Lesson 3 Guide: Overview of Windows 8 App Development with Visual Studio 2012 and VB/C#

Table of Contents

Creating a Simple Windows Store App

Launch Visual Studio 2012 from Windows 8 Start Screen or Windows 8 Desktop— either way it will launch in desktop mode.

Creating a New Project

Step 1: Choose **File > New > Project**. In the New Project dialog, choose **Installed > Templates > Visual C# > Windows Store** or **Installed > Templates > Other Languages > Visual Basic > Windows Store** on the left, depending on whether you wish to program in C# or VB. Either way, you will be able to choose a template. In this course you will only be concerned with the first three options - Blank App (XAML), Grid App (XAML), and Split App (XAML).

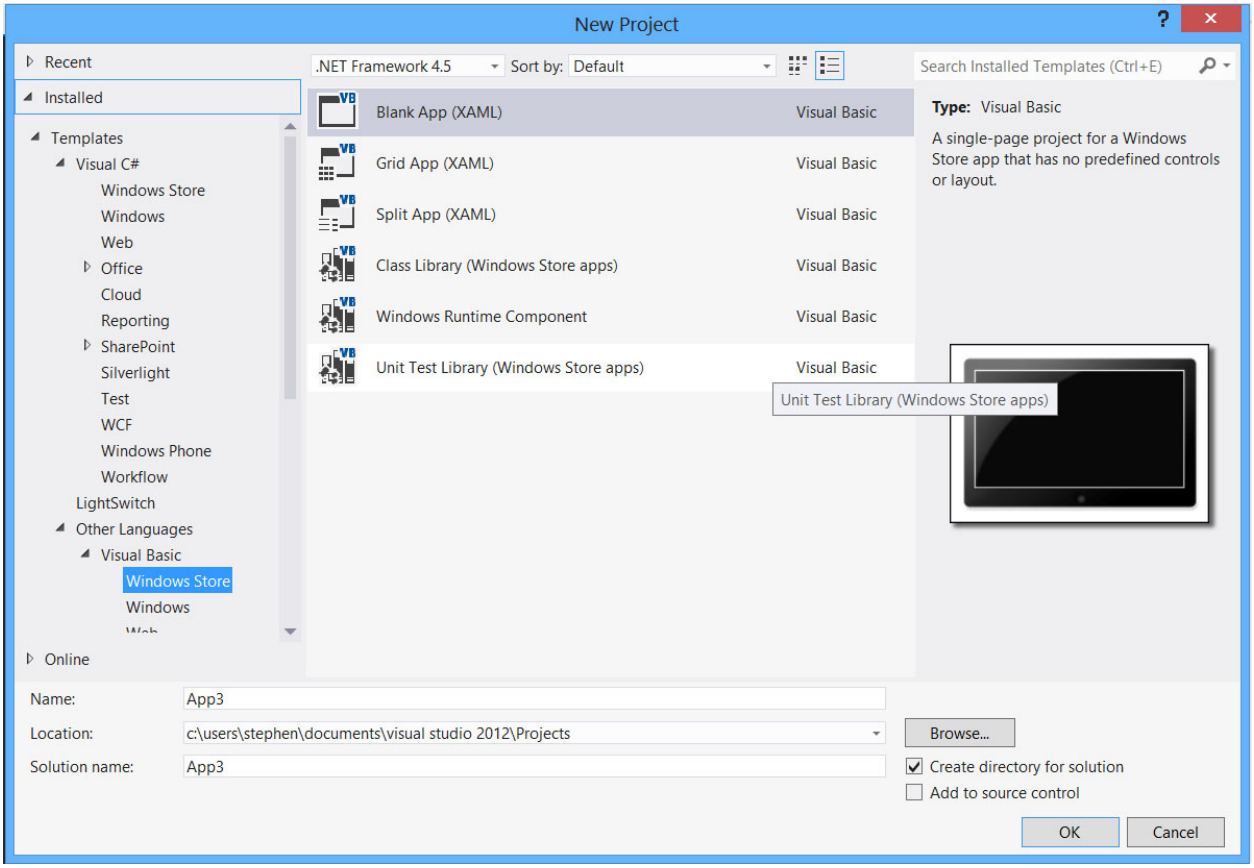

Figure 1 *– The New Project Dialog*

Blank App (XAML)

The Blank App (XAML) template creates a new Windows Store project consisting of a blank page with no functional elements. This is the place to start when creating a flat (single page) project. However, you can add other pages that may be navigated to, including more blank pages, grid-style pages, and split-style pages. I recommend starting here and then adding a Basic Page to the project. This method provides a bit more of a starting point for design—a grid element with the app title in place.

The Grid App (XAML) template allows you to create a three-level hierarchical project utilizing grouped data. It initially creates three pages. The first (top) page of the hierarchy presents a GridView of grouped items. Tapping or clicking one of the groups navigates to the second-level page presenting a ListView of items on the right with group info on the left. Choosing an item in the ListView navigates to the item details page. Grid Apps are ideal for catalogues. GridViews and ListViews are differentiated by how they scroll. GridViews scroll horizontally and present data in more of a block format. ListViews scroll vertically and present information much like a ListBox does on the desktop application side.

How else might a Grid App be used? Suppose you wanted to create an app for a "Who's Who in Baseball?" A Grouped Items Page would be the starting point and would contain a GridView of the thirty Major League Baseball teams. Tapping or clicking a team reference would take you to a Group Details Page displaying more details about the team on the left and a list of the players for that team on the right. When the user taps a player reference, the app navigates to an Item Detail Page with a biography, photos, and stats for that particular player.

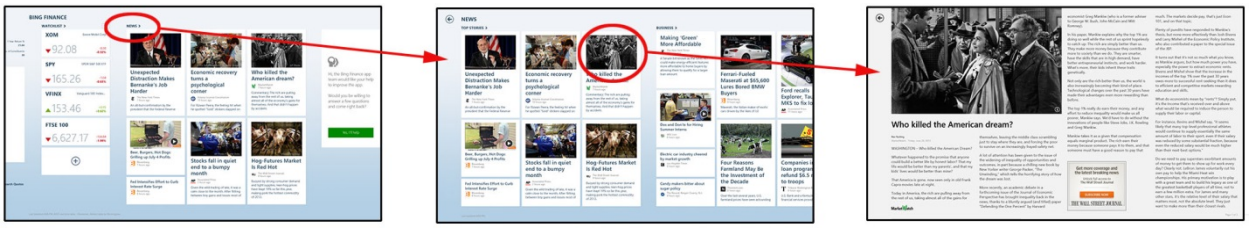

Figure 2 *– Pages can be added and removed providing flexibility. The Bing Finances app utilizes a Grid App approach. The main screen has groups, such as News. Clicking the News header navigates to another Grouped Items Pages with different groups of news such as "Top Stories" and "Business." Clicking one of the stories brings up a page presenting that particular news story with a back button to return to the previous page.*

Split App (XAML)

The Split App (XAML) template provides for a dual-level hierarchical structured app in which the user can burrow down from a GridView group collection page to a detailed page. Here, group items are presented in a ListView on the left and details about selected items on the right.

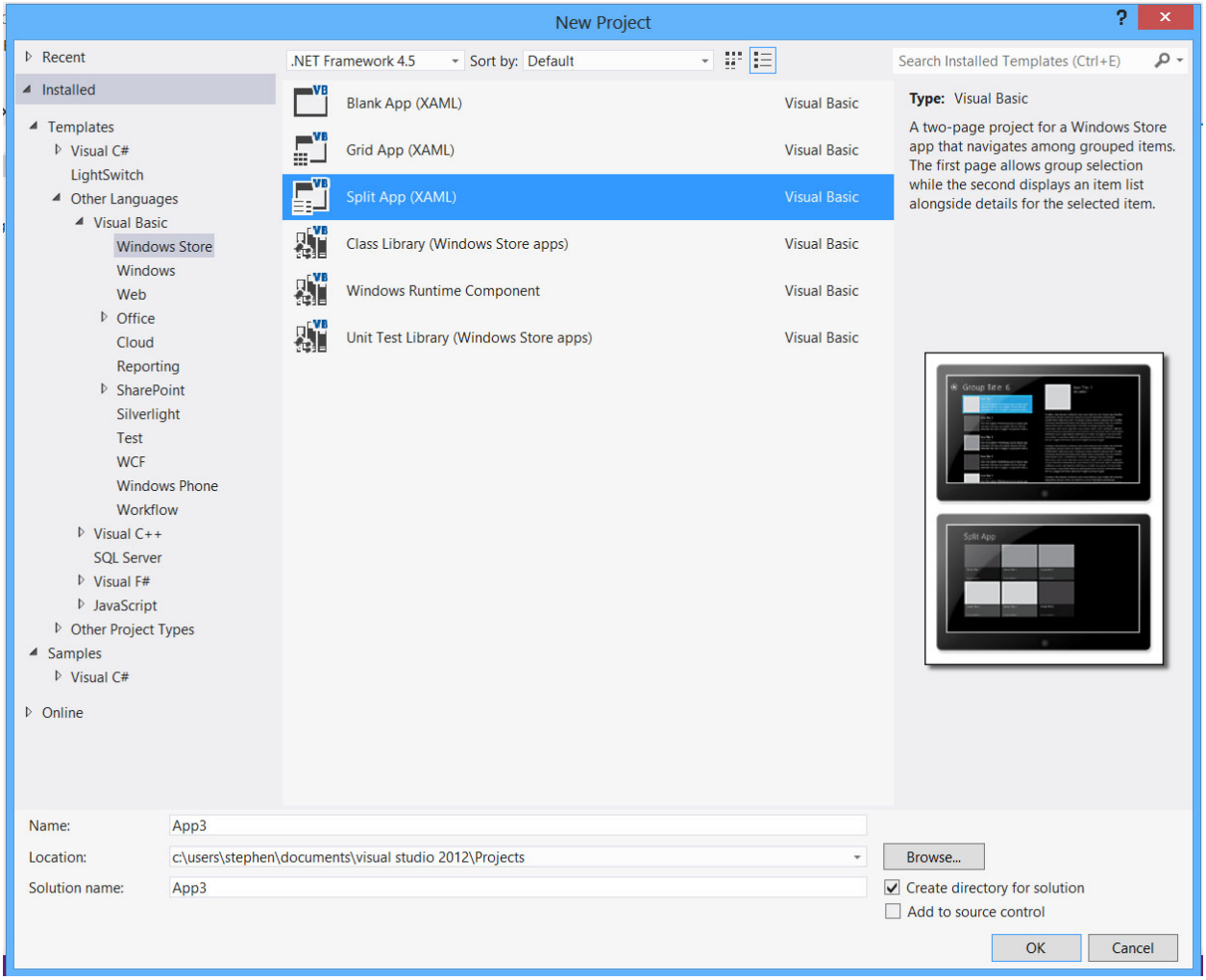

Figure 3 *– The Split App (XAML) template opens with two pages – one for a broad group view and another for a detailed list view to present more information for the selected item.*

A news reader might be built using a Split Page template. The Items Page would be a starting point for the app. It would list major groups of an RSS data feed. Clicking a group item in the Items Page would reveal a list of stories on the left with the full text of a selected story appearing on the right.

Choosing a Template

Step 2: Choose the Blank App (XAML) template. Provide a name for the project, such as "CIS165DB My First App" at the bottom of the dialog and select the location of where to save the project. The "Create Directory for Solution" checkbox should be checked. Click OK.

Examining the Source Code Files

In the Solution Explorer, the various source files of the project are listed by categories (or folders). The first category is Properties. It contains a file named AssemblyInfo.cs (or AssemblyInfo.vb). This contains key information about your project, such as title, description, copyright information, and version number. The Windows Store can use the

assembly version number to provide automated updating of your project as it is revised over time.

The next category is the References. Here, you will see two files initially. One is ".NET for Windows Store Apps" and the other is "Windows". These provide references to the built-in classes for .NET and Windows 8. For example, open up the .NET reference file. Scroll down to Windows.UI and expand it to see Colors. Here is a listing of all the predefined named colors. To use Chartreuse in your project, you can refer to it as Windows.UI.Colors.Chartreuse.

The Assets folder holds resources such as graphics, strings, colors, and data. Initially, you see four PNG graphic files for the required project logos. *Logo.png* is 150x150 pixels. *SmallLogo.png* is 30x30 pixels. The *SplashScreen.png* file is 620x300 and the *StoreLogo.png* is 50x50 pixels. You are encouraged to create your own logos and splash screen graphic of these sizes. You can either replace the default files with your version, maintaining the same names, or add your own files to the Assets folder and modify the referenced files in the *Package.appxmanifest* document discussed below. You can also add an optional Wide logo file (310x150 pixels) for use as a wide tile on the Start Screen, as well as a Badge logo (24x24 pixels) if creating a Badge app for use on the Lock Screen. These graphics might be created using image software such as Adobe Photoshop, but the default logos can also be edited directly in Visual Studio.

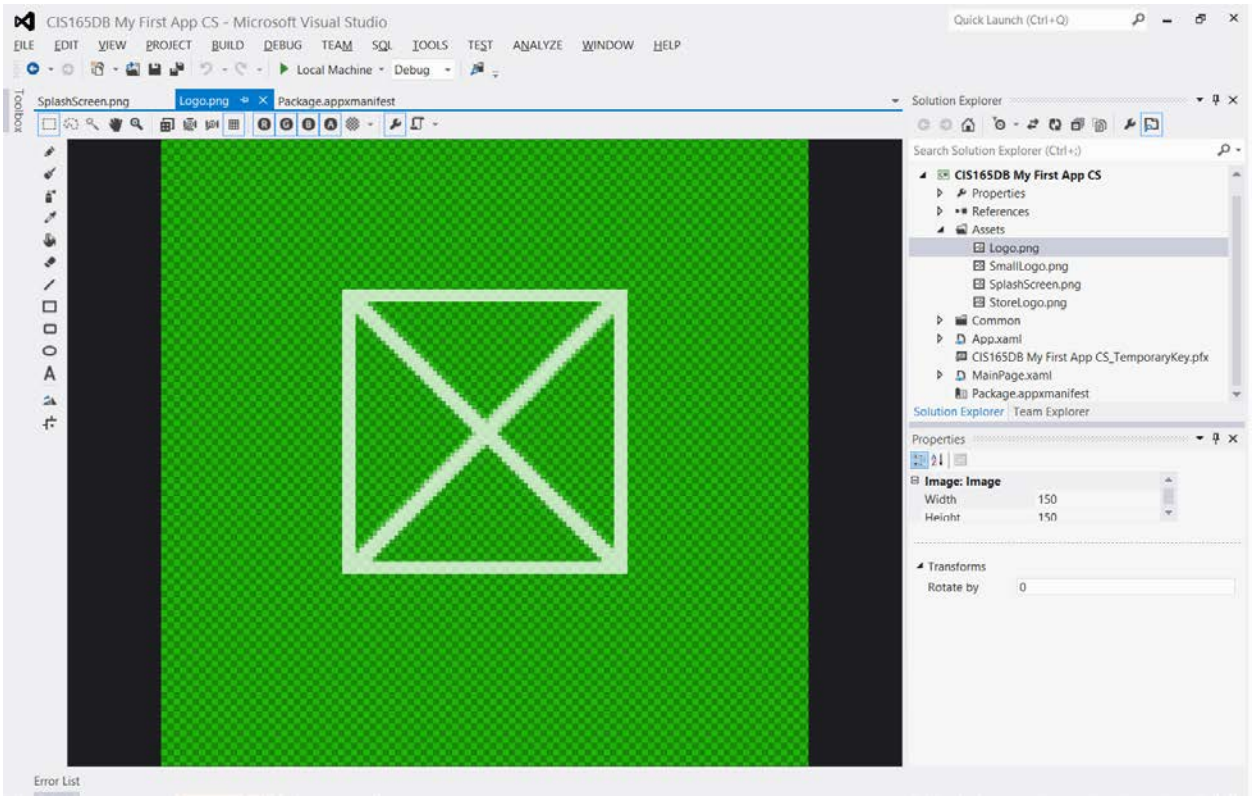

Figure 4 *– The default logo files can be edited directly in Visual Studio.*

The Common folder contains the StandardStyles.xaml file. This document provides predefined styles or RichTextBlocks, TextBlocks, Buttons, AppBars, AppBarButtons, and other objects. These styles may be modified and/or new styles may be added.

The App.xaml file contains information about the style documents used in the project. An underlying App.xaml.cs or App.xaml.vb file houses code used in launching or activating the app. Within the App.xaml.cs or App.xaml.vb file is a line of code that designates the starting page of the app. You will use MainPage.xaml as your starting page; there is a line of code that specifies this in thei App.xaml.cs file:

C# Code snippet

```
if (! root.Frame.Navigate(typeof(MainPage), args.Arguments))
```

The VB version of the document expresses it similarly:

VB Code snippet

```
If Not root.Frame.Navigate(GetType(MainPage), args.Arguments) Then
```

This is the statement you would change to utilize a different page as a starting point for the app.

Since you will choose the Blank Page (XAML) template to begin the project, a MainPage.xaml file was added that in essence is a Blank page. This is your starting page for the app. The XAML file contains XAML code defining the interface controls and their attributes and event handler functions. These functions are coded with C# or VB in the underlying *MainPage.xaml.cs* or *MainPage.xaml.vb* file.

The Package.appxmanifest contains property settings for the project, including the app's display name, description, and references to the various logo files to be used. It also contains a listing of what device capabilities and features should be available to the app including the Internet, camera, location (GPS), or microphone.

Creating the Interface

Step 3: Double click the MainPage.xaml reference in the Solution explorer. This will open the Designer and XAML code panels for the MainPage. Examine the XAML code. This code displays the pages visually in the Designer panel.

Highlight the value between the quotes "({StaticResourceApplicationPageBackgroundThemeBrush})" and replace it with "DarkRed". The page in the Designer panel will display the change. What can you learn from this? XAML controls (in this case, a Grid) are defined by attribute setting pairs, such as Background="DarkRed". Attributes and properties are synonymous.

Step 4: Next add some elements to the page. In the ToolBox, drag a rectangle control to the page. Note that a <Rectangle . . ./> tag is added between the <Grid> and </Grid> tags. It contains several attributes that define the rectangle's position, size, fill color, and more.

Attributes can be modified in three ways:

1. Changing a control's properties in the Properties panel
2. Changing attributes visually in the Designer panel (such as dragging the handles of a control to resize it, or dragging the control to reposition its location)
3. Modifying the XAML code for the page

Give the rectangle a name in the Properties panel of “rect1”. An attribute of x:Name=”rect1” will appear in the rectangle’s definition tag in the XAML code. View the Brush > Fill property. Change the fill to white (255,255,255,100%). The Fill attribute of the rectangle will change to “White”. (Visual Studio will automatically substitute named colors for the ARGB numerical values where a named color exists.)

Set the position of the rectangle by changing the XAML code. Set the Margin values to “140,140,0,0”. Change the Height and Width attributes to each have values of 400.

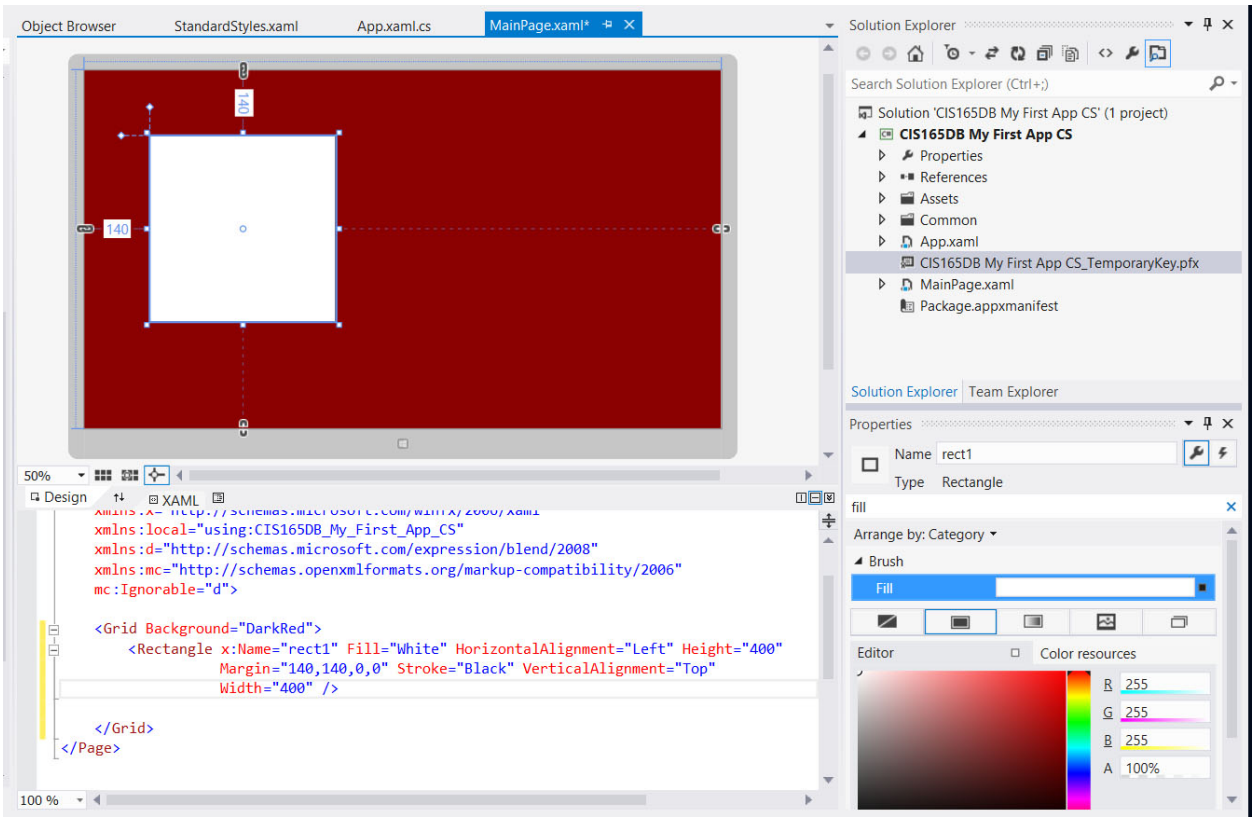

Figure 5 *– Details of the rect1 control tag.*

Step 5: Add a second rectangle by copying and pasting the full <Rectangle . . ./> tag. Modify the x:Name attribute of the copy to “rect2” and Margin attribute to “640,140,0,0”.

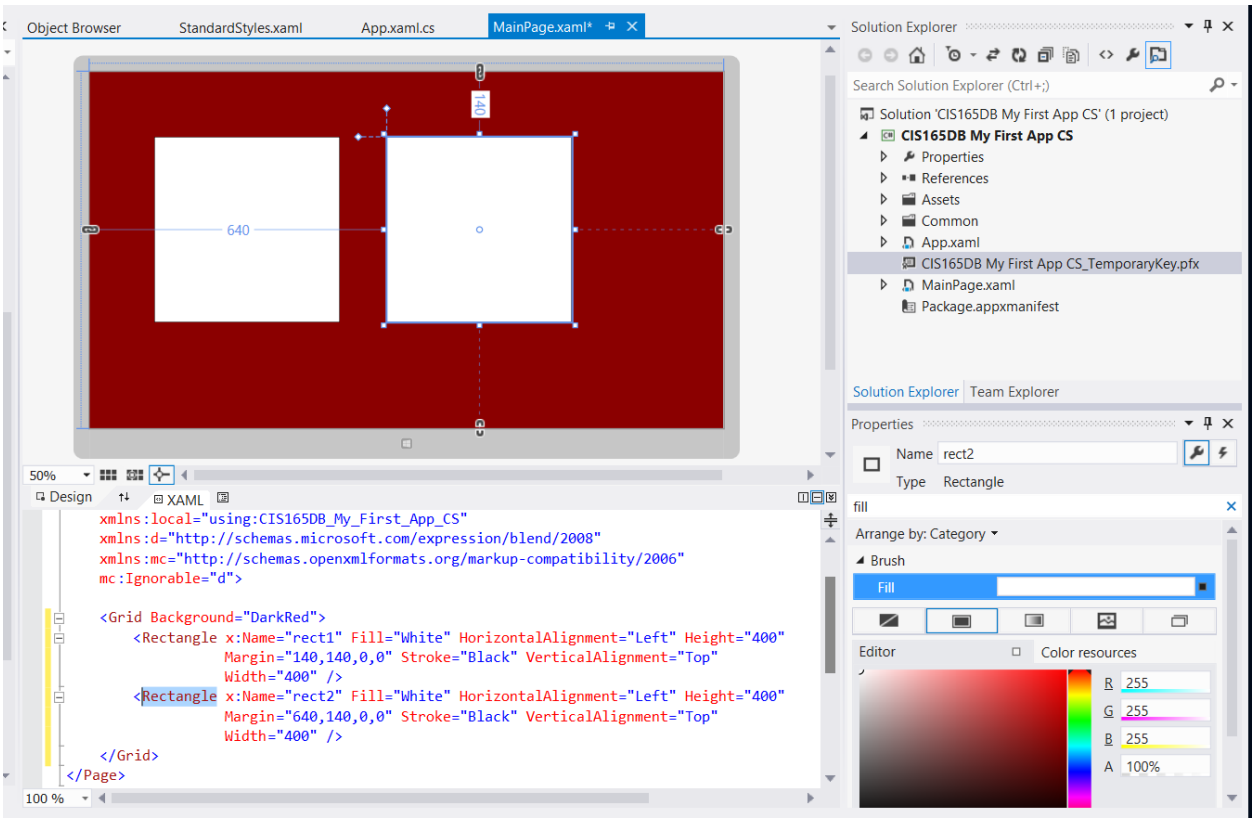

Figure 6 *– Details of the rect2 control tag.*

Step 6: Add a ToggleSwitch control below the rect1 rectangle. Set the x:Name to "toggleRect" and change the Header value to "Choose the rectangle". In the Properties panel set the OffContent property to "Left" and the OnContent property to "Right".

Step 7: Add a Button. Set the XAML as follows:

```
<Button HorizontalAlignment="Left" Margin="340,600,0,0"
VerticalAlignment="Top" Background="White" Height="60" Width="60" />
```

Step 8: Next, you will add an event handler to the button. In the properties panel, click the Event Handler button in the upper right of the panel (the one with the lightning bolt icon). In the Tapped value, enter "ChangeColor" as the function name. Pressing the Enter key will open the *MainPage.xaml.cs* or *MainPage.xaml.vb* file in a new tab of the IDE.

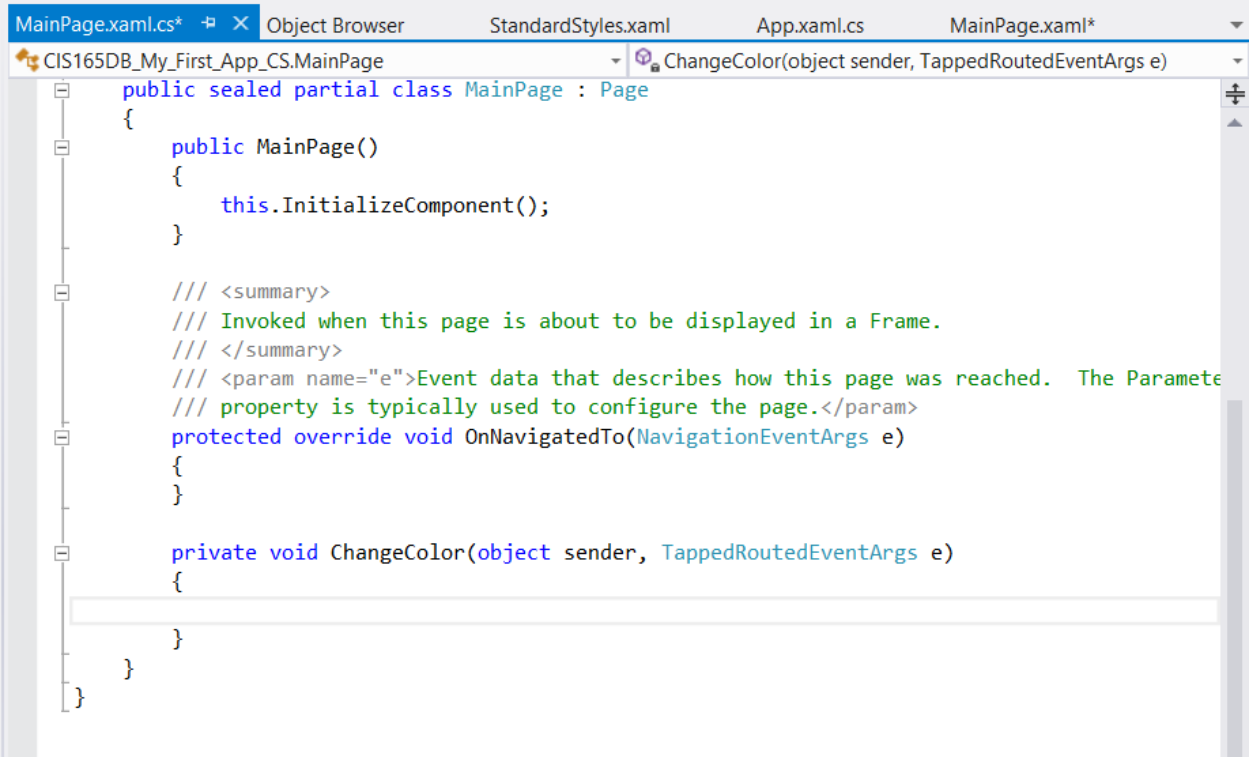

Figure 7 *– The underlying .xaml.cs file is where the event handler function is coded for an element on a .xaml page. For a VB project, the function will be in the.xaml.vb file.*

Step 9: You will modify the code later. Click the *MainPage.xaml* tab of the IDE to return to the Deisgner/XAML panels. Copy the Button tag and paste two copies of it in the XAML code. Alter the Margin attributes in the copies to "440,600,0,0" and "540,600,0,0" respectively. Change the Background attribute of the second button to "Yellow" and to "Cyan" for the third button. All three buttons will use the same function to handle the Tapped event. Note that you did not name these buttons – it is not necessary for the code to work. But you should, because you will need to do so later when modifying the layout for different view states.

Writing the Code

Step 10: If creating a C# Windows Store app, click the MainPage.xaml.cs tab at the top of the IDE and edit the ChangeColor function to be as follows:

C# Code:

```
private void ChangeColor(object sender, TappedRouteEventArgs e)
{
      Button xyz = (Button) sender;
      if (! toggleRect.IsOn)
      {
            rect1.Fill = xyz.Background;
      }
      else
      {
            rect2.Fill – xyz.Background;
      }
}
```

If creating a VB Windows Store app, click the MainPage.xaml.vb tab at the top of the IDE and edit the ChangeColor function to be as follows:

VB Code:

```
Private Sub ChangeColor(sender As Object, e AsTappedRouteEventArgs)
      If TypeOf sender Is Button Then
            Dim xyz as Button = CType(send, Button)
            If (Not toggleRect.IsOn) Then
                  rect1.Fill = xyz.Background
            Else
                  rect2.Fill – xyz.Background
            End If
      End If
End Sub
```

This event handler function will work to change the fill color of the rectangle (selected by the toggle switch control) to the Background color of the button that was tapped or clicked.

Test/Debug the App

Step 11: There is a drop-down list of testing environment options in the middle of the Visual Studio IDE button bar, as shown in the following figure. Click the drop down arrow on its right and choose Simulator.

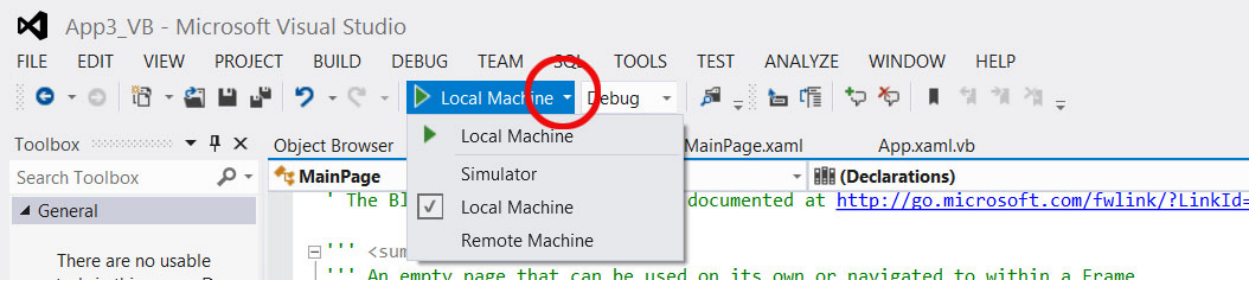

Figure 8 *– The app can be tested in the device Simulator or on the Local Machine, adding a tile to the Start Screen and filing the screen resolution of the development device. It can also be deployed to a Remote Machine, such as connected tablet.*

Click the Green arrow to the left of the designated testing environment. In this case the Simulator will be launched with your app displayed in it.

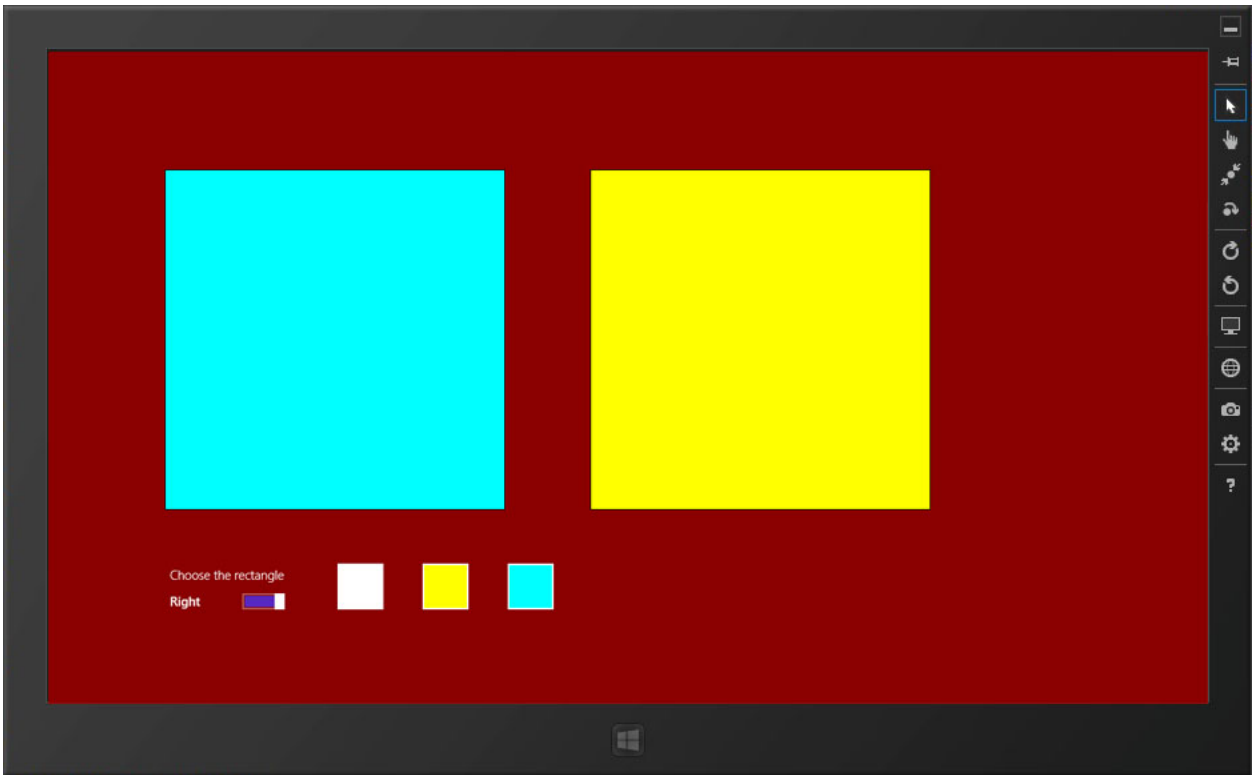

Figure 9 *– The app displayed in the simulator at 1366x768 resolution.*

The buttons on the right of the simulator can be used to simulate touch environment or mouse or to change the simulated resolution of the device. Go ahead and explore them. For more information on using the simulator, click the Help button located at the bottom of the column of buttons on the right side of the simulator window.

Step 12: Test the app, simulating both touch taps and mouse clicks. Verify that you can set each rectangle to all three colors. Simulate the app at different resolutions in the simulator. Note that in the rotated portrait view, the app is cut off on the right.

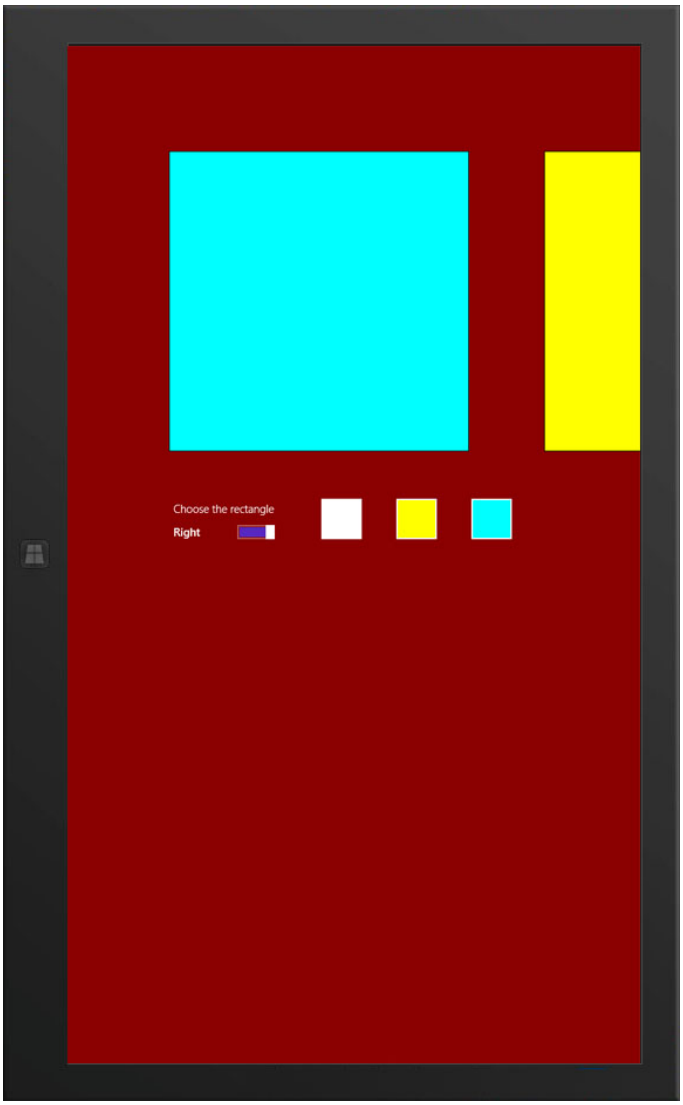

Figure 10 *– Some of the interface is cut off on the right when the app is displayed in the simulator at a rotated portrait view.*

Switch back to a resolution of 1366 x 768 pixels.

To deploy your app on the Windows Store, it must accommodate snapping. You can view the snapped view in the simulator. Point to the left edge and drag another open app in as a snapped app. Your app now looks like it did in the 1024x768 view with part of it cut off on the right.

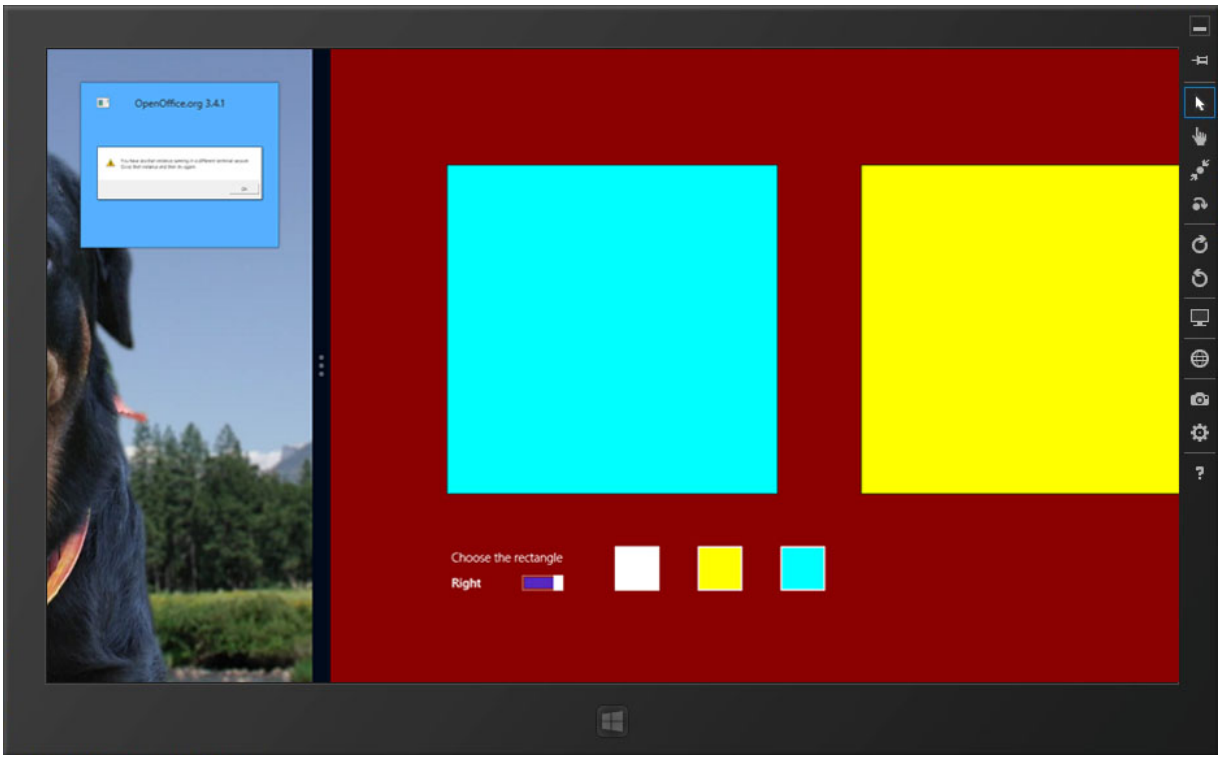

Figure 11 *– In the Filled view state, the right rectangle is partially cut off, but the app still functions okay.*

Drag the separator bar in the simulator so the app is now the snapped app.

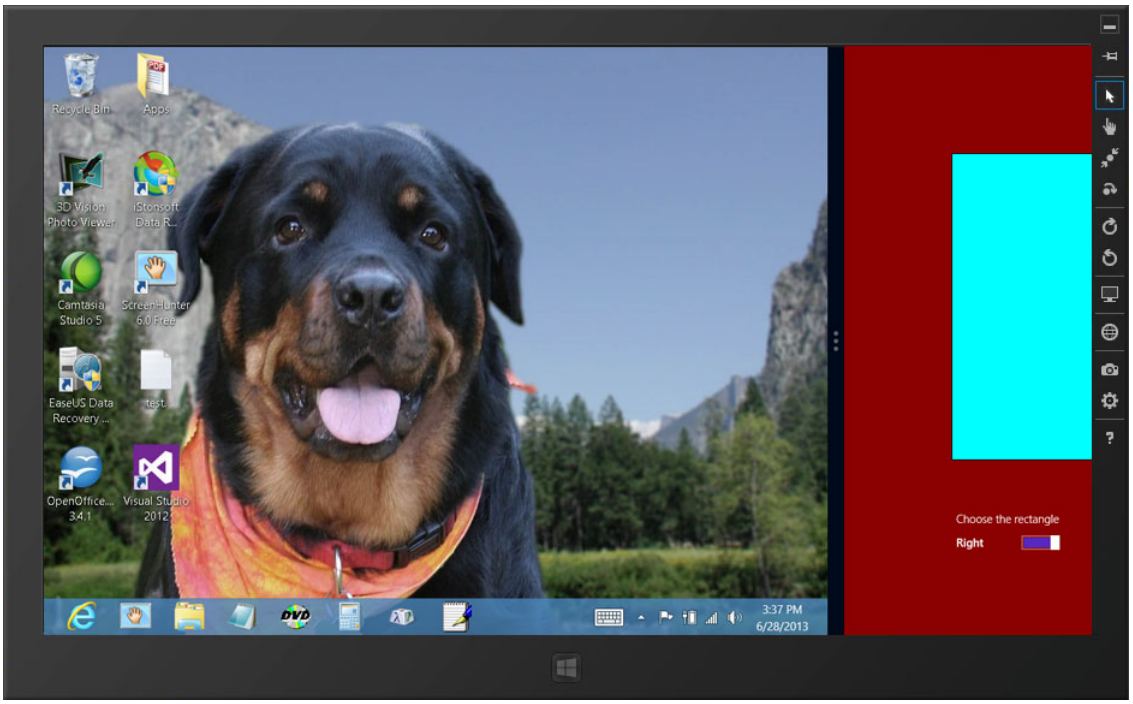

Figure 12 *– In the Snapped view state, most of the interface is hidden and the app is not useful.*

In this example, the usefulness of the app is greatly diminished in the snapped view state. Controls are lost off the nonvisible right. It is important to modify the app's layout for the snapped view state so it remains at least somewhat viable in that view.

Close the simulator and return to the IDE by pressing Ctrl + Alt + F4.

Modifying the App

Changing the Starting Page

Step 13: Open the *MainPage.xaml* tab. You can add XAML code here to handle the various view states by adding a <VisualStateManager.VisualStateGroups> block structure. But it is easier to add a Basic Page to your project that already has the <VisualStateManager.VisualStateGroups> structure in place. Beforehand, copy the XAML code for the rectangles, toggle, and buttons from the *MainPage.xaml* code.

From the Project menu, select "Add New Item…" Select the Basic Page item, set the name to "*BasicDemo.xaml*" at the bottom, and click 'OK'.

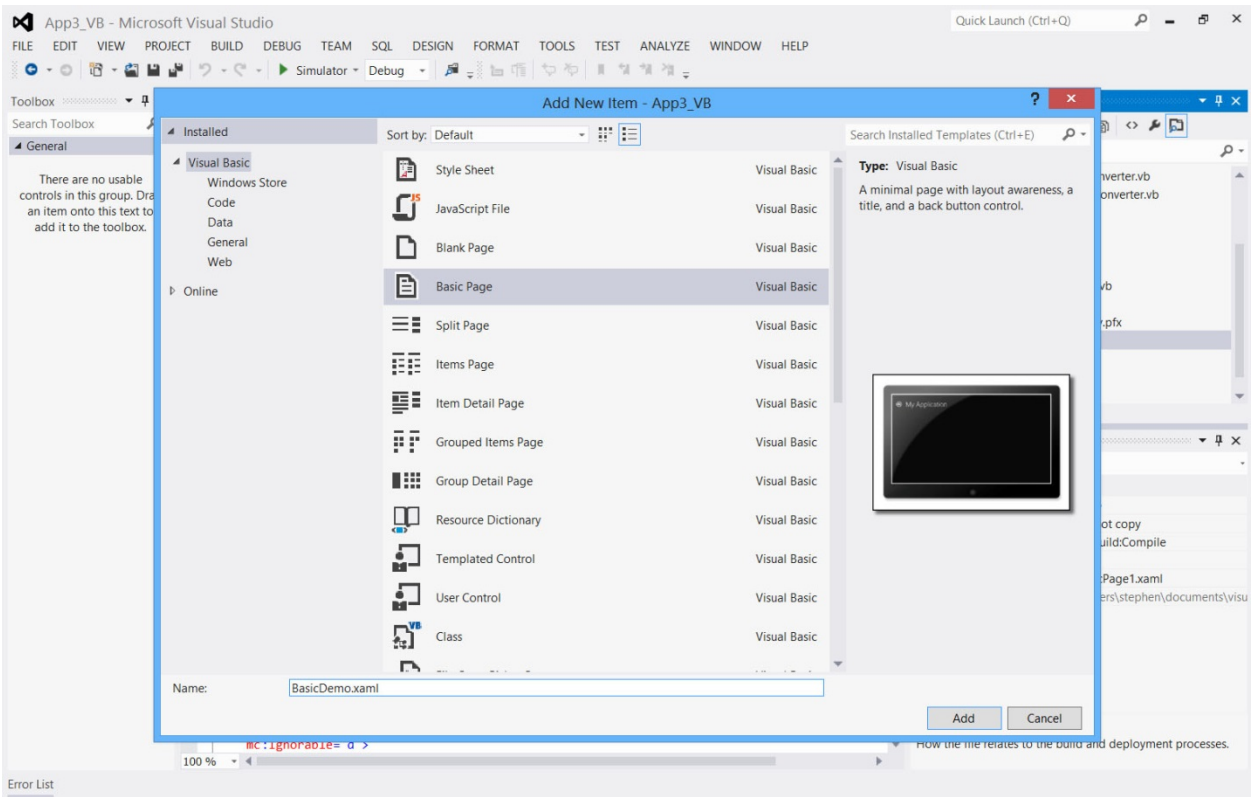

Figure 13– *The Project > Add New Item menu choice allows for any of the seven different page types to be added to the project.*

The XAML page along with its underlying C# or VB code will be added to the project. Several files are added to the Commons folder to accommodate the new template. Initially, you may see an "Invalid Markup" error in the Designer panel. Simply choose "Build Solution" from the Build menu or press the F6 key and the error should go away and will be replaced with the visual design of the BasicDemo page.

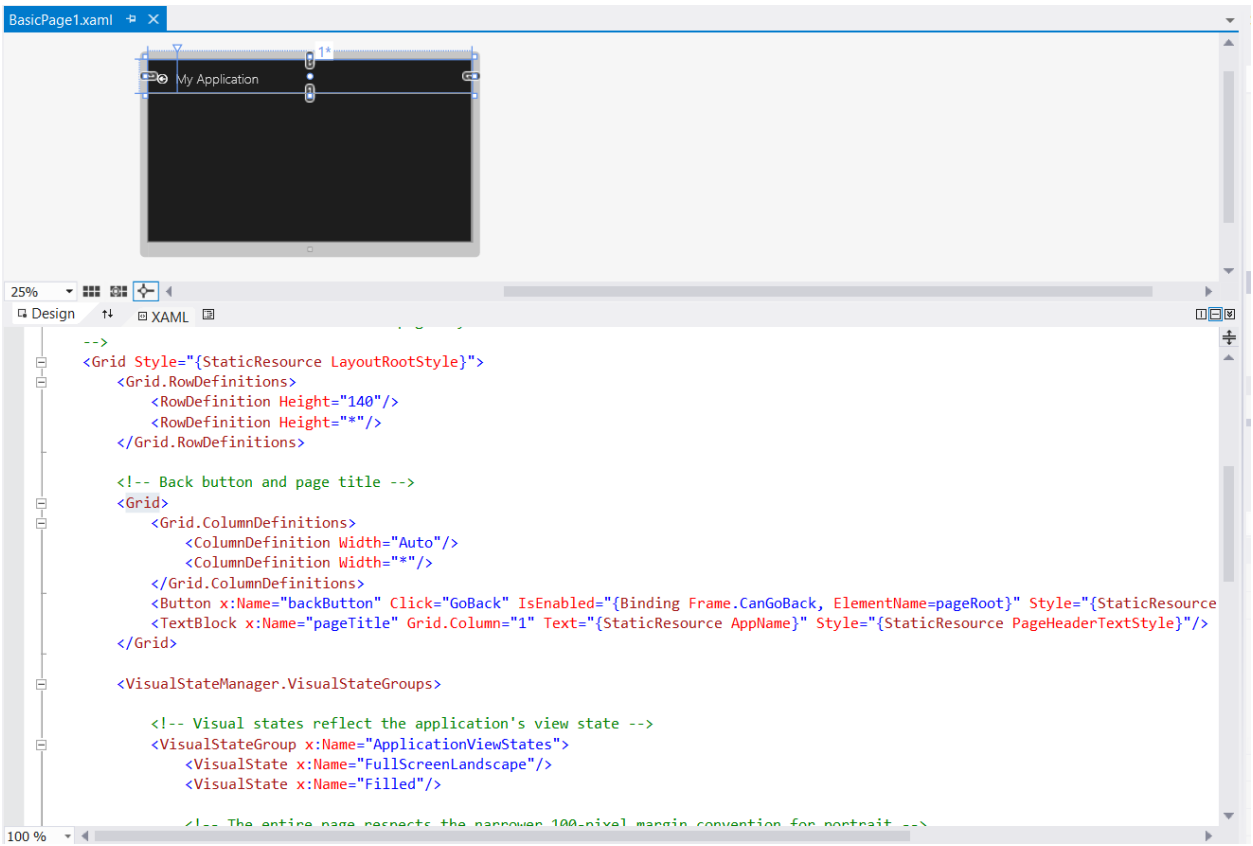

Figure 14– *The starting XAML code and display of the default Basic Page.*

Step 14: Explore the XAML code of the BasicDemo page. Several differences from the previous MainPage.xaml blank page are evident, including the following:

- A block of Page Resources
- A fuller Grid section with a specified style
- Two grid rows (the top at 140 pixels and the second having a wildcard value of "*", denoting occupation of all the remaining vertical space)
- A VisualStatesManager code block.

It is this VisualStatesManager that you will modify to specify different layouts for different view states. You will also note that a title and a back button is visible in the Designer panel. The display of the back button is conditional on the context and that is controlled in the underlying VB or C# code for the page.

Step 15: Before creating the different layouts, you will paste the XAML code for the elements copied from the MainPage to this document. Add a <Grid Grid.Row="1"> and </Grid> pair of tags before the <VisualStateManager…> block. Then paste the copied code for the elements between these two tags as shown below.

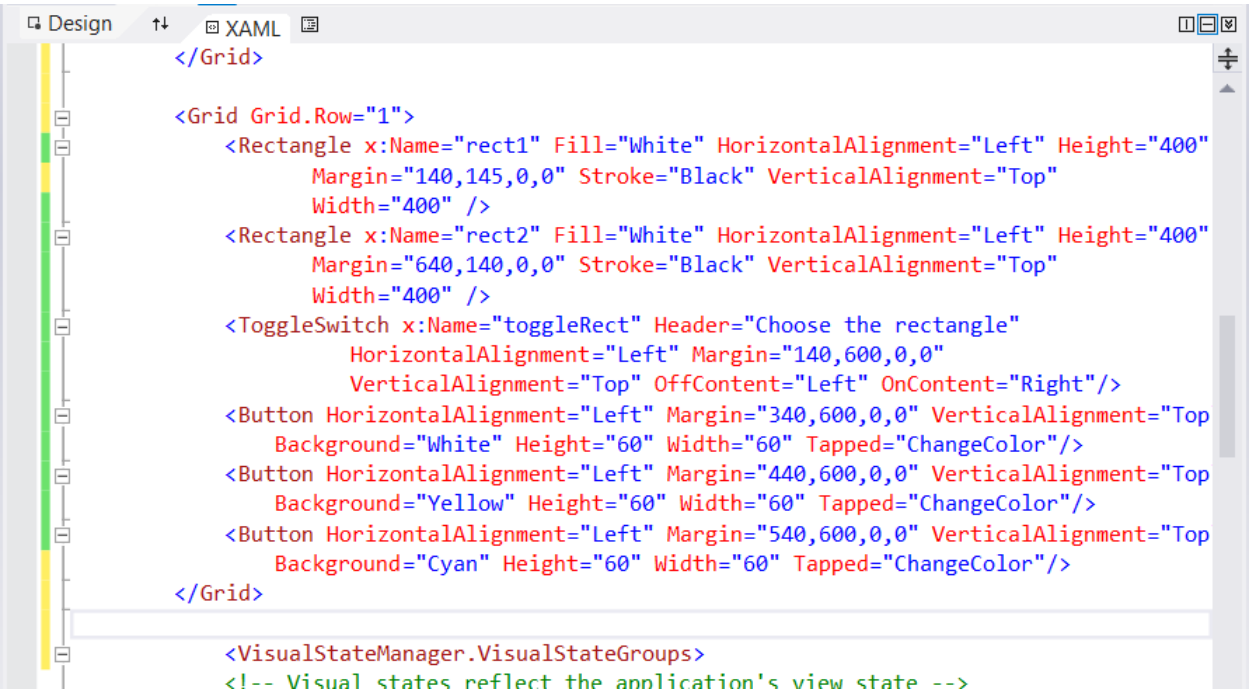

Figure 15– *The code is pasted between a pair of Grid tags that defines row 1 of a parent Grid.*

Provide names for the three buttons of btnWhite, btnYellow and btnCyan.

In the Designer panel, you can see that the pasted controls are too far down. This is because the Margin attributes are relative to being within the Grid. But this is a nested Grid and begins 140 pixels from the top edge. Alter the margin values in the XAML for all six controls as shown in the following table:

Margin Values for Six Controls

Control	New Margin value
rect1	140,10,0,0
rect2	640,10,0,0
toggleRect	140,450,0,0
btnWhite	340,450,0,0
btnYellow	440,450,0,0
btnCyan	540,450,0,0

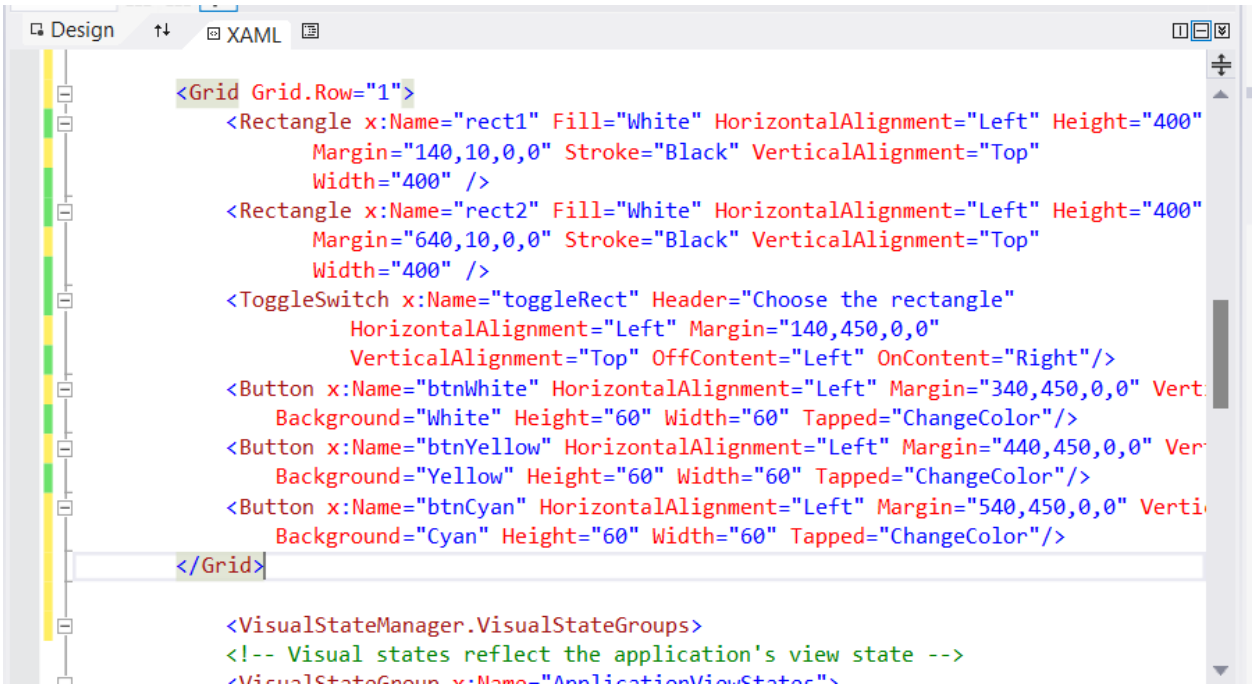

Figure 16 *– The BasicDemo.xaml code with the new Margin values and the Buttons named.*

Step 16: Before you can test the app, you need to change the *App.xaml.vb* code or the *App.xaml.cs* code to reflect the use of the BasicDemo page as the startup page for the app. In App.xaml.cs, change the "mainPage" reference to "BasicDemo" (about line 67):

C# Code snippet:

```
        if (! root.Frame.Navigate(typeof(BasicDemo), args.Arguments))
```

```
62              if (rootFrame.Content == null)
63              {
64                  // When the navigation stack isn't restored navigate to the first page,
65                  // configuring the new page by passing required information as a navigation
66                  // parameter
67                  if (!rootFrame.Navigate(typeof(BasicDemo), args.Arguments))
68                  {
69                      throw new Exception("Failed to create initial page");
70                  }
71              }
```

Figure 17*–The modified App.xaml.cs code to reflect the BasicDemo page as the starting page*

If using Visual Basic, change the reference of "MainPage" in App.xaml.vb to "BasicDemo" (about line 34):

VB Code Snippet:

```
        If Not root.Frame.Navigate(GetType(BasicDemo), args.Arguments) Then
```

```
        End If
        If rootFrame.Content Is Nothing Then
            ' When the navigation stack isn't restored navigate to the first page,
            ' configuring the new page by passing required information as a navigation
            ' parameter
            If Not rootFrame.Navigate(GetType(BasicDemo), args.Arguments) Then
                Throw New Exception("Failed to create initial page")
            End If
        End If
```

Figure 18 *–The modified App.xaml.vb code to reflect the BasicDemo page as the starting page.*

Step 17: Test the app to make sure the BasicDemo page starts up and all buttons work as they did previously. Then return to the *BasicDemo.xaml* document.

Handling Visual States

Step 18: Return to the BasicDemo.xaml tab. Examine the <VisualStateManager.VisualStateGroups> code block. There are four <VisualState> objects named:

- FullScreenLandscape
- Filled
- FullScreenPortrait
- Snapped

Some code already exists to modify the layout for the latter two view states. First, you will work with the FullScreenPortrait block of code. Object animations include modify attributes of controls. The animations are grouped together in Storyboard tags. Currently there is an ObjectAnimationUsingKeyFrames tag in which the BackButton's Style property is altered to the PortraitBackButtonStyle (defined in the *StandardStyles.xaml* document of your project). The KeyTime is set to 0, which means it happens immediately upon entering this visual state. Since the app only has one page, you never see the BackButton and thus the effect of different view states is lost. The format of this animation, however, will provide the model for resizing and relocating the controls.

First, you will move the rect1 Rectangle control closer to the left and resize it so it is only 300 x 300 pixels. Copy the current <ObjectAnimatonUsingKeyFrames> block (3 lines). Paste it three times and change the TargetName in each to "rect1" and the TargetProperty to "Margin", "Height" and "Width" respectively. Finally, change the Value properties to "40,10,0,0", "300", and "300" respectively.

XAML Code snippet:

```
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
Storyboard.TargetProperty="Margin">
         <DiscreteObjectKeyFrame KeyTime="0" Value="40,10,0,0"/>
</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
Storyboard.TargetProperty="Height">
         <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
Storyboard.TargetProperty="Width">
         <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
</ObjectAnimationUsingKeyFrames>
```

```
        <VisualStateGroup x:Name="ApplicationViewStates">
            <VisualState x:Name="FullScreenLandscape"/>
            <VisualState x:Name="Filled"/>

            <!-- The entire page respects the narrower 100-pixel margin convention for portrait -->
            <VisualState x:Name="FullScreenPortrait">
                <Storyboard>
                    <ObjectAnimationUsingKeyFrames Storyboard.TargetName="backButton" Storyboard.TargetProperty="Style">
                        <DiscreteObjectKeyFrame KeyTime="0" Value="{StaticResource PortraitBackButtonStyle}"/>
                    </ObjectAnimationUsingKeyFrames>
                    <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1" Storyboard.TargetProperty="Margin">
                        <DiscreteObjectKeyFrame KeyTime="0" Value="40,10,0,0"/>
                    </ObjectAnimationUsingKeyFrames>
                    <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1" Storyboard.TargetProperty="Height">
                        <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
                    </ObjectAnimationUsingKeyFrames>
                    <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1" Storyboard.TargetProperty="Width">
                        <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
                    </ObjectAnimationUsingKeyFrames>

                </Storyboard>
            </VisualState>
```

Figure 19 *– Add the highlighted code to the FullScreenPortrait Visual State block to move the rect1 Rectangle control and resize it.*

Step 19: Now test the app in the simulator. Rotate the simulator so it is in portrait mode—the rect1 rectangle should move to the left and grow smaller. Rotate back to landscape mode. The controls revert to the standard attributes. Close the simulator (Ctrl + Alt + F4).

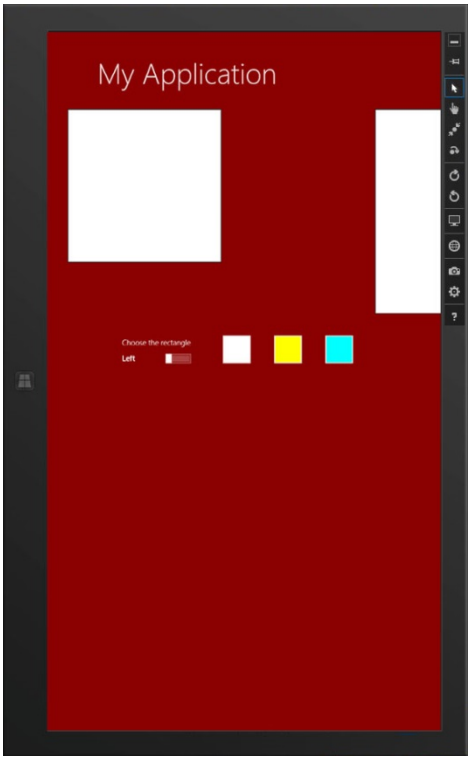

Figure 20 *– In the portrait view state, the rect1 Rectangle is now moved to the left and shrunk to 300x300 pixels.*

Step 20: Next, you will add code to move the rect2 Rectangle to 380 pixels from the left and resize it to 300,300:

```
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
Storyboard.TargetProperty="Margin">
       <DiscreteObjectKeyFrame KeyTime="0" Value="380,10,0,0"/>
</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
Storyboard.TargetProperty="Height">
        <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
Storyboard.TargetProperty="Width">
        <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
</ObjectAnimationUsingKeyFrames>
```

Step 21: Test again and view the app in both landscape and portrait modes.

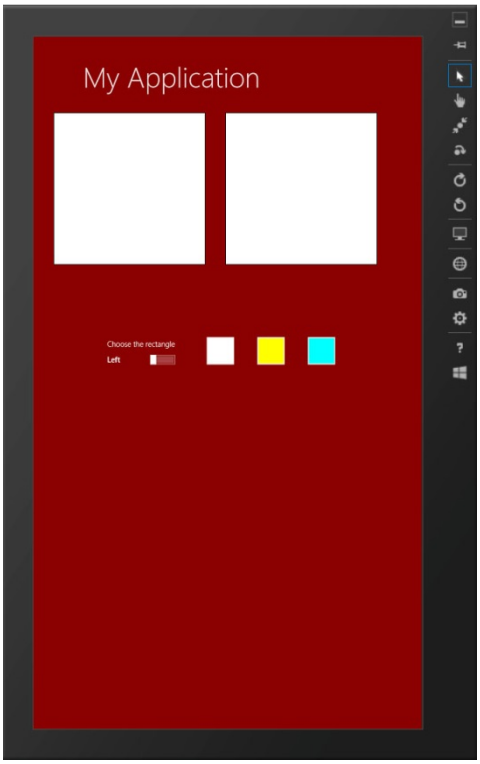

Figure 21*– The portrait view in the simulator now shows both rectangles resized to fit the view.*

Step 22: The other controls are fine for the Portrait view. Now, you will turn your attention to the FilledState view. Scroll up and break the <VisualState x:Name="Filled"/> tag so it is comprised of opening and closing tags.

Add opening and closing Storyboard tags between the opening and closing VisualState tags:

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

</VisualState>
```

```
<VisualState x:Name="Filled">
      <Storyboard>

      </Storyboard>
</VisualState>
```

Copy and paste the animation code you wrote for the rect1 and rect2 rectangle controls between the Storyboard tags. Since snapping and filled states only appear on devices that are 1366x768 or larger, this animation should work well for this state.

```
<VisualState x:Name="Filled">
      <Storyboard>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
             Storyboard.TargetProperty="Margin">
                   <DiscreteObjectKeyFrame KeyTime="0" Value="40,10,0,0"/>
            </ObjectAnimationUsingKeyFrames>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
             Storyboard.TargetProperty="Height">
                    <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
            </ObjectAnimationUsingKeyFrames>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
             Storyboard.TargetProperty="Width">
                    <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
            </ObjectAnimationUsingKeyFrames>

            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
             Storyboard.TargetProperty="Margin">
                    <DiscreteObjectKeyFrame KeyTime="0" Value="380,10,0,0"/>
            </ObjectAnimationUsingKeyFrames>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
             Storyboard.TargetProperty="Height">
                    <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
            </ObjectAnimationUsingKeyFrames>
            <ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
             Storyboard.TargetProperty="Width">
                    <DiscreteObjectKeyFrame KeyTime="0" Value="300"/>
            </ObjectAnimationUsingKeyFrames>
      </Storyboard>
</VisualState>
```

Step 23: Test your app in the simulator and snap another app to view your app in the filled state.

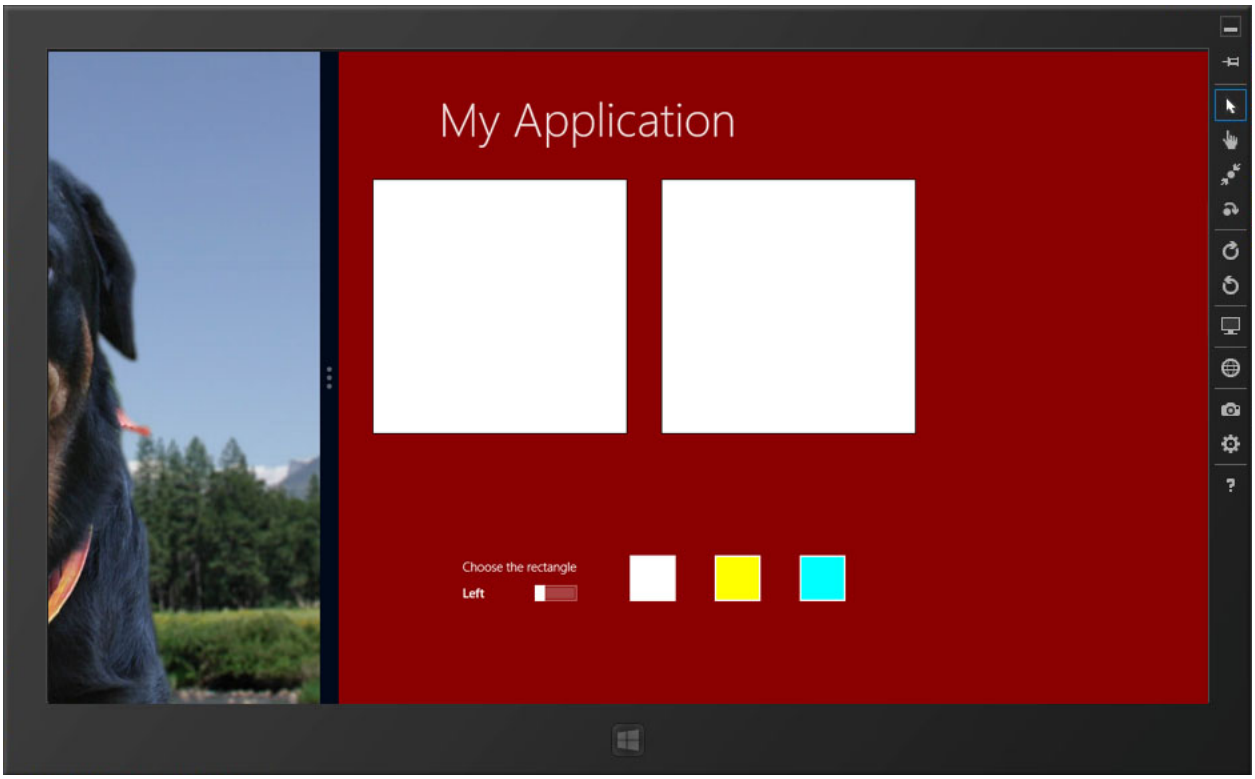

Figure 22– *The app in the Filled view state.*

Step 24: Finally, you will work on the snapped view. You will have the two rectangles move and shrink to be arranged top and bottom instead of side by side. Move the toggle, but also change the OffContent and OnContent properties to "Top" and "Bottom". You will also move the three buttons to be located below the toggle. Add the following code between the Storyboard tags of the Snapped VisualState.

```
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
Storyboard.TargetProperty="Margin">
        <DiscreteObjectKeyFrame KeyTime="0" Value="40,10,0,0"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
Storyboard.TargetProperty="Height">
        <DiscreteObjectKeyFrame KeyTime="0" Value="220"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect1"
Storyboard.TargetProperty="Width">
        <DiscreteObjectKeyFrame KeyTime="0" Value="220"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
Storyboard.TargetProperty="Margin">
        <DiscreteObjectKeyFrame KeyTime="0" Value="40,250,0,0"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
Storyboard.TargetProperty="Height">
        <DiscreteObjectKeyFrame KeyTime="0" Value="220"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="rect2"
Storyboard.TargetProperty="Width">
```

```
        <DiscreteObjectKeyFrame KeyTime="0" Value="220"/>
</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="toggleRect"
Storyboard.TargetProperty="Margin">
        <DiscreteObjectKeyFrame KeyTime="0" Value="40,475,0,0"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="toggleRect"
Storyboard.TargetProperty="OffContent">
        <DiscreteObjectKeyFrame KeyTime="0" Value="Top"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="toggleRect"
Storyboard.TargetProperty="OnContent">
        <DiscreteObjectKeyFrame KeyTime="0" Value="Bottom"/>
</ObjectAnimationUsingKeyFrames>

<ObjectAnimationUsingKeyFrames Storyboard.TargetName="btnWhite"
Storyboard.TargetProperty="Margin">
        <DiscreteObjectKeyFrame KeyTime="0" Value="40,550,0,0"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="btnYellow"
Storyboard.TargetProperty="Margin">
        <DiscreteObjectKeyFrame KeyTime="0" Value="140,550,0,0"/>
</ObjectAnimationUsingKeyFrames>
<ObjectAnimationUsingKeyFrames Storyboard.TargetName="btnCyan"
Storyboard.TargetProperty="Margin">
        <DiscreteObjectKeyFrame KeyTime="0" Value="240,550,0,0"/>
</ObjectAnimationUsingKeyFrames>
```

Step 25: Test the app in the simulator. Snap another app and then move the separator bar view to see the app in snapped view.

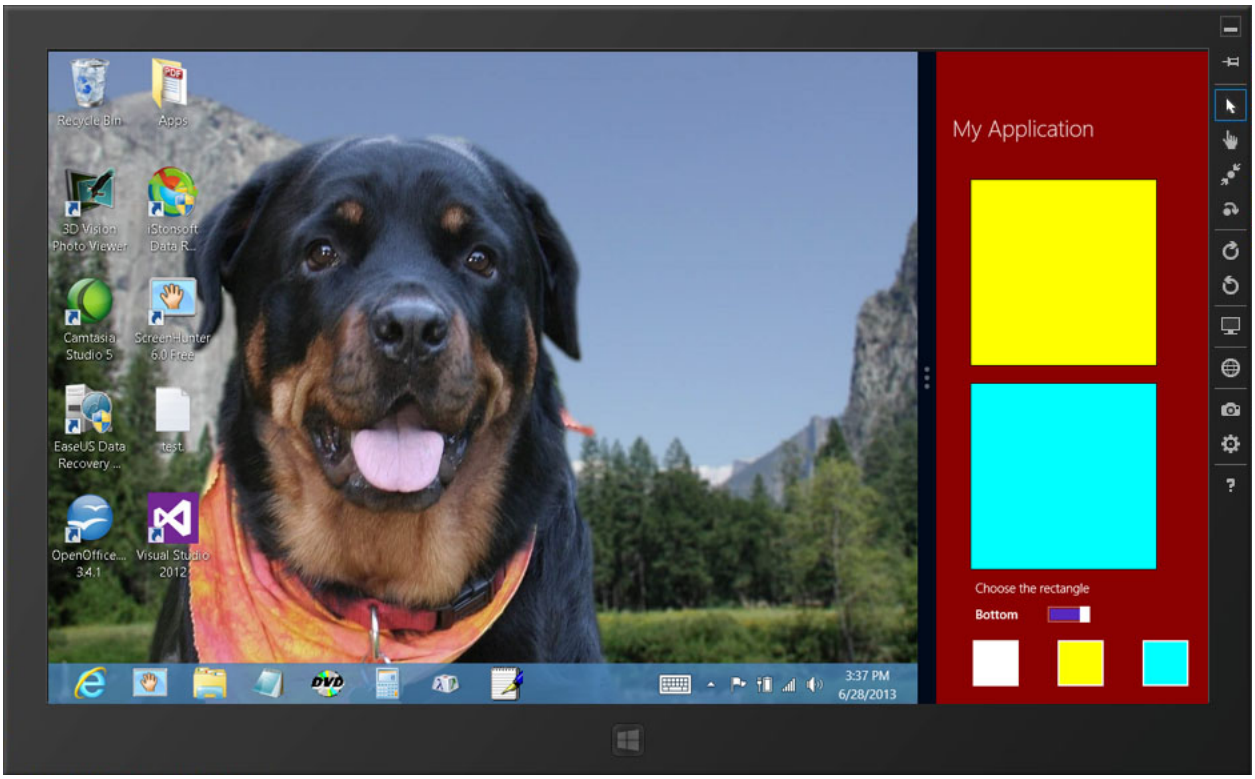

Figure 23 *– The app displayed in the Snapped view on the right with the Rectangles now stacked (rather than side by side) and the color Buttons below the ToggleSwitch.*

Close the simulator to return to the IDE. Save your project. If you test your app using the Local Machine rather than the simulator, a tile will be added on the far right of the Start Screen. You might want to do this and then test the various views on your computer.