment, TextWrapping, LineHeight, and CharacterSpacing attributes.

```
        <TextBlock HorizontalAlignment="Left" Margin="184,409,0,0" TextWrapping="Wrap"
                   Text="Enter First Name:" VerticalAlignment="Top"/>
        <TextBox x:Name="fname" HorizontalAlignment="Left" Margin="284,398,0,0"
                 VerticalAlignment="Top" Width="128"/>
        <TextBlock HorizontalAlignment="Left" Margin="184,449,0,0" TextWrapping="Wrap"
                   Text="Enter Last Name:" VerticalAlignment="Top"/>
        <TextBox x:Name="lname" HorizontalAlignment="Left" Margin="284,438,0,0"
                 VerticalAlignment="Top" Width="128"/>
```

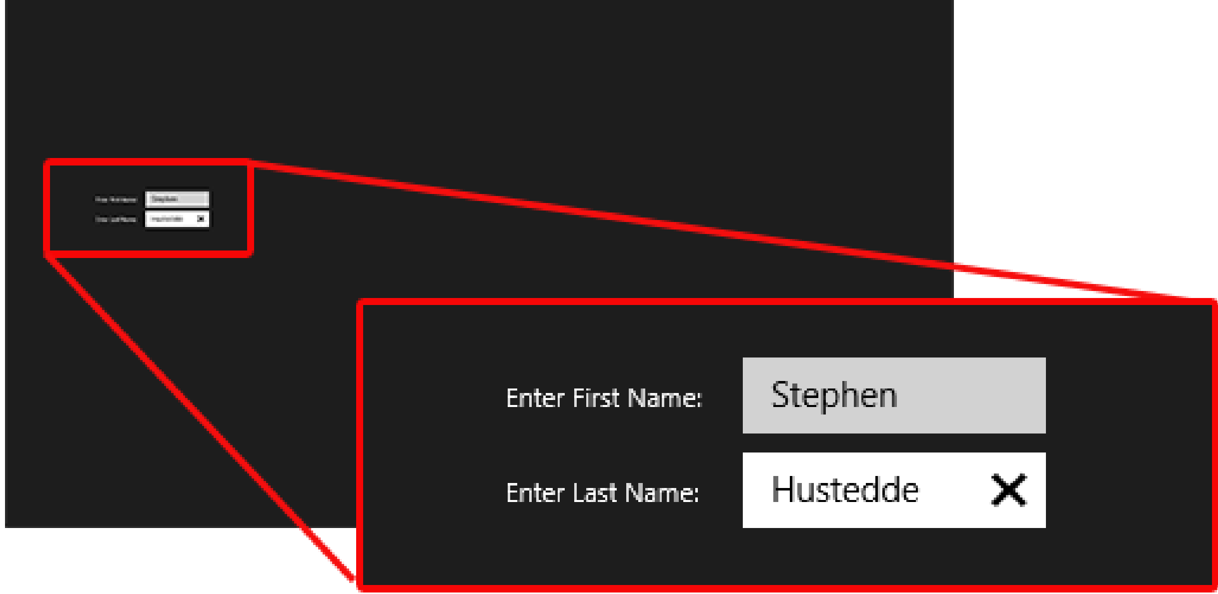


Figure 22 *– The TextBox control is used to get input from the user.*

Explore the resources in the "TextBox" class on the Windows Dev Center – Windows Store Apps website.

Changing the Text Display Programmatically

The text property of a named textbox may be altered in code as was done for Windows desktop applications:

txtTeam.Text = "The Bears"

It is trickier for TextBlocks or RichTextBlocks. The display of a TextBlock control is within an Inlines container. The Inlines container may consist of multiple items, such as several Runs or Spans. The format of these items may be modified by referencing the Item index number

within the object's Inlines container. Likewise, for a RichTextBlock, the display is comprised of multiple items in a Blocks container. These items might be Runs, Spans, or Paragraphs. To modify the actual text in either a TextBlock or RichTextBlock via code, it is best to give the item (the Run element or the Paragraph element) a name and then set the Text for the named element.

Consider the following page:

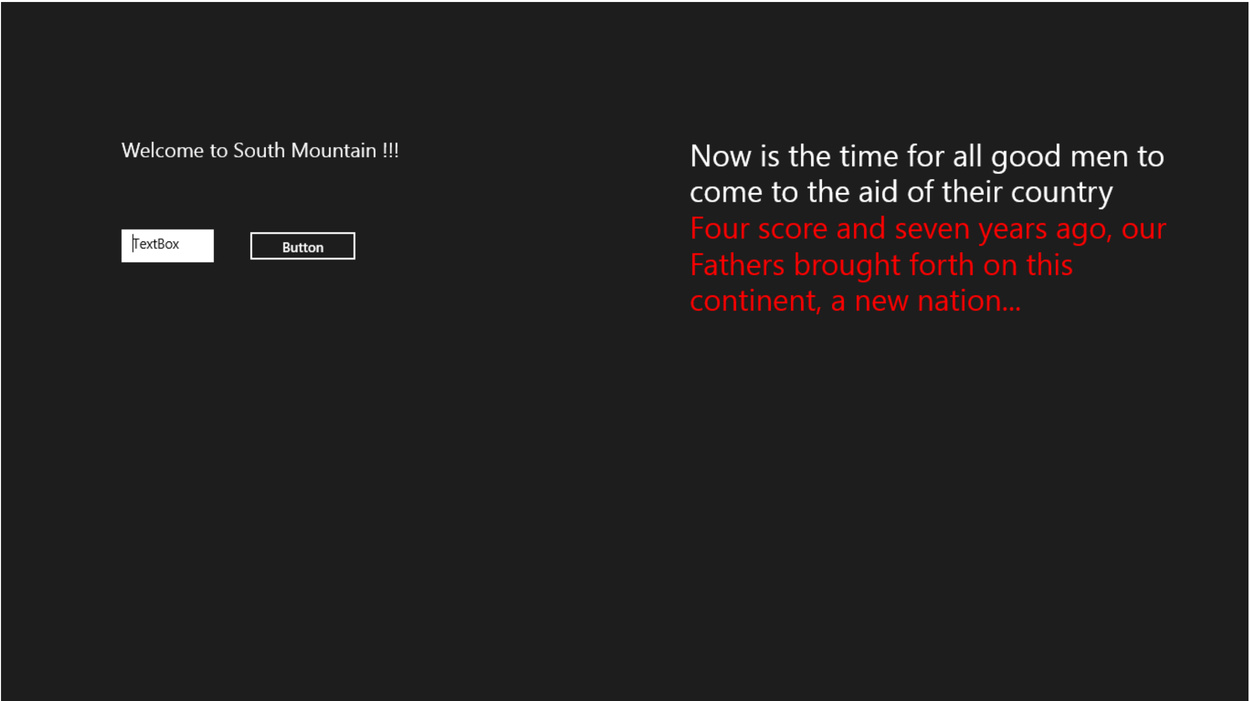


***Figure 23** – The page before the Button is pressed.*

The XAML code for the above page contains:

```
<TextBlock x:Name="TextBlock1" HorizontalAlignment="Left" Height="60"
      Margin="132,150,0,0" FontSize="22" TextWrapping="Wrap" VerticalAlignment="Top"
      Width="584">
    <Run>Welcome to </Run>
    <Run x:Name="campus">South Mountain</Run>
    <Run>!!!</Run>
</TextBlock>

<RichTextBlock x:Name="rtbQuotes" HorizontalAlignment="Left" Height="240"
        Margin="754,150,0,0" VerticalAlignment="Top" Width="525" FontSize="33">
    <Paragraph>Now is the time for all good men to come to the
        aid of their country</Paragraph>
    <Paragraph x:Name="P2" Foreground="Red">
        <Run x:Name="Run2">Four score and seven years ago,
        our fathers brought forth on this continent, a new nation...</Run></Paragraph>
</RichTextBlock>


<TextBox x:Name="txtInfo" HorizontalAlignment="Left" Height="36" Margin="132,249,0,0"
         TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="101"/>
<Button Content="Button" HorizontalAlignment="Left" Height="36" Margin="270,249,0,0"
```

```
        VerticalAlignment="Top" Width="121" Tapped="Test"/>
```

After executing the following code, you see that the Textbox contains the value of five, which was the count of items for the Inlines container of the Textblock1 TextBlock.

VB Code:

```
    Private Sub Test(sender As Object, e As TappedRoutedEventArgs)
        Dim j As Integer = TextBlock1.Inlines.Count
        txtInfo.Text = j.ToString()
        'First Run is Item(0), Second Run is Item(2) and the third RUn is Item(4)
        TextBlock1.Inlines.Item(0).FontSize = 33
        TextBlock1.Inlines.Item(2).Foreground =
                             New SolidColorBrush(Windows.UI.Colors.Red)
        TextBlock1.Inlines.Item(4).Foreground =
                             New SolidColorBrush(Windows.UI.Colors.Yellow)
        campus.Text = "Arizona"  'named Run
        rtbQuotes.Blocks.Item(0).FontWeight = Windows.UI.Text.FontWeights.Bold
        P2.FontSize = 44
        Run2.Text = "A long time ago in a galaxy far, far way ...."
    End Sub
```

The font size of the first Run (Item 0) has been changed to 33, and the foreground colors of the second and third run (items 2 and 4) have been altered to red and yellow respectively. The text of the Run named "campus" has been changed from "South Mountain" to "Arizona".

For the RichTextBlock, the first item of the Blocks container has its FontWeight set to Bold. The second Paragraph element, given a name of "P2" has had its size increased to 44 and its Run element (named "Run2") has had its Text property modified.

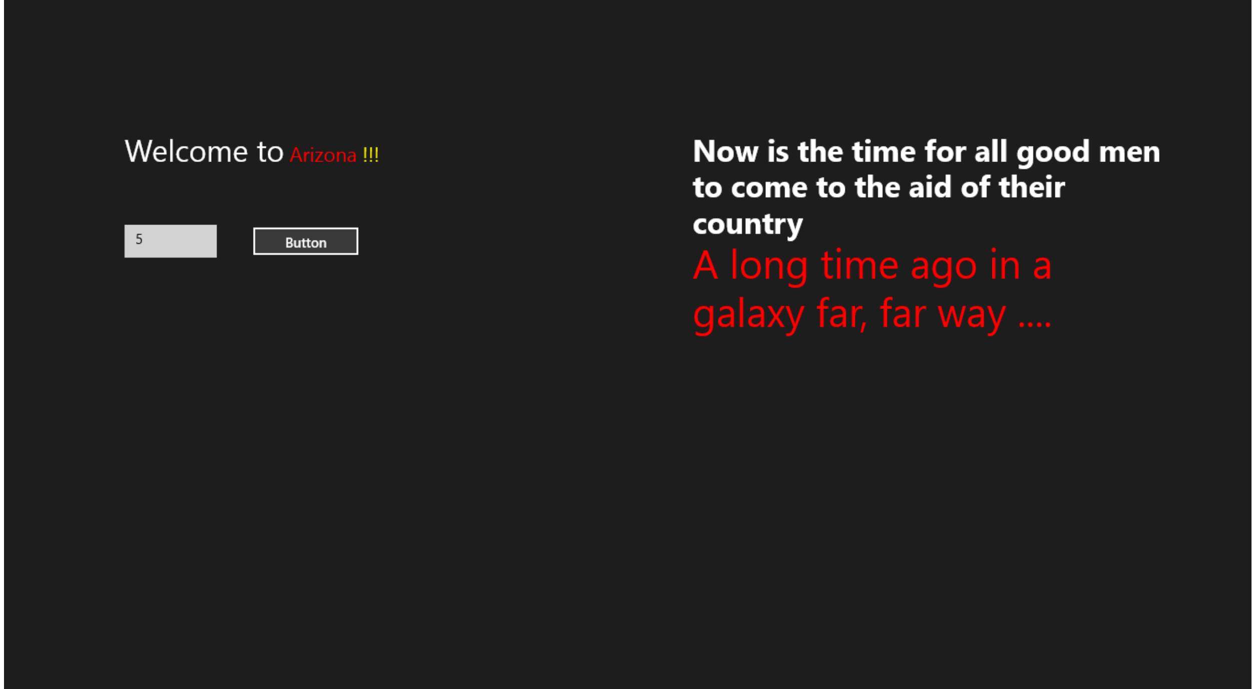


***Figure 25** – The page, after being modified, in the code behind.*

Predefined strings can be stored in Resource (.resw) files added to a project and loaded by a name given to the string. This is often done to make apps multilingual. To add a Resource (.resw) file to a project, right click on the Solution (or on a folder in the Solution Explorer such as Assets) and choose "Add…" > "New Item….". In the dialog, scroll down to select "Resource File (.resw)" and give the file a name before clicking "Add".

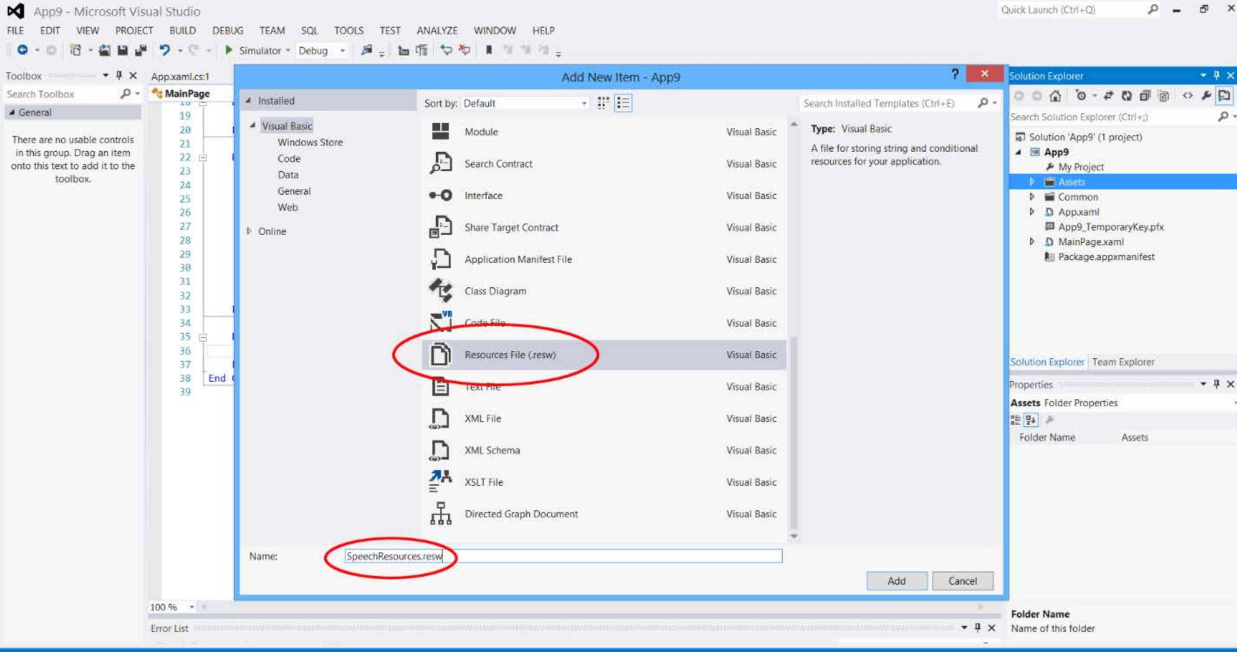
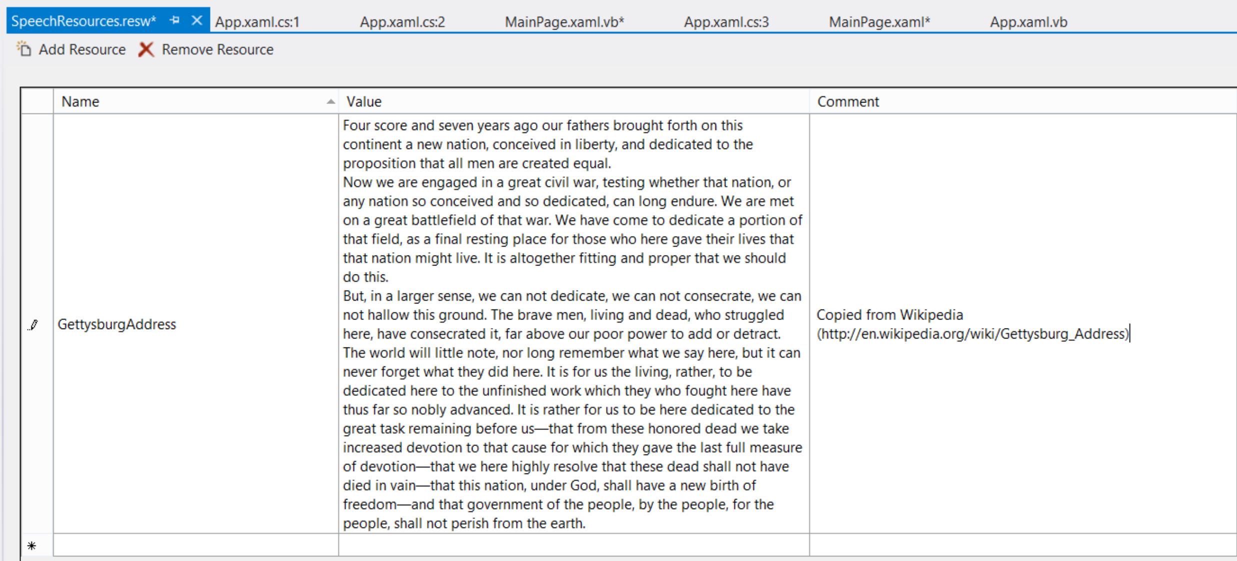


***Figure 24** – Adding a Resource file (.resw) to a project.*

The Resource file contains three columns. A unique name is given to the string resource in the first column, the value (the string itself) is entered or pasted into the second column, and any comments (if desired) can be placed in the last column.



Name	Value	Comment
GettysburgAddress	Four score and seven years ago our fathers brought forth on this continent a new nation, conceived in liberty, and dedicated to the proposition that all men are created equal. Now we are engaged in a great civil war, testing whether that nation, or any nation so conceived and so dedicated, can long endure. We are met on a great battlefield of that war. We have come to dedicate a portion of that field, as a final resting place for those who here gave their lives that that nation might live. It is altogether fitting and proper that we should do this. But, in a larger sense, we can not dedicate, we can not consecrate, we can not hallow this ground. The brave men, living and dead, who struggled here, have consecrated it, far above our poor power to add or detract. The world will little note, nor long remember what we say here, but it can never forget what they did here. It is for us the living, rather, to be dedicated here to the unfinished work which they who fought here have thus far so nobly advanced. It is rather for us to be here dedicated to the great task remaining before us—that from these honored dead we take increased devotion to that cause for which they gave the last full measure of devotion—that we here highly resolve that these dead shall not have died in vain—that this nation, under God, shall have a new birth of freedom—and that government of the people, by the people, for the people, shall not perish from the earth.	Copied from Wikipedia (http://en.wikipedia.org/wiki/Gettysburg_Address)

***Figure 25** – A Resource file contains strings with a unique name that can be easily loaded into a text control later.*

The string is loaded by its name using a ResourceLoader object.

C# Code snippet:

```
//Add directive: using Windows.ApplicationModel.Resources
var rl = new ResourceLoader(("SpeechResources");
speech.Text = rl.GetString("GettysburgAddress");
```

VB Code snippet:

```
'Add directive: Imports Windows.ApplicationModel.Resources
 Dim rl As ResourceLoader = New ResourceLoader("SpeechResources")
 speech.Text = rl.GetString("GettysburgAddress")
```

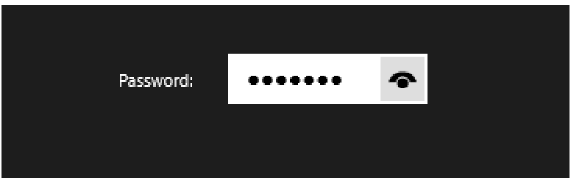
Other Text Controls

You may wish to explore the following additional controls through reference book or online resources.

The RichEditBox performs much like the Textbox control, but with added support for rich-formatted text. Thus if you wanted to creates some type of text editor with specialized formatting, the RichEditBox would be the choice control for doing so.

The PasswordBox provides for text entry whose characters are displayed as small black circles to protect the input value from other observers. One key property is the IsPasswordRevealButtonEnabled attribute which, if set to true, allows the user to review the actual input text by holding down a reveal button at the right of the control. This is helpful on mobile devices where the user may be using a more-challenging, on-screen virtual keyboard.

```
<TextBlock HorizontalAlignment="Left" Margin="184,409,0,0" TextWrapping="Wrap"
           Text="Password:" VerticalAlignment="Top"/>
<PasswordBox HorizontalAlignment="Left" Margin="254,398,0,0" VerticalAlignment="Top"
             Width="128" IsPasswordRevealButtonEnabled="True"/>
```



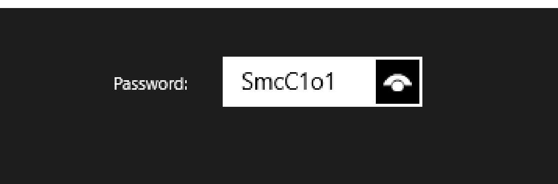


***Figure 26** – When the IsPasswordRevealButtonEnabled property is set to true, the PasswordBox control contains an eye button that when tapped reveals the actual characters entered into the PasswordBox.*

Image Control

The Image control is used for displaying bitmap images. In Lesson Two, you learned that it is advantageous to have three different image sizes (100%, 140%, and 180%) to achieve the best scaling on devices with different resolutions.

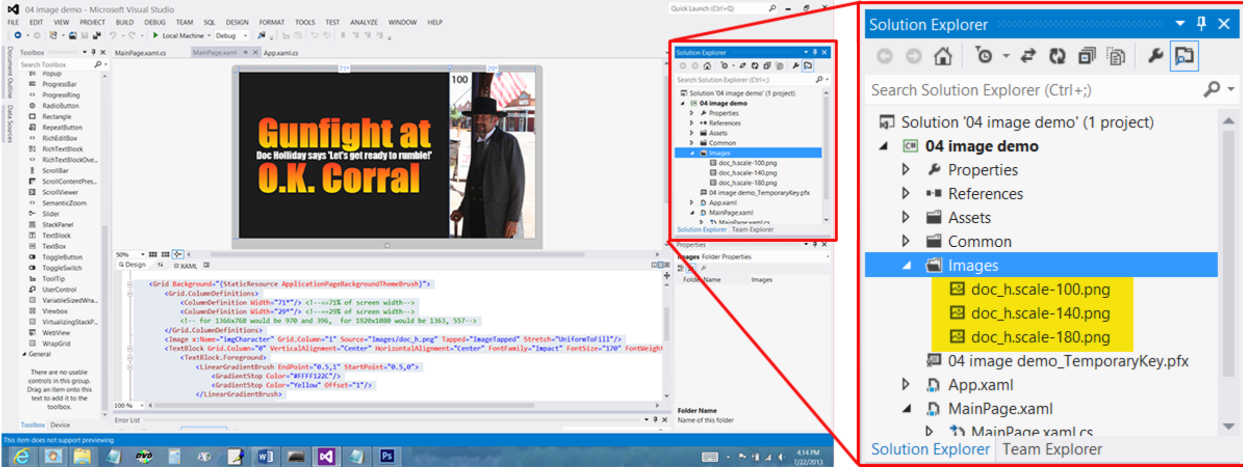


Figure 27 *– A folder can be added by right-clicking on the Solution Explorer and choosing Add > New Folder. Name the folder "Images." Then right-click on the Images folder and choose Add > Existing Item. Navigate to the image source and select the images(s) to be embedded in the project.*

The Image control utilizes a Source property to select the file. The source may be embedded as a full path if importing from a local directory (like the example below) or as a full URL if importing from an online source.

```
<Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="71*"/> <!--==71% of screen width-->
        <ColumnDefinition Width="29*"/> <!--==29% of screen width-->
        <!-- for 1366x768 would be 970 and 396,  for 1920x1080 would be 1363, 557-->
    </Grid.ColumnDefinitions>
    <Image x:Name="imgCharacter" Grid.Column="1" Source="Images/doc_h.png"
            Stretch="UniformToFill"/>
    <TextBlock Grid.Column="0" VerticalAlignment="Center"
                HorizontalAlignment="Center" FontFamily="Impact" FontSize="170"
                FontWeight="ExtraBold">
        <TextBlock.Foreground>
            <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                <GradientStop Color="#FFFF122C"/>
                <GradientStop Color="Yellow" Offset="1"/>
            </LinearGradientBrush>
        </TextBlock.Foreground>
        <Run Text="Gunfight at"/><LineBreak/><Run Text="O.K. Corral" />
    </TextBlock>
    <TextBlock VerticalAlignment="Center" HorizontalAlignment="Center"
                FontFamily="Impact" FontSize="45"
               Text="Doc Holliday says 'Let's get ready to rumble!'" Margin="0"/>
</Grid>
```

Figure 28 *– You can test the auto image resolution in the Simulator. The project is displayed on a simulated 1280x800 device on the left and a simulated 1920x1080 device on the right. The left device uses the 100 scale version and the right uses the 140 scale version of the graphic.* ***TIP:*** *To test, you must change the resolution of the simulator, end the project, and re-launch it to display the correct scale. In this example, the 100 and 140 numbers are part of the graphic.*

One of the key properties of the Image control is its Stretch attribute. There are four options for Stretch:

- None: The image displays in its native resolution.
- Fill: The image fills the Image control. It may stretch nonproportionally.
- Uniform: The image fills the Image control, but stretches proportionally.
- UniformToFill: The image fills the Image control and crops as necessary.

The following project demonstrates the effect of the Stretch attribute. One bitmap is loaded into four different Image controls, each with a different Stretch value.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="400"/>
    </Grid.ColumnDefinitions>
    <Button x:Name="backButton" Click="GoBack" IsEnabled="{Binding Frame.CanGoBack,
           ElementName=pageRoot}" Style="{StaticResource BackButtonStyle}"/>
    <TextBlock x:Name="pageTitle" Grid.Column="1" Text="{StaticResource AppName}"
               Style="{StaticResource PageHeaderTextStyle}"/>
    <TextBlock Grid.Column="2" Margin="0,50,0,0" FontWeight="Bold" FontSize="16">
        <Run>CIS165DB - Windows 8 Mobile Programming</Run><LineBreak/>
        <Run>South Mountain Community College</Run><LineBreak/>
        <Run>Instructor: Stephen Hustedde</Run>
    </TextBlock>
</Grid>
<Grid Grid.Row="1">
    <Grid.RowDefinitions>
        <RowDefinition Height="25" />
        <RowDefinition Height="*" />
         <RowDefinition Height="75" />
     </Grid.RowDefinitions>
     <Grid.ColumnDefinitions >
         <ColumnDefinition Width="1*"/>
         <ColumnDefinition Width="2*"/>
         <ColumnDefinition Width="1*"/>
```

```
        <ColumnDefinition Width="2*"/>
        <ColumnDefinition Width="1*"/>
        <ColumnDefinition Width="2*"/>
        <ColumnDefinition Width="1*"/>
        <ColumnDefinition Width="2*"/>
        <ColumnDefinition Width="1*"/>
    </Grid.ColumnDefinitions>
    <Image Grid.Row="1" Grid.Column="1" Source="Images/imageDemo.png" Stretch="None" />
    <Rectangle Grid.Row="1" Grid.Column="1" Stroke="White" StrokeThickness="5"/>
    <TextBlock Grid.Row="2" Grid.Column="1" FontSize="18"
             FontWeight="Bold"TextWrapping="Wrap"  Text='Stretch="None"'
             HorizontalAlignment="Center" VerticalAlignment="Center"/

    <Image Grid.Row="1" Grid.Column="3" Source="Images/imageDemo.png"
             Stretch="Fill"/>
    <Rectangle Grid.Row="1" Grid.Column="3" Stroke="White" StrokeThickness="5" />
    <TextBlock Grid.Row="2" Grid.Column="3" FontSize="18" FontWeight="Bold”
             TextWrapping="Wrap" Text='Stretch="Fill"'
             HorizontalAlignment="Center" VerticalAlignment="Center"/>

    <Image Grid.Row="1" Grid.Column="5" Source="Images/imageDemo.png"
             Stretch="Uniform"/>
    <Rectangle Grid.Row="1" Grid.Column="5" Stroke="White" StrokeThickness="5" />
    <TextBlock Grid.Row="2" Grid.Column="5" FontSize="18" FontWeight="Bold"
             TextWrapping="Wrap" HorizontalAlignment="Center"
             VerticalAlignment="Center">
             <Run>Stretch="Uniform"</Run><LineBreak/><Run>(Default value)</Run>
    </TextBlock>

    <Image Grid.Row="1" Grid.Column="7" Source="Images/imageDemo.png"
           Stretch="UniformToFill"/>
    <Rectangle Grid.Row="1" Grid.Column="7" Stroke="White" StrokeThickness="5" />
    <TextBlock Grid.Row="2" Grid.Column="7" FontSize="18" FontWeight="Bold"
             TextWrapping="Wrap" Text='Stretch="UniformToFill"'
             HorizontalAlignment="Center" VerticalAlignment="Center"/>
</Grid>
```

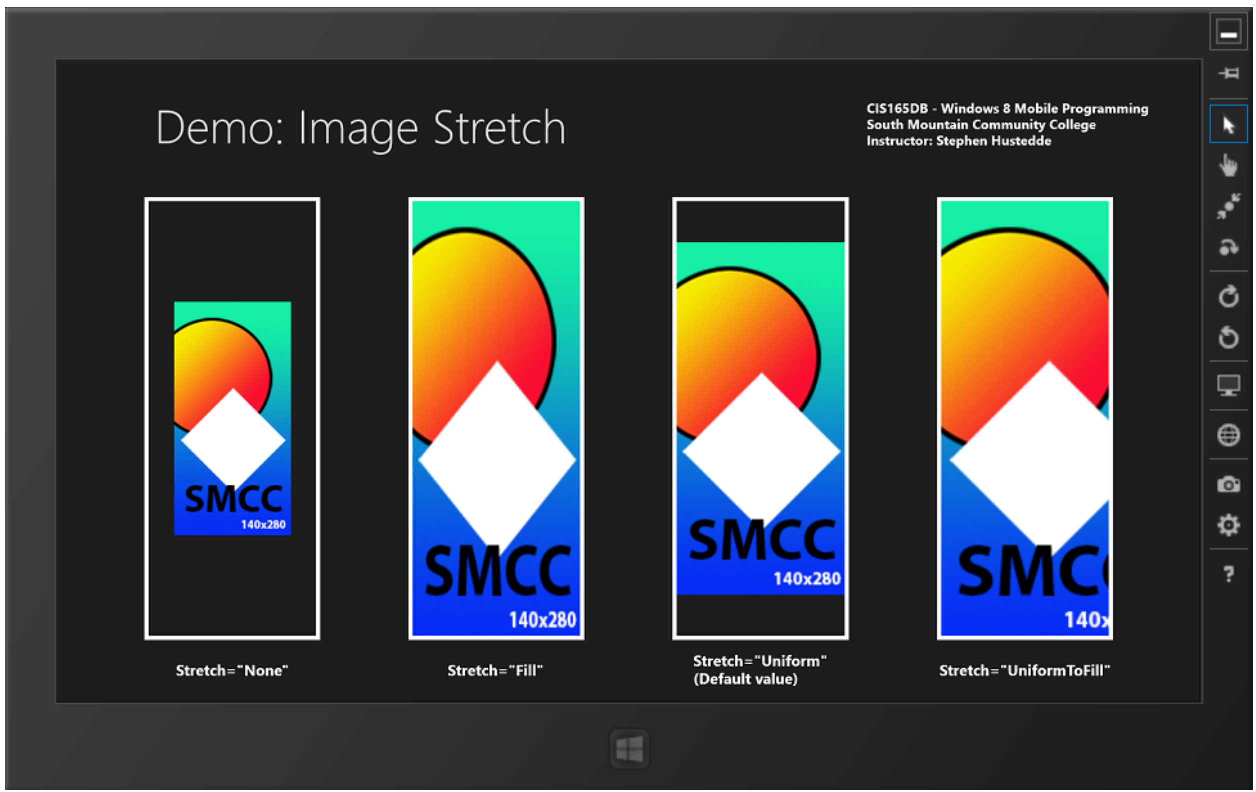


***Figure 29** – The same image displayed with different Stretch properties. From left to right: None; Fill, Uniform, UniformToFill.*

Images can be loaded programmatically by creating a BitmapImage object, specifying its Uri name/path and then setting the Image controls Source property to the BitmapImage. You will need to add a directive of "using Windows.UI.Xaml.Media.Imaging;".

C# Code:

```
private void imgOneTapped(object sender, TappedRoutedEventArgs e)
{
    //Uri myUri = new Uri("http://www.tricalico.com/test.jpg");
    // The "ms-appx:///" prefix in the URI below refers to the app's install directory.
    Uri myUri = new Uri("ms-appx:///Images/doc_h.png");
    BitmapImage bmi = new BitmapImage(myUri);
    imgTest.Source = bmi;
}
```

VB Code:

```
    Private Async Sub SubmitTapped(sender As Object, e As TappedRoutedEventArgs)
                                          Handles btnSubmit.Tapped
        'Dim myUri as Uri = new Uri("http://www.tricalico.com/test.jpg");'
        'The "ms-appx:///" prefix in the URI below refers to the app's install directory.
        Dim myUri As Uri = New Uri("ms-appx:///Images/doc_h.png")
        Dim bmi As BitmapImage = New BitmapImage(myUri)
        imgTest.Source = bmi
    End Sub
```

Buttons

Buttons work as they did in Windows desktop applications. One principle difference is that the caption of the button is displayed via the Context attribute rather than Text.

Add an event handler via the Properties panel by clicking the 'lightning bolt' tab to access the events for the control.

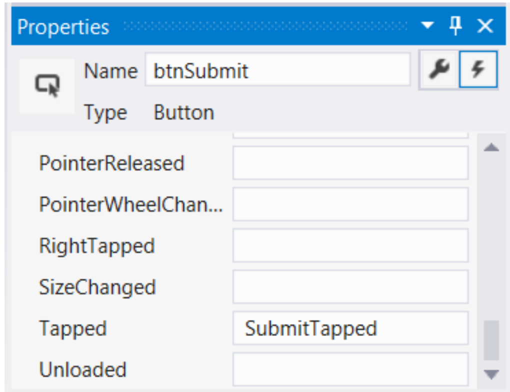


Figure 30 *– Add an event handler to the button in the same manner as you did with desktop applications.*

XAML Code (to display button):

```
  <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
       <TextBox x:Name="txtName" HorizontalAlignment="Left" Margin="163,160,0,0"
          TextWrapping="Wrap" Text="John Doe" VerticalAlignment="Top" Width="155"/>
       <Button x:Name="btnSubmit" Content="Submit" HorizontalAlignment="Left"
          Margin="345,160,0,0" VerticalAlignment="Top" Tapped="SubmitTapped" />
   </Grid>
```

C# Code: (to handle the Tapped event)

```
private async void SubmitTapped(object sender, TappedRoutedEventArgs e)
{
    string myName = txtName.Text;
    // Add directive:  using Windows.UI.Popups;
    MessageDialog md = new MessageDialog("Welcome to my app, " + myName + "!","Welcome");
    await md2.ShowAsync();
}
```

VB Code: (to handle the Tapped event)

```
    Private Async Sub SubmitTapped(sender As Object, e As TappedRoutedEventArgs)
                                Handles btnSubmit.Tapped
        Dim myName As String = txtName.Text
        ' Add directive:  imports Windows.UI.Popups
        Dim md As MessageDialog =
              New MessageDialog("Welcome to my app, " & myName & "!", "Welcome")
        Await md.ShowAsync()
    End Sub
```

The above code displays a MessageDialog object when the user taps (or clicks) the Submit button.

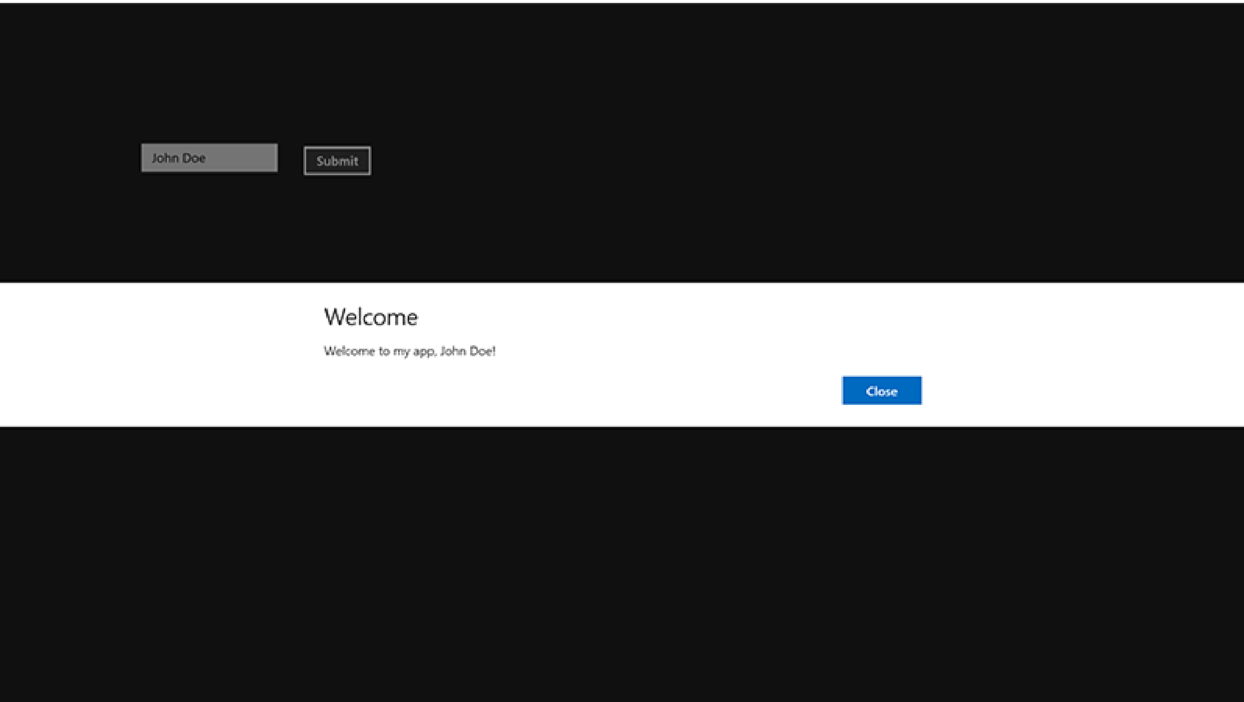


Figure 31 *– Screenshots from code execution for a button, which was demonstrated in the above example. A textbox and button are in the upper left.*

RadioButtons

RadioButtons are mutually exclusive within a grid or panel container and function the same as RadioButtons in Windows desktop applications. As with Buttons, the caption of the RadioButton is the Content property.

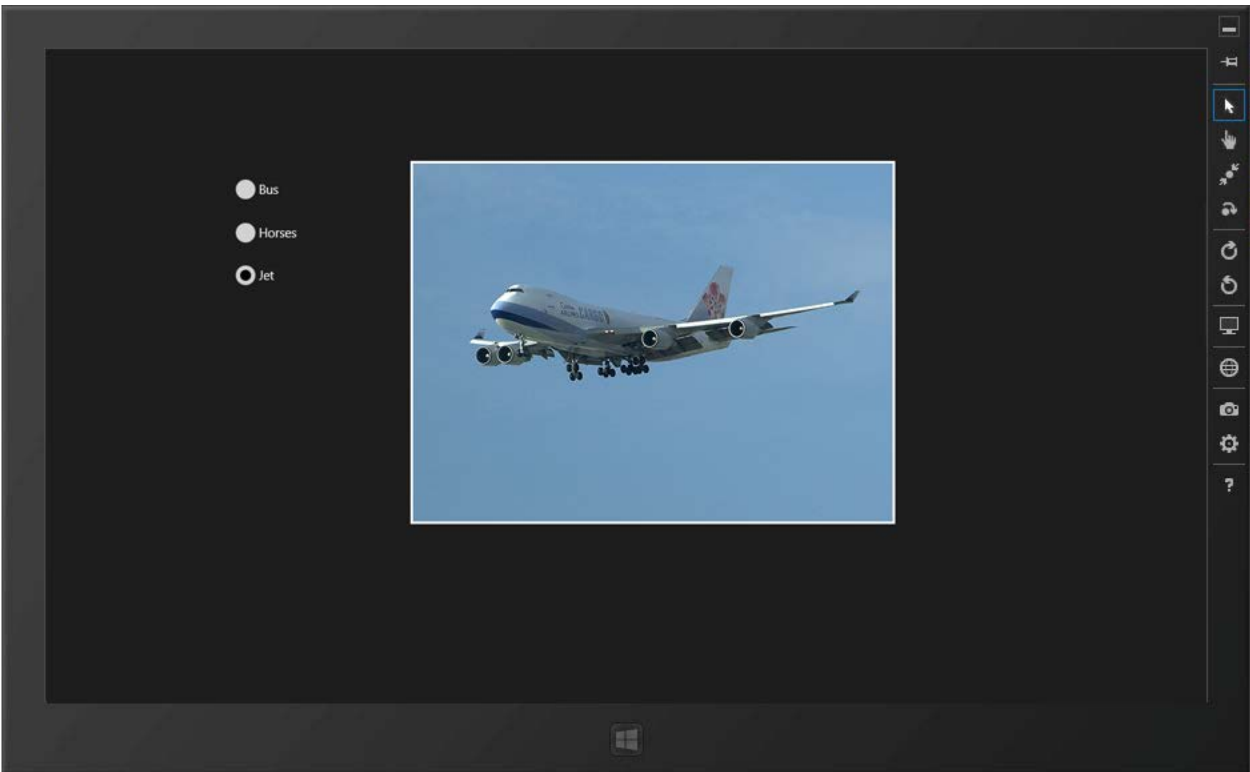


Figure 32 *– RadioButton controls are mutually exclusive within a container. Only one may be selected at any given time. The [Historical Bus] photo by Petr Kratochvil is from PublicDomainPictures.net and is in the public domain. The [Horses] photo by Petr Kratochvil is from PublicDomainPictures.net and is in the public domain. The [Airplanes_jets.jpg] by Jon Sullivan is from Wikimedia and is in the public domain.*

XAML Code of the above app

```
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <RadioButton x:Name ="rbBus" Content="Bus" HorizontalAlignment="Left"
                     Margin="222,149,0,0" VerticalAlignment="Top" Width="130"
                     RenderTransformOrigin="0.5,0.5" Checked="ShowImage"/>
        <RadioButton x:Name ="rbHorses" Content="Horses" HorizontalAlignment="Left"
                     Margin="222,200,0,0" VerticalAlignment="Top" Width="130"
                     RenderTransformOrigin="0.5,0.5" Checked="ShowImage"/>
        <RadioButton x:Name="rbJet" Content="Jet" HorizontalAlignment="Left"
                     Margin="222,250,0,0" VerticalAlignment="Top" Width="130"
                     RenderTransformOrigin="0.5,0.5" Checked="ShowImage"/>
        <Image x:Name="myImage" HorizontalAlignment="Left" Height="427"
                   Margin="430,130,0,0" VerticalAlignment="Top" Width="569"/>
        <Rectangle HorizontalAlignment="Left" Height="427" Margin="430,130,0,0"
                   VerticalAlignment="Top" Width="569" Stroke="#FFFFFF"
                   StrokeThickness="3"/>
    </Grid>
```

Programmatically, it is easiest to handle the Checked event, sharing the same handler and processing the name of the checked RadioButton.

C# Code:

```
        private void ShowImage(object sender, RoutedEventArgs e)
        {
```

```
            RadioButton rb = (RadioButton)sender;
            string imageFile;
            switch (rb.Name)
            {
                case "rbBus":
                    imageFile = "ms-appx:///Images/bus.png";
                    break;
                case "rbHorses":
                    imageFile = "ms-appx:///Images/horses.png";
                    break;
                case "rbJet":
                    imageFile = "ms-appx:///Images/jet.png";
                    break;
                default:
                    imageFile = "ms-appx:///Images/jet.png";
                    break;
            }
            Uri myUri = new Uri(imageFile);
            BitmapImage bmi = new BitmapImage(myUri);
            myImage.Source = bmi;
        }
```

VB Code:

```
    Private Sub ShowImage(sender As Object, e As RoutedEventArgs)
        Dim target As RadioButton = CType(sender, RadioButton)
        Dim myUri As Uri
        Dim bmi As BitmapImage
        Select Case target.Name
            Case "rbBus"
                myUri = New Uri("ms-appx:///Images/bus.png")
            Case "rbHorses"
                myUri = New Uri("ms-appx:///Images/horses.png")
            Case "rbJet"
                myUri = New Uri("ms-appx:///Images/jet.png")
            Case Else
                myUri = New Uri("ms-appx:///Images/jet.png")
        End Select
        bmi = New BitmapImage(myUri)
        myImage.Source = bmi
    End Sub
```

CheckBoxes

Similar to how RadioButtons function, checkboxes work as their desktop application counterparts. They are not mutually exclusive and the user may make multiple selections.

Programmatically, you might handle the Checked event, the Unchecked event, or the Tapped event. The Tapped Event is useful for handling both checked and unchecked states using an *if* structure, which you can see in the following example.

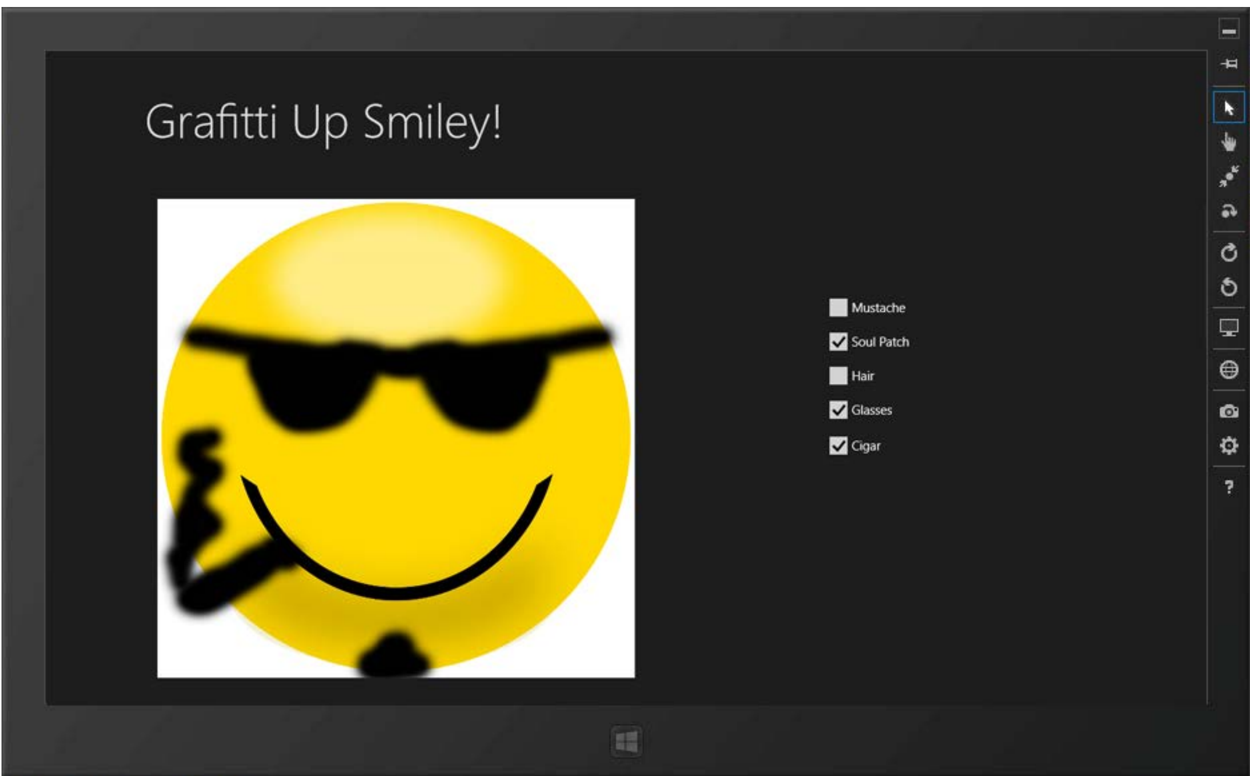


***Figure 33** – CheckBox controls are not mutually exclusive. The user may make multiple selections. The adapted [Smiley]() photo, originally by Paul Lloyd, is from PublicDomainPictures.net and is in the public domain.*

In the demonstrated example (in the figure above), all five checkboxes respond to a Tapped event and one procedure, ShowHide, which handles all five textboxes by evaluating the targeted object (passed as sender). Note that the approach used here was to have six images stacked on top of each other. The base image of smile is always visible. Each of the other images contain transparent PNG files whose Visibility property is initially set to "Collapsed" to keep it from being viewed. The underlying code simply toggles the Visibility for the appropriate graphic between Visible and Collapsed.

XAML Code

```
<Grid.RowDefinitions>
    <RowDefinition Height="140"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>

<!-- Back button and page title -->
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Button x:Name="backButton" Click="GoBack" IsEnabled="{Binding Frame.CanGoBack,
        ElementName=pageRoot}" Style="{StaticResource BackButtonStyle}"/>
    <TextBlock x:Name="pageTitle" Grid.Column="1" Text="{StaticResource AppName}"
               Style="{StaticResource PageHeaderTextStyle}"/>
</Grid>
```

```
<Image HorizontalAlignment="Left" Height="562" Margin="133,32,0,0" Grid.Row="1"
       VerticalAlignment="Top" Width="562" Source="Images/smiley-base2.png"/>
<Image x:Name="imgMustache" HorizontalAlignment="Left" Height="562"
       Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
       Source="Images/smiley-mustache.png" Visibility="Collapsed" />
<Image x:Name="imgSoulpatch" HorizontalAlignment="Left" Height="562"
       Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
       Source="Images/smiley-soulpatch.png" Visibility="Collapsed" />
<Image x:Name="imgHair" HorizontalAlignment="Left" Height="562"
       Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
       Source="Images/smiley-hair.png" Visibility="Collapsed" />
<Image x:Name="imgSunglasses" HorizontalAlignment="Left" Height="562"
       Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
       Source="Images/smiley-sunglasses.png" Visibility="Collapsed" />
<Image x:Name="imgCigar" HorizontalAlignment="Left" Height="562"
       Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
       Source="Images/smiley-cigar.png" Visibility="Collapsed" />
<CheckBox x:Name="cbxMustache" Content="Mustache" HorizontalAlignment="Left"
          Margin="919,146,0,0" Grid.Row="1" VerticalAlignment="Top" Height="32"
          Width="110" Tapped="ShowHide"/>
<CheckBox x:Name="cbxSoulpatch" Content="Soul Patch" HorizontalAlignment="Left"
      Margin="919,186,0,0" Grid.Row="1" VerticalAlignment="Top" Height="32" Width="110"
      Tapped="ShowHide"/>
<CheckBox x:Name="cbxHair" Content="Hair" HorizontalAlignment="Left"
      Margin="919,226,0,0" Grid.Row="1" VerticalAlignment="Top" Height="32" Width="110"
      Tapped="ShowHide"/>
<CheckBox x:Name="cbxSunglasses" Content="Glasses" HorizontalAlignment="Left"
      Margin="919,266,0,0" Grid.Row="1" VerticalAlignment="Top" Height="32" Width="110"
      Tapped="ShowHide"/>
<CheckBox x:Name="cbxCigar" Content="Cigar" HorizontalAlignment="Left"
      Margin="919,308,0,0" Grid.Row="1" VerticalAlignment="Top" Height="32" Width="110"
      Tapped="ShowHide"/>
```

C# Code

```
      private void ShowHide(object sender, TappedRoutedEventArgs e)
       {
           CheckBox cbx = (CheckBox)sender;
           Image img;
           switch (cbx.Name)
           {
               case "cbxMustache":
                   img = imgMustache;
                   break;
               case "cbxSoulpatch":
                   img = imgSoulpatch;
                   break;
               case "cbxHair":
                   img = imgHair;
                   break;
               case "cbxSunglasses":
                   img = imgSunglasses;
                   break;
               case "cbxCigar":
                   img = imgCigar;
                   break;
               default:
                   img = imgHair;
```

```
                break;
            }
            if (img.Visibility == Visibility.Visible)
            {
                img.Visibility = Visibility.Collapsed;
            }
            else
            {
                img.Visibility = Visibility.Visible;
            }
        }
```

VB Code

```
    Private Sub ShowHide(sender As Object, e As TappedRoutedEventArgs) _
            Handles cbxCigar.Tapped, cbxSunglasses.Tapped, cbxHair.Tapped,
                     cbxSoulpatch.Tapped, cbxMustache.Tapped
        Dim cbx As CheckBox = CType(sender, CheckBox)
        Dim img As Image
        Select Case cbx.Name
            Case "cbxMustache"
                img = imgMustache
            Case "cbxSoulpatch"
                img = imgSoulpatch
            Case "cbxHair"
                img = imgHair
            Case "cbxSunglasses"
                img = imgSunglasses
            Case "cbxCigar"
                img = imgCigar
            Case Else
                img = imgHair
        End Select
        If img.Visibility = Visibility.Visible Then
            img.Visibility = Visibility.Collapsed
        Else
            img.Visibility = Visibility.Visible
        End If
    End Sub
```

ToggleSwitches

In the previous CheckBox demo, you saw that checkboxes were used to toggle the display of images. For that app, a ToggleSwitch control could be an appropriate choice instead of the CheckBoxes. The user interacts with the Toggle switch by dragging the switch to the on (right) or off (left) position. Key Properties for the ToggleSwitch are the OffContent and OnContent attributes to display the current state of the Switch. These are “Off” and “On” by default, but may be customized to other options, such as “Hidden” and “Shown”. The Toggled event is the most likely event to handle, using a conditional structure to handle the IsOn state (Boolean value).

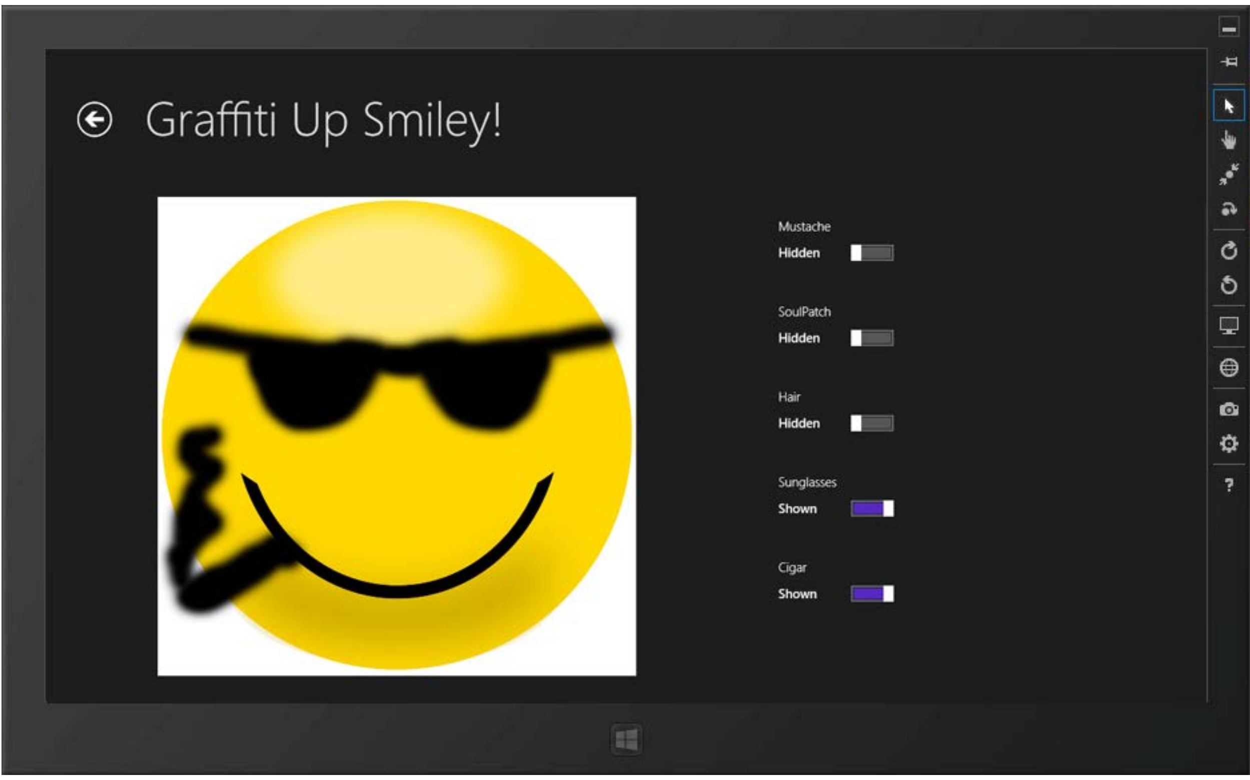


Figure 34 *– The Checkbox controls were replaced with ToggleSwitch controls. The adapted* *Smiley* *photo, originally by Paul Lloyd, is from PublicDomainPictures.net and is in the public domain.*

XAML Code:

```
<Grid Style="{StaticResource LayoutRootStyle}" Tapped="MustacheTapped">
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!-- Back button and page title -->
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Button x:Name="backButton" IsEnabled="{Binding Frame.CanGoBack,
                ElementName=pageRoot}" Style="{StaticResource BackButtonStyle}"/>
            <TextBlock x:Name="pageTitle" Grid.Column="1" Text="Graffiti Up Smiley!"
                       Style="{StaticResource PageHeaderTextStyle}"/>
        </Grid>
        <Image HorizontalAlignment="Left" Height="562" Margin="133,32,0,0" Grid.Row="1"
              VerticalAlignment="Top" Width="562" Source="Images/smiley-base2.png" />
        <Image x:Name="imgMustache" HorizontalAlignment="Left" Height="562"
              Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
              Source="Images/smiley-mustache.png" Visibility="Collapsed" />
        <Image x:Name="imgSoulpatch" HorizontalAlignment="Left" Height="562"
              Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
              Source="Images/smiley-soulpatch.png" Visibility="Collapsed" />
        <Image x:Name="imgHair" HorizontalAlignment="Left" Height="562"
```

```
            Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
            Source="Images/smiley-hair.png" Visibility="Collapsed" />
        <Image x:Name="imgSunglasses" HorizontalAlignment="Left" Height="562"
            Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
            Source="Images/smiley-sunglasses.png" Visibility="Collapsed" />
        <Image x:Name="imgCigar" HorizontalAlignment="Left" Height="562"
            Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
            Source="Images/smiley-cigar.png" Visibility="Collapsed" />
        <ToggleSwitch Header="Mustache" HorizontalAlignment="Left"
            Margin="854,50,0,0" Grid.Row="1" VerticalAlignment="Top"
            OffContent="Hidden" OnContent="Shown" Toggled="MustacheToggled"/>
        <ToggleSwitch Header="SoulPatch" HorizontalAlignment="Left"
            Margin="854,150,0,0" Grid.Row="1" VerticalAlignment="Top"
            OffContent="Hidden" OnContent="Shown" Toggled="SoulpatchToggled"/>
        <ToggleSwitch Header="Hair" HorizontalAlignment="Left"
         Margin="854,250,0,0" Grid.Row="1" VerticalAlignment="Top"
         OffContent="Hidden" OnContent="Shown" Toggled="HairToggled" />
        <ToggleSwitch Header="Sunglasses" HorizontalAlignment="Left"
         Margin="854,350,0,0"  Grid.Row="1" VerticalAlignment="Top"
         OffContent="Hidden" OnContent="Shown" Toggled="SunglassesToggled" />
        <ToggleSwitch Header="Cigar" HorizontalAlignment="Left"
         Margin="854,450,0,0" Grid.Row="1" VerticalAlignment="Top"
         OffContent="Hidden" OnContent="Shown" Toggled="CigarToggled" />
    </Grid >
```

C# Code:

```
        private void MustacheToggled(object sender, RoutedEventArgs e)
        {
            ToggleSwitch ts = (ToggleSwitch)sender;
            if (ts.IsOn)
            {
                imgMustache.Visibility = Windows.UI.Xaml.Visibility.Visible;
            }
            else
            {
                imgMustache.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }

        private void SoulpatchToggled(object sender, RoutedEventArgs e)
        {
            ToggleSwitch ts = (ToggleSwitch)sender;
            if (ts.IsOn)
            {
                imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Visible;
            }
            else
            {
                imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }

        private void HairToggled(object sender, RoutedEventArgs e)
        {
            ToggleSwitch ts = (ToggleSwitch)sender;
            if (ts.IsOn)
            {
```

```
            imgHair.Visibility = Windows.UI.Xaml.Visibility.Visible;
        }
        else
        {
            imgHair.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
        }
    }

    private void SunglassesToggled(object sender, RoutedEventArgs e)
    {
        ToggleSwitch ts = (ToggleSwitch)sender;
        if (ts.IsOn)
        {
            imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Visible;
        }
        else
        {
            imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
        }
    }

    private void CigarToggled(object sender, RoutedEventArgs e)
    {
        ToggleSwitch ts = (ToggleSwitch)sender;
        if (ts.IsOn)
        {
            imgCigar.Visibility = Windows.UI.Xaml.Visibility.Visible;
        }
        else
        {
            imgCigar.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
        }
    }
```

VB Code:

```
    Private Sub MustacheToggled(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleSwitch = CType(sender, ToggleSwitch)
        If ts.IsOn Then
            imgMustache.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgMustache.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub SoulpatchToggled(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleSwitch = CType(sender, ToggleSwitch)
        If ts.IsOn Then
            imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub HairToggled(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleSwitch = CType(sender, ToggleSwitch)
        If ts.IsOn Then
            imgHair.Visibility = Windows.UI.Xaml.Visibility.Visible
```

```
        Else
            imgHair.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub SunglassesToggled(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleSwitch = CType(sender, ToggleSwitch)
        If ts.IsOn Then
            imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub CigarToggled(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleSwitch = CType(sender, ToggleSwitch)
        If ts.IsOn Then
            imgCigar.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgCigar.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub
```

ToggleButtons

A third option for displaying the various facial graffiti in the previous project is to use ToggleButton controls. ToggleButtons are push buttons that are highlighted when turned on (checked). The Tapped event can be handled and the IsChecked property evaluated in a conditional structure.

Figure 35 *–The ToggleSwitch controls have been replaced with ToggleButton controls. The adapted [Smiley]() photo, originally by Paul Lloyd, is from PublicDomainPictures.net and is in the public domain.*

XAML Code:

```
<Grid Style="{StaticResource LayoutRootStyle}" >
        <Grid.RowDefinitions>
            <RowDefinition Height="140"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <!-- Back button and page title -->
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto"/>
                <ColumnDefinition Width="*"/>
            </Grid.ColumnDefinitions>
            <Button x:Name="backButton1" IsEnabled="{Binding Frame.CanGoBack,
                    ElementName=pageRoot}" Style="{StaticResource BackButtonStyle}"/>
            <TextBlock x:Name="pageTitle1" Grid.Column="1" Text="Graffiti Up Smiley!"
                           Style="{StaticResource PageHeaderTextStyle}"/>
        </Grid>
        <Image HorizontalAlignment="Left" Height="562" Margin="133,32,0,0" Grid.Row="1"
               VerticalAlignment="Top" Width="562" Source="Images/smiley-base2.png" />
        <Image x:Name="imgMustache" HorizontalAlignment="Left" Height="562"
                     Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
                     Source="Images/smiley-mustache.png" Visibility="Collapsed" />
        <Image x:Name="imgSoulpatch" HorizontalAlignment="Left" Height="562"
                     Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
                     Source="Images/smiley-soulpatch.png" Visibility="Collapsed" />
        <Image x:Name="imgHair" HorizontalAlignment="Left" Height="562"
                     Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
                     Source="Images/smiley-hair.png" Visibility="Collapsed" />
        <Image x:Name="imgSunglasses" HorizontalAlignment="Left" Height="562"
                     Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
                     Source="Images/smiley-sunglasses.png" Visibility="Collapsed" />
        <Image x:Name="imgCigar" HorizontalAlignment="Left" Height="562"
                     Margin="133,32,0,0" Grid.Row="1" VerticalAlignment="Top" Width="562"
                     Source="Images/smiley-cigar.png" Visibility="Collapsed" />
        <ToggleButton Content="Mustache" HorizontalAlignment="Left"
                     Margin="860,32,0,0" Grid.Row="1" VerticalAlignment="Top"
                     Tapped="MustacheToggleTapped" Width="120" />
        <ToggleButton Content="Soul Patch" HorizontalAlignment="Left"
                     Margin="860,92,0,0" Grid.Row="1" VerticalAlignment="Top"
                     Tapped="SoulPatchToggleTapped" Width="120" />
        <ToggleButton Content="Hair" HorizontalAlignment="Left"
              Margin="860,152,0,0" Grid.Row="1" VerticalAlignment="Top"
              Tapped="HairToggleTapped" Width="120" />
        <ToggleButton Content="Sunglasses" HorizontalAlignment="Left"
              Margin="860,213,0,0" Grid.Row="1" VerticalAlignment="Top"
              Tapped="SunglassesToggleTapped" Width="120" />
        <ToggleButton Content="Cigar" HorizontalAlignment="Left"
              Margin="860,276,0,0" Grid.Row="1" VerticalAlignment="Top"
              Tapped="CigarToggleTapped" Width="120" />
    </Grid>
```

C# Code Snippet (to handle the ToggleButtons' Tapped events)

```
        private void MustacheToggleTapped(object sender, TappedRoutedEventArgs e)
        {
            ToggleButton tb = (ToggleButton)sender;
            if ((bool)tb.IsChecked)
            {
                imgMustache.Visibility = Windows.UI.Xaml.Visibility.Visible;
            }
            else
            {
                imgMustache.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }

        private void SoulPatchToggleTapped(object sender, TappedRoutedEventArgs e)
        {
            ToggleButton tb = (ToggleButton)sender;
            if ((bool)tb.IsChecked)
            {
                imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Visible;
            }
            else
            {
                imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }

        private void HairToggleTapped(object sender, TappedRoutedEventArgs e)
        {
            ToggleButton tb = (ToggleButton)sender;
            if ((bool)tb.IsChecked)
            {
                imgHair.Visibility = Windows.UI.Xaml.Visibility.Visible;
            }
            else
            {
                imgHair.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }

        private void SunglassesToggleTapped(object sender, TappedRoutedEventArgs e)
        {
            ToggleButton tb = (ToggleButton)sender;
            if ((bool)tb.IsChecked)
            {
                imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Visible;
            }
            else
            {
                imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }

        private void CigarToggleTapped(object sender, TappedRoutedEventArgs e)
        {
            ToggleButton tb = (ToggleButton)sender;
            if ((bool)tb.IsChecked)
            {
                imgCigar.Visibility = Windows.UI.Xaml.Visibility.Visible;
```

```
            }
            else
            {
                imgCigar.Visibility = Windows.UI.Xaml.Visibility.Collapsed;
            }
        }
```

VB Code (to handle the ToggleButtons' Tapped events):

```
    Private Sub MustacheToggleTapped(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleButton = CType(sender, ToggleButton)
        If ts.IsChecked Then
            imgMustache.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgMustache.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub SoulpatchToggleTapped(sender As Object, e As RoutedEventArgs)
       Dim ts As ToggleButton = CType(sender, ToggleButton)
        If ts.IsChecked Then
            imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgSoulpatch.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub HairToggleTapped(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleButton = CType(sender, ToggleButton)
        If ts.IsChecked Then
            imgHair.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgHair.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub SunglassesToggleTapped(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleButton = CType(sender, ToggleButton)
        If ts.IsChecked Then
            imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgSunglasses.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub

    Private Sub CigarToggleTapped(sender As Object, e As RoutedEventArgs)
        Dim ts As ToggleButton = CType(sender, ToggleButton)
        If ts.IsChecked Then
            imgCigar.Visibility = Windows.UI.Xaml.Visibility.Visible
        Else
            imgCigar.Visibility = Windows.UI.Xaml.Visibility.Collapsed
        End If
    End Sub
```

Creating and Applying Styles

Implicit Styles

Open up the new styles dictionary to begin creating an implicit style. Implicit styles do not contain a key name, but rather reference a target control type via the TargetType attribute.

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:_04_Styles">

    <!-- IMPLICIT STYLES-->

    <Style TargetType="Button">
        <Setter Property="Background" Value="Blue" />
        <Setter Property="Foreground" Value="White"/>
    </Style>

</ResourceDictionary>
```

On the MainPage of the app, create several buttons.

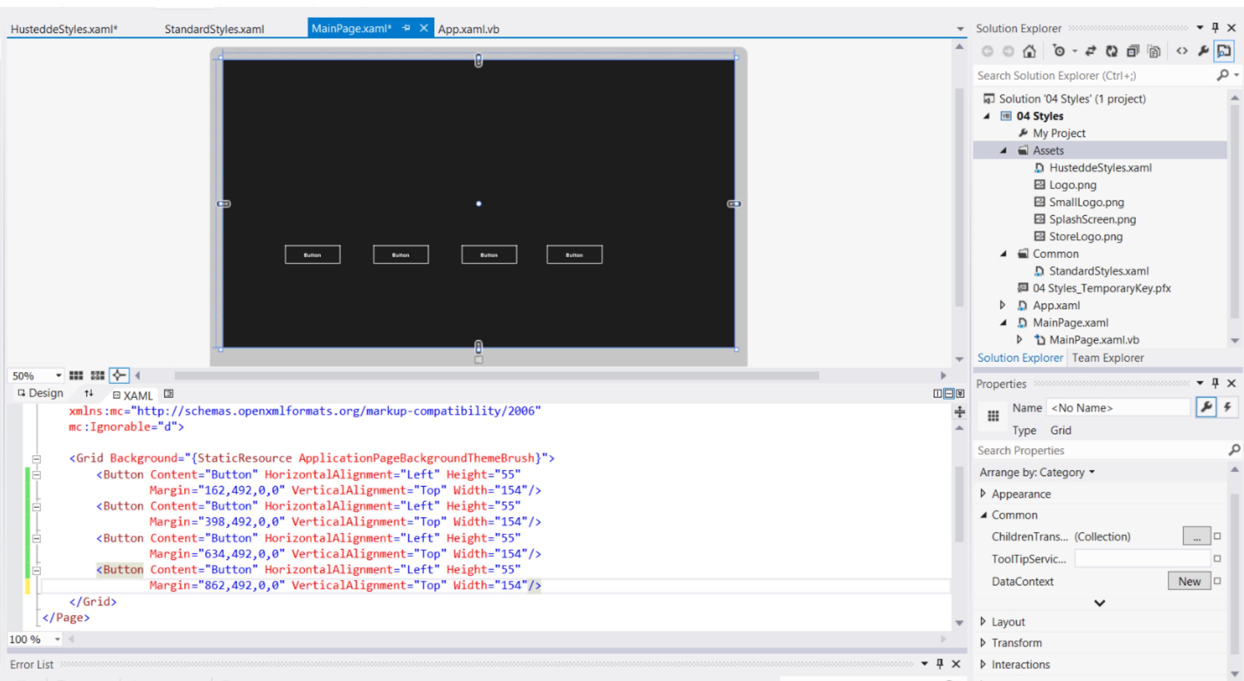


Figure 36 – Four buttons were added to the MainPage via XAML code. The XAML code is below.

XAML Code:

```
    <Grid Background="{StaticResource ApplicationPageBackgroundThemeBrush}">
        <Button Content="Button" HorizontalAlignment="Left" Height="55"
                Margin="162,492,0,0" VerticalAlignment="Top" Width="154"/>
        <Button Content="Button" HorizontalAlignment="Left" Height="55"
                Margin="398,492,0,0" VerticalAlignment="Top" Width="154"/>
```

```
        <Button Content="Button" HorizontalAlignment="Left" Height="55"
                Margin="634,492,0,0" VerticalAlignment="Top" Width="154"/>
        <Button Content="Button" HorizontalAlignment="Left" Height="55"
                Margin="862,492,0,0" VerticalAlignment="Top" Width="154"/>
    </Grid>
```

The buttons do not have the implicit style, because you first must reference the added dictionary resource in the *App.xaml* document.

App.xaml

```
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>

                <!--
                    Styles that define common aspects of the platform look and feel
                    Required by Visual Studio project and item templates
                 -->
                <ResourceDictionary Source="Common/StandardStyles.xaml"/>
                <ResourceDictionary Source="Assets/HusteddeStyles.xaml"/>
            </ResourceDictionary.MergedDictionaries>

        </ResourceDictionary>
    </Application.Resources>
```

With this step completed, the buttons of the app now have the implicit style.

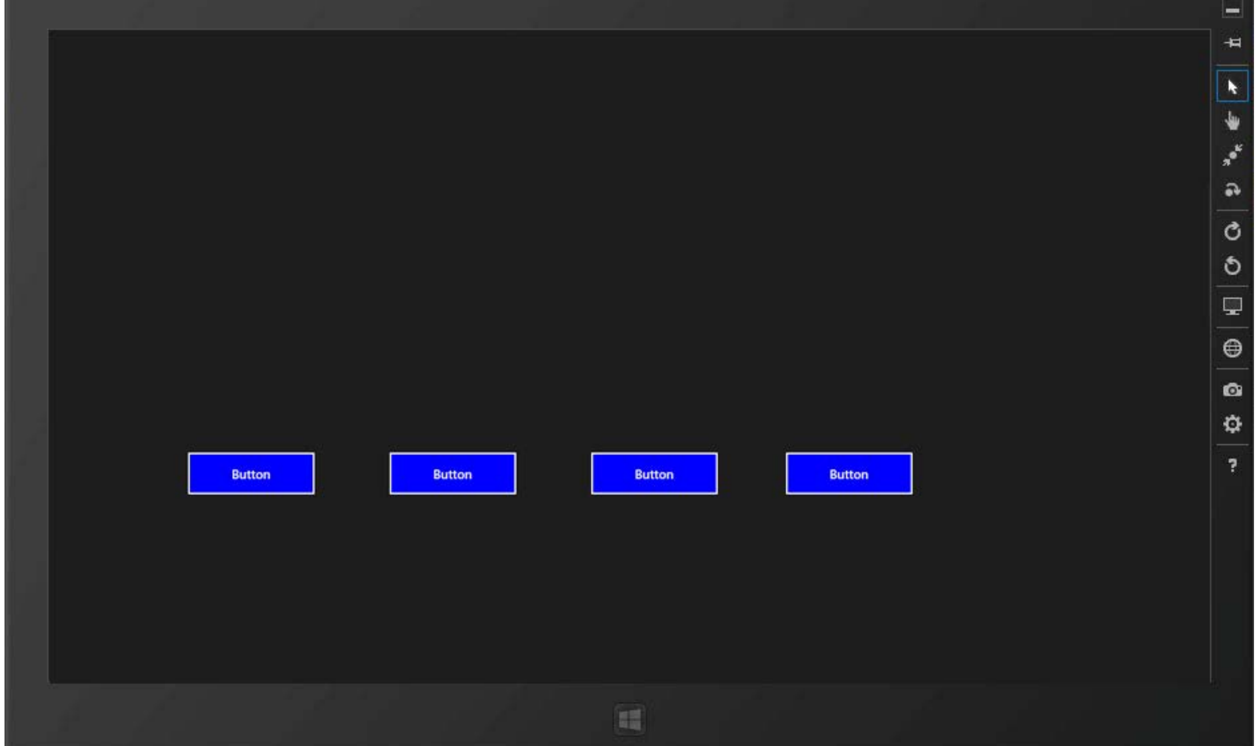


Figure 37 – The app's buttons now display the implicit style.

Next, add code to the style to handle the Pressed and Disabled states of buttons.

XAML Code

```
    <Style TargetType="Button">
        <Setter Property="Background" Value="Blue" />
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="Template">
            <Setter.Value>
            <ControlTemplate TargetType="Button">
                    <Grid Margin="{TemplateBinding Margin}">
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal"/>
                            <VisualState x:Name="PointerOver"/>
                            <VisualState x:Name="Pressed">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
                                              Storyboard.TargetName="Text"
                                              Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                                              Value="Yellow"/>
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Disabled">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
                                              Storyboard.TargetName="Text"

                                              Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
                                               Value="MediumGray"/>
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                        </VisualStateGroup>
                        </VisualStateManager.VisualStateGroups>
                    </Grid>
                </ControlTemplate>
        </Setter.Value>
        </Setter>
    </Style>
```

Within a page, you can define a template to give the object a new look and feel. In the following XAML code, an implicit button style is created that has a rounded rectangle form with a thicker stroke. The template is borrowed from the ButtonBase. The Rectangle and ContentPresenter controls were named ("bg" and "Caption") for the purposes of changing the button's appearance under various conditions.

```
<Style TargetType="Button">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ButtonBase">
                    <Grid >
                        <Rectangle x:Name="bg" Fill="Blue"
                            Stroke="White"
                            StrokeEndLineCap="Round"
                            Opacity="1"
```

```
                            StrokeThickness="5" RadiusX="20" RadiusY="20"/>
                        <ContentPresenter x:Name="Caption" Foreground="White"
                          Content="{TemplateBinding Content}"
                          HorizontalAlignment="Center"
                          VerticalAlignment="Center"/>
                    </Grid>
                </ControlTemplate>
             </Setter.Value>
        </Setter>
    </Style>
```

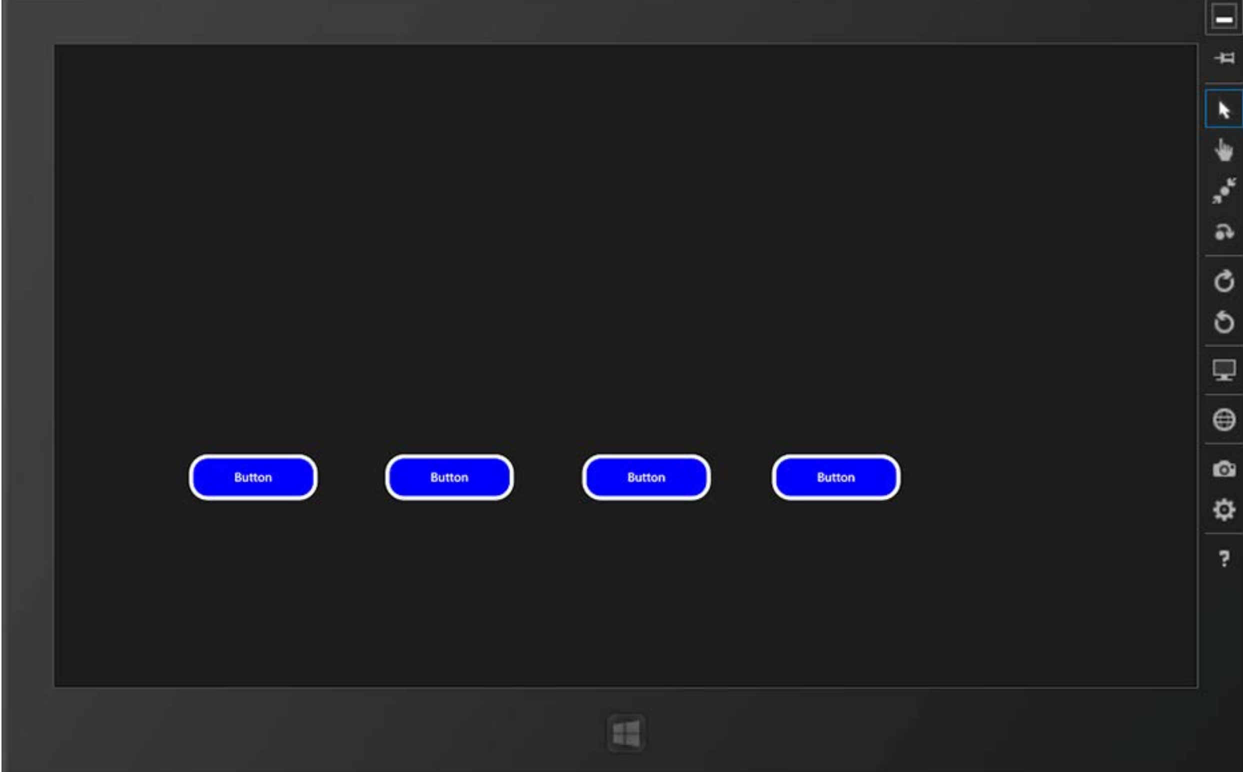


Figure 38 – The template for the implicit button style now utilizes a rounded rectangle.

By using a template with a Grid control, you also can place a VisualStatesManager element structure within the Grid to handle various states of the button, including MouseEnter, Pressed, Disabled, and more. KeyFrame-based animations are used to change the display similarly to how you changed device view states in Lesson 3.

```
    <Style TargetType="Button">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ButtonBase">
                    <Grid >
                        <Rectangle x:Name="bg" Fill="Blue"
                            Stroke="White"
                            StrokeEndLineCap="Round"
                            Opacity="1"
                            StrokeThickness="5" RadiusX="20" RadiusY="20"/>
                        <ContentPresenter x:Name="Caption" Foreground="White"
                          Content="{TemplateBinding Content}"
```

```
                                  HorizontalAlignment="Center"
                                  VerticalAlignment="Center"/>
                                <VisualStateManager.VisualStateGroups>
                                <VisualStateGroup x:Name="CommonStates">
                                    <VisualState x:Name="Normal"/>

                                    <VisualState x:Name="Pressed">
                                        <Storyboard>
                                            <ObjectAnimationUsingKeyFrames
                                                     Storyboard.TargetName="bg"
                                                     Storyboard.TargetProperty="Fill">
                                                <DiscreteObjectKeyFrame KeyTime="0"
                                                       Value="Yellow"/>
                                            </ObjectAnimationUsingKeyFrames>
                                            <ObjectAnimationUsingKeyFrames
                                                     Storyboard.TargetName="Caption"
                                                     Storyboard.TargetProperty="Foreground">
                                                 <DiscreteObjectKeyFrame KeyTime="0"
                                                        Value="Blue"/>
                                             </ObjectAnimationUsingKeyFrames>
                                        </Storyboard>
                                    </VisualState>
                                    <VisualState x:Name="Disabled">
                                        <Storyboard>
                                            <ObjectAnimationUsingKeyFrames
                                                     Storyboard.TargetName="bg"
                                                     Storyboard.TargetProperty="Fill">
                                                <DiscreteObjectKeyFrame KeyTime="0"
                                                       Value="MediumGray"/>
                                            </ObjectAnimationUsingKeyFrames>
                                            <ObjectAnimationUsingKeyFrames
                                                     Storyboard.TargetName="Caption"
                                                     Storyboard.TargetProperty="Foreground">
                                                 <DiscreteObjectKeyFrame KeyTime="0"
                                                        Value="Gray"/>
                                             </ObjectAnimationUsingKeyFrames>
                                        </Storyboard>
                                    </VisualState>
                                </VisualStateGroup>
                                </VisualStateManager.VisualStateGroups>

                            </Grid>
                        </ControlTemplate>
                     </Setter.Value>
                </Setter>
            </Style>
```

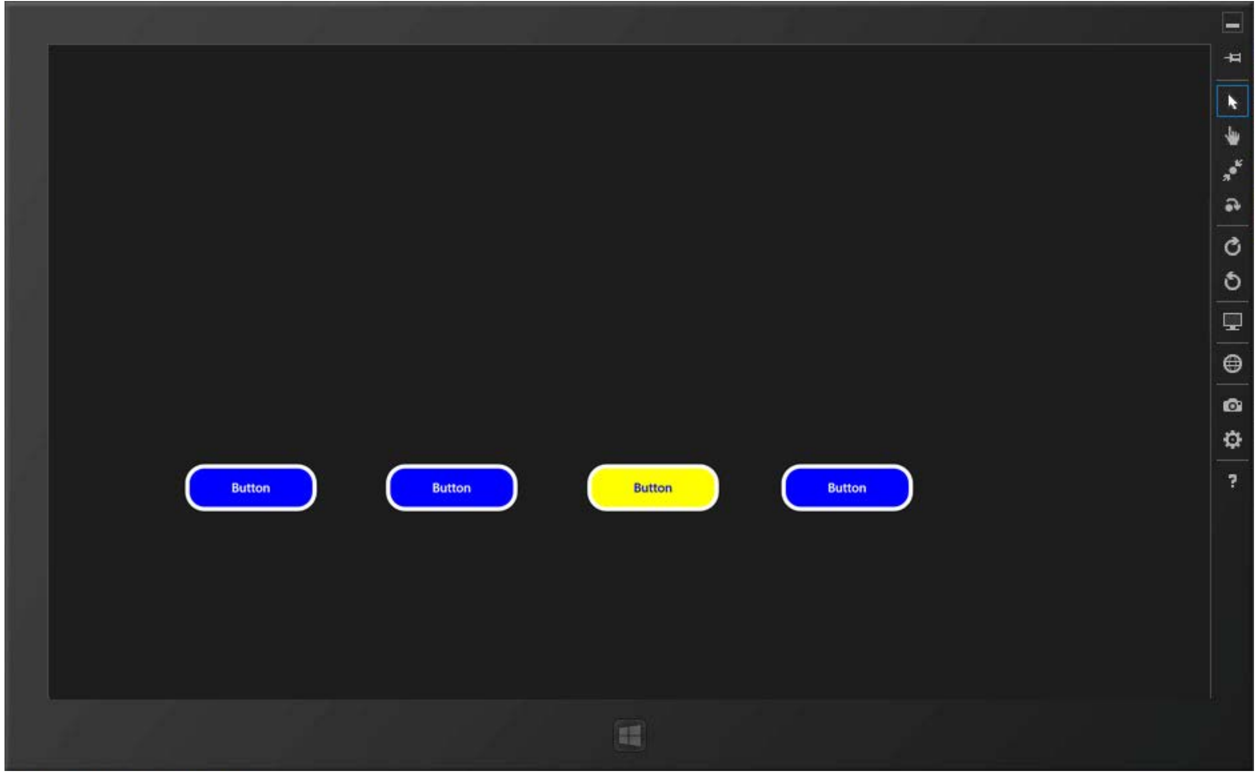


Figure 39 *– With the implicit button style utilizing the VisualStatesManager, the button can change appearances as directed, such as here when the user presses a button. The third button is being pressed. Its background turns yellow and the text goes from white to blue.*

Explicit Styles

While implicit styles are applied to any object of the specified type, explicit styles are named using an x:Key pair.

```
    <Style x:Key="RedButton"  TargetType="ButtonBase">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ButtonBase">
                    <Grid >
                        <Rectangle x:Name="bg" Fill="Red"
                            Stroke="White"
                            StrokeEndLineCap="Round"
                            Opacity="1"
                            StrokeThickness="5" RadiusX="20" RadiusY="20"/>
                        <ContentPresenter x:Name="Caption" Foreground="Yellow"
                          Content="{TemplateBinding Content}"
                          HorizontalAlignment="Center"
                          VerticalAlignment="Center"/>
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

An explicit style is applied individually to controls via the Style pair of its XAML code.

```
<Button Content="Button" HorizontalAlignment="Left" Height="55"
                Margin="162,492,0,0" VerticalAlignment="Top" Width="154"
                Style="{StaticResource RedButton}"/>
```

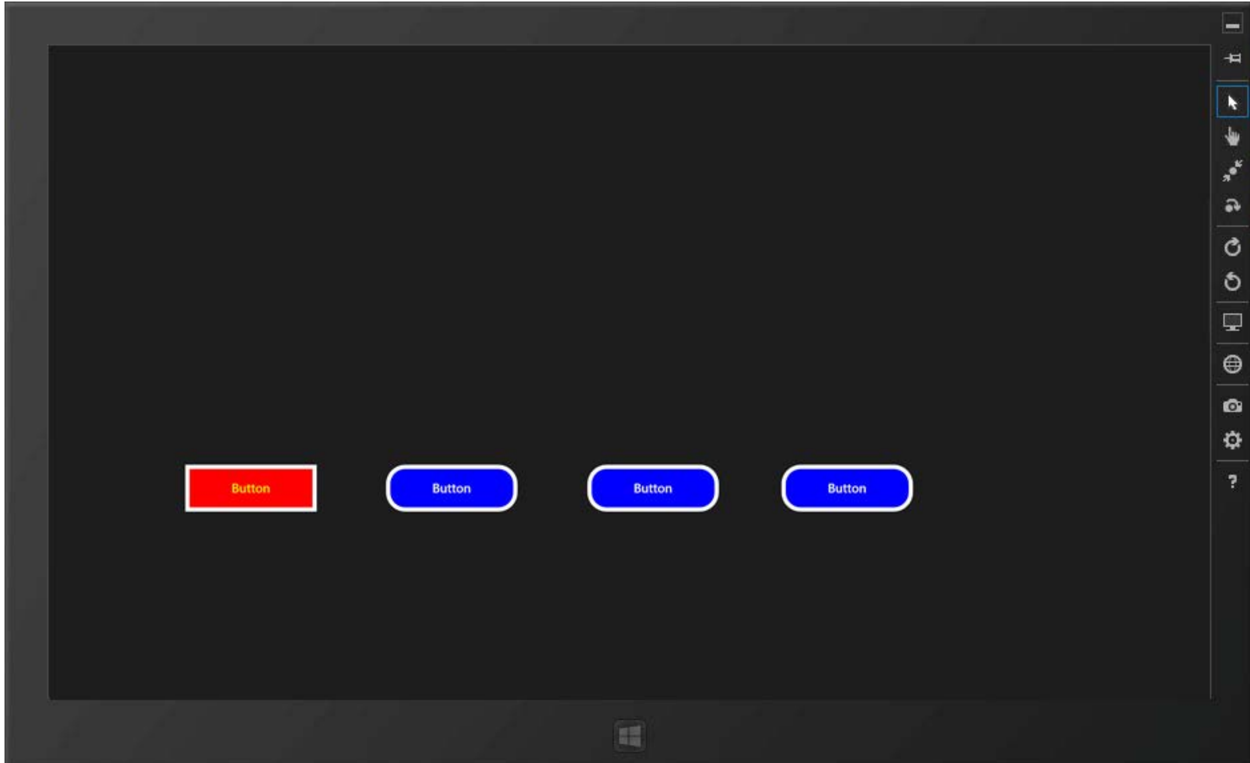


Figure 40 *– The explicit style applied to the first button on the left overrides the implicit style otherwise applied to all buttons.*

Programmatically, styles can be changed in the underlying code in response to events. Consider the addition of a “LimeGreenButton” in the ResourceDictionary and the leftmost button given a name of “btn1” in the XAML code. We could change its style from the explicitly applied “RedButton” style to the “LimeGreenStyle” upon clicking another buttons or handling some other event.

C# Code

```
    private void btnTapped(object sender, TappedRoutedEventArgs e)
    {
       Style myStyle = Application.Current.Resources["LimeGreenButton"] as Style;
       btn1.Style = myStyle;
    }
```

VB Code:

```
    Private Sub btnTapped(sender As Object, e As TappedRoutedEventArgs)
        Dim myStyle As Style = Application.Current.Resources("LimeGreenButton")
        btn1.Style = myStyle
    End Sub
```

A new style can be based on a previously defined style by referencing it in a BasedOn value-pair with a reference to the base style. For example:

```
<Style x:Key="MyNewStyleOne" TargetType="TextBlock" >
    <Setter Property="FontFace" Value="Impact"/>
    <Setter Property="FontSize" Value="44"/>
    <Setter Property="FontWeight" Value="Bold"/>
</Style>


<Style x:Key="MyNewStyleTwo" TargetType="TextBlock"
                         BasedOn="{StaticResource MyNewStyleOne}">
    <Setter Property="FontSize" Value="22"/>
    <Setter Property="Foreground" Value="Red"/>
</Style>
```

The MyNewStyleTwo incorporates all the attributes of the MyNewStyleOne, but its FontSize is overridden to be 22 instead of 44, and a Foreground attribute of Red is also added.