UWP-017 - XAML Layout with RelativePanel

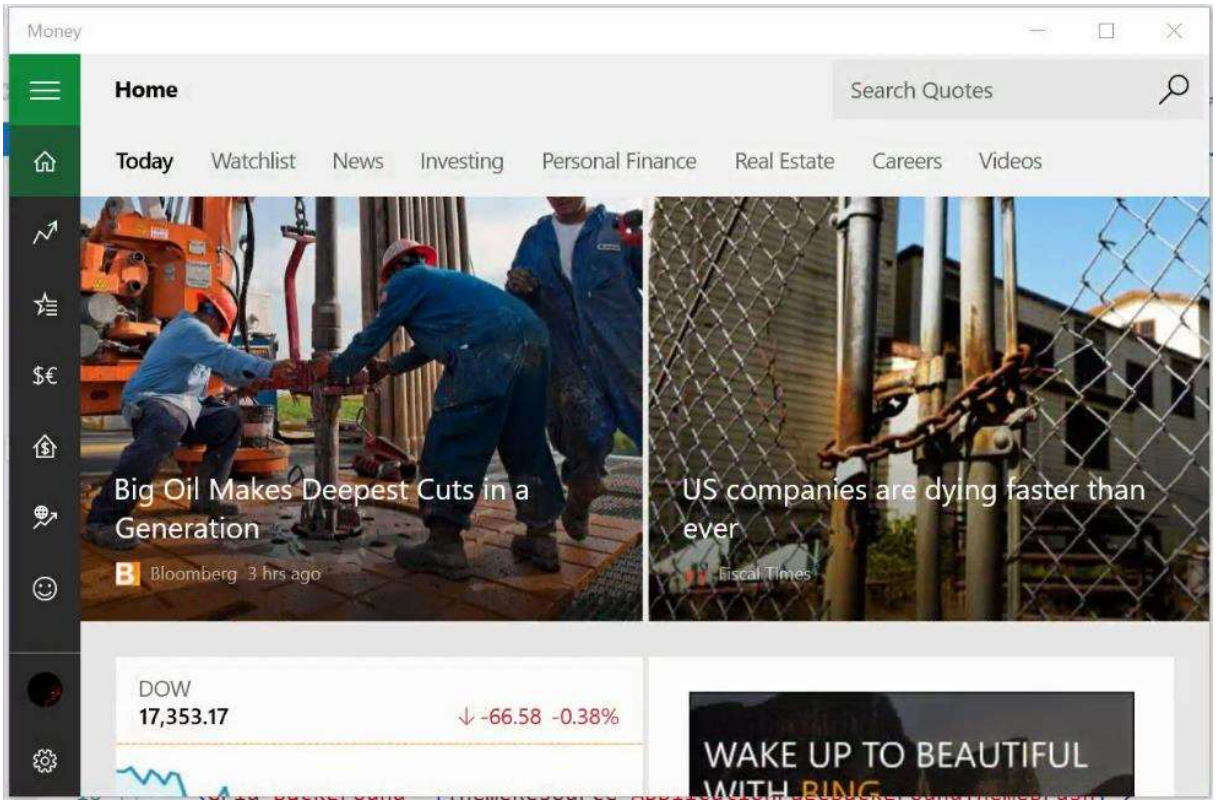
We're not done with Layout quite yet. Next we will talk about new Layout controls that were introduced with the Universal Windows Platform API.

You might wonder why are there new Layout controls? What has changed from previous APIs that we used to build apps for the Windows 8.1 store and the Windows Phone 8.1 store. There are two reasons why we need new Layout controls.

First of all, to help us build apps that look like they belong in Windows 10. I'll talk about that more in just a moment.

Secondly, to help us build applications that adapt to different device family and screen sizes and talk about that in an upcoming lesson.

If you're not familiar with the new Windows 10 aesthetic see the screenshot of the Money app, one of the stock applications that come preinstalled with Windows 10.



If you were to open the other stock apps, namely News, Weather, Sports, and a few others, you would see that they all share some common characteristics in their aesthetic and functionality that really identify themselves as Windows 10 applications. I want to take note of those features then, over the

course of the next few lessons, show how we can duplicate this ourselves with the built-in controls available in the Universal Windows Platform.

The aesthetic includes both the chrome and the “cards” in the main area. By “chrome”, I'm referring to the top and left-most area that provides navigation and other services to your application. The style of navigation which includes the icon with three horizontal lines is known as "hamburger navigation". When you click that button in the upper left-hand corner it will show a SplitView control that will display navigation to the various areas or functionality of your application.

In expanded mode both an icon and a title for the given area of the application is displayed at the top. We can navigate around using the expanded view or the compacted view. When we're in compacted mode, you can only see the icons on the left. The selected item is highlighted in one of the primary accent colors for the application.

Next, if you look at the very top of the application you'll see a bar containing a search box. Each app every app like a search bar in the upper right hand corner. In this case, we're searching for the current stock price for a given company. Also then to the left of that, docked over to the left-hand side is the title of the area that we're currently in and then to the left of that a navigation button that allows us to go back through the navigation history to get back to the homepage.

There are some other features of these applications like the card-based layout, where you have all of these little panels of cards that will dynamically resize themselves and then depending on the view port (the size of the window) that has been resized by the user, they will either shift down, wrap down to the next line or they will expand and contract.

In this lesson I’ll introduce the RelativePanel which is a Layout container that allows you to create user interfaces that don't have a clear linear pattern. When I say “linear pattern” what I mean is that it's for layouts that are not fundamentally stacked or wrapped like that card layout or even tabular like in a grid. So these are layouts that you may not find as easy to reproduce using a StackPanel or a Grid.

Now certainly, what we saw in the Windows 10 application we probably could achieve that using a StackPanel and Grid combination, however, I think you're going to find that these two new controls will help you achieve this a little bit more elegantly.

The RelativePanel defines an area where you position and align child objects, so you position other controls either in relation to each other or in relation to the parent panel itself. And there are three basic categories of attached properties that allow you to position the controls inside of the panel.

- There are panel alignment relationships. These have attached properties like align the top of my control with the panel. So AlignTopWithPanel, AlignLeftWithPanel, and so on.
- Then there are sibling alignment relationships. So AlignTopWith = (and then you give the name of the control that you want to be aligned top with).
- And then there are sibling positional relationships, so I want to be above my sibling, I want to be below my sibling, I want to be to the right and the left of my sibling.

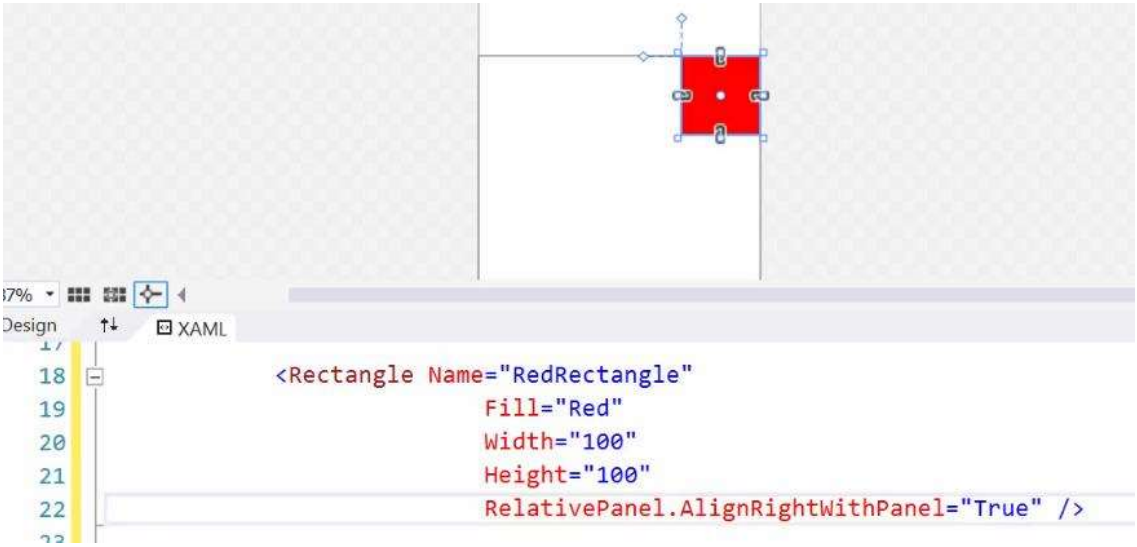
I’ve created a project named RelativePanelExample which I’ll expand the example to learn about RelativePanel.

So first I’ll add some RowDefinitions and add a RelativePanel in the second RowGrid.Row="1".

```
    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="1*" />
            <RowDefinition Height="2*" />
            <RowDefinition Height="1*" />
        </Grid.RowDefinitions>
        <RelativePanel MinHeight="300" Grid.Row="1" >
```

I also set a minimum height, so no matter what I never want the height of this RelativePanel to be less than 300 pixels.

Next I'll add a series of rectangles.



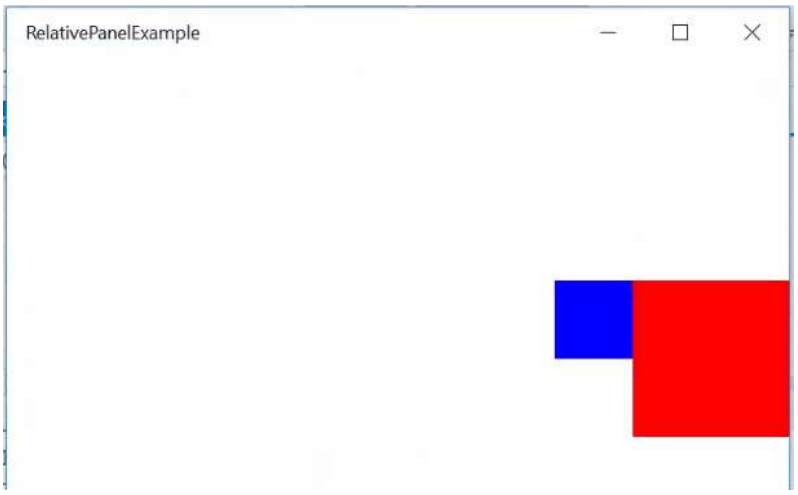

```
            <Rectangle Name="RedRectangle"
                       Fill="Red"
                       Width="100"
                       Height="100"
                       RelativePanel.AlignRightWithPanel="True" />
```

The first rectangle will have a fill set to Red. Notice I've set the RelativePanel.AlignedRightWithPanel ="True". This aligns the right side of the Rectangle with the panel.

Next I'll add another Rectangle that will be positioned in relation to that sibling Rectangle.

```
            <Rectangle Name="BlueRectangle"
                       Fill="Blue"
                       Width="50"
                       Height="50"
                       RelativePanel.LeftOf="RedRectangle" />
```

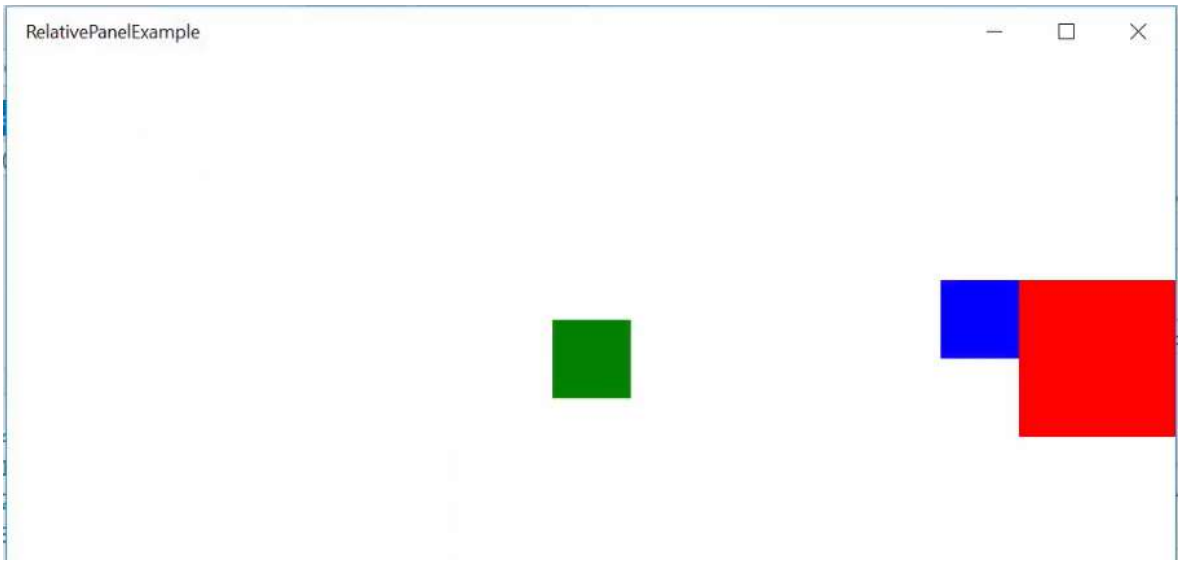
When running the application, even as we resize the application, the blue rectangle will always be to the left of the red rectangle. And the red rectangle will always be aligned to the right-hand side.



Next I'll demonstrate that you can also set multiple attached properties an XAML control.

```
30          <Rectangle Name="GreenRectangle"
31                     Fill="Green"
32                     Width="50"
33                     Height="50"
34                     RelativePanel.AlignVerticalCenterWith="RedRectangle"
35                     RelativePanel.AlignHorizontalCenterWithPanel="True" />
```

On the green rectangle I've set the AlignVerticalCenterWith="RedRectangle". So now the center of the red rectangle and the center of the green rectangle will be the same. Furthermore, I set the AlignHorizontalCenterWithPanel="True". So show me where the center is of the horizontal size of the app and I want to be right there in the middle.



If you were to run the project, you could resize the height and width of the app and see how it responds to different screen sizes; the green rectangle with its center aligned vertically to the red rectangle and its horizontal alignment centered to the center of the application.

Next, I add a yellow Rectangle:

```
37          <Rectangle Name="YellowRectangle"
38                     Fill="Yellow"
39                     MinWidth="50"
40                     MinHeight="50"
41                     RelativePanel.AlignBottomWithPanel="True"
42                     RelativePanel.AlignTopWith="PurpleRectangle"/>
```

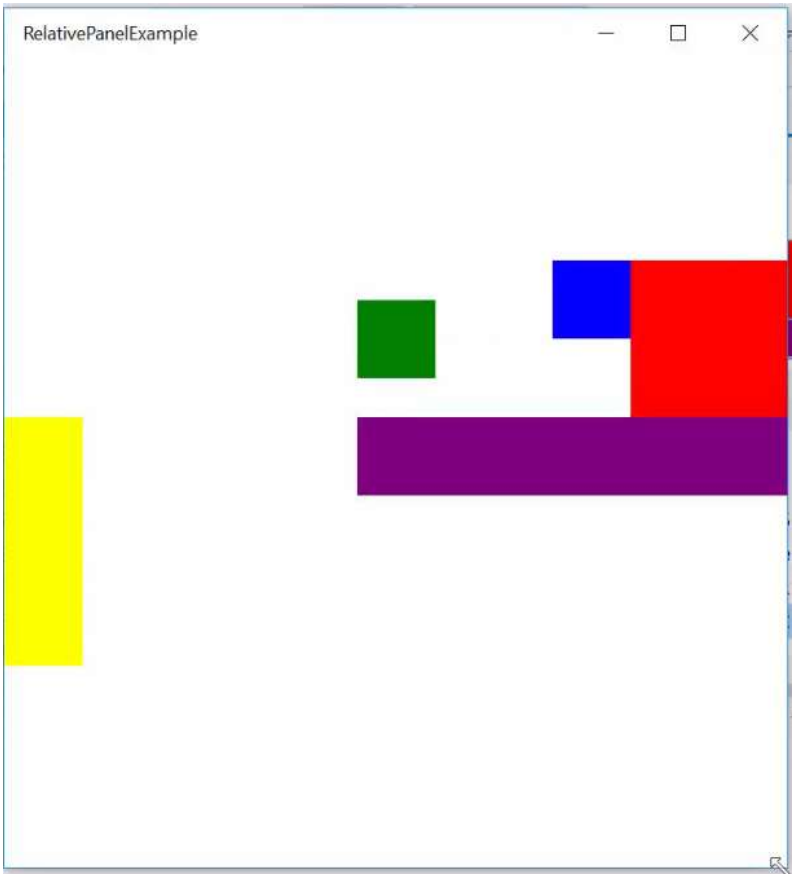
You can see that I set the minimum width to 50 and the minimum height to 50. And why this is important is because align the bottom of our rectangle with the bottom of the panel and align the top with the purple rectangle.

And so I'll define the purple rectangle above it:

```
45          <Rectangle Name="PurpleRectangle"
46                     Fill="Purple"
47                     MinWidth="50" MinHeight="50"
48                     RelativePanel.Below="RedRectangle"
49                     RelativePanel.AlignRightWith="RedRectangle"
50                     RelativePanel.AlignLeftWith="GreenRectangle"
51                      />
```

I want the purple rectangle to be positioned below the red rectangle. I also want its right aligned with the right of the red rectangle, and I want the left aligned with the green rectangle.

When I run the project we can observe the results:

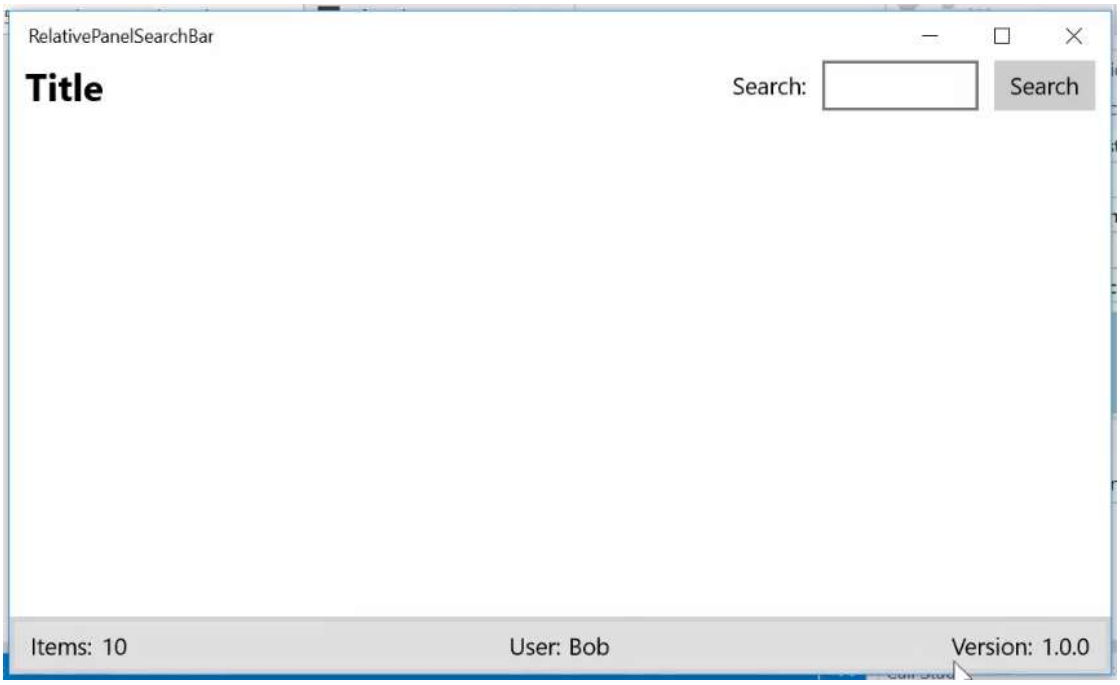


As I re-position the actual size of the Purple and Yellow rectangles will change. In both cases you'll see that by resizing, we will get a larger width or height, a larger rectangle.

This is why you can perform interesting positional logic with the RelativePanel because you can set sides relative to other controls or relative to the panel itself and things can automatically resize themselves.

Why is this important? Hopefully this will be evident in another project I created named RelativePanelSearchBar.

This project contains the beginnings of a Windows 10 application with a search bar on the right-hand side. It doesn't look finished yet, but you can see how no matter what, over on the left-hand side, or on the right-hand side rather, we have it always docked to the right and we have the title to the left.



Furthermore, I wanted to show that we can create a little status bar that's always docked to the bottom. Depending on the information, it's either aligned to the left, aligned to the right, or aligned center. So let's see how we achieve this effect in our XAML.

First, the Grid's RowDefinitions:

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

The top and the bottom rows are Auto so just take up enough room, just what you need, and then the middle area will be star sizing and so we didn't do anything in there.

Next, I added a RelativePanel to sit in the top row. Inside of the RelativePanel a SearchButton whose right side is aligned with the panel, a TextBox that is to the left of the SearchButton, and so on.

```
        <RelativePanel>
            <Button Name="SearchButton"
                    Content="Search"
                    RelativePanel.AlignRightWithPanel="True"
                    Margin="0,0,10,0"  />
            <TextBox Name="SearchTextBox"
                     RelativePanel.LeftOf="SearchButton"
                     Width="100"
                     Margin="0,0,10,0" />
            <TextBlock Text="Search:"
                       RelativePanel.LeftOf="SearchTextBox"
                       RelativePanel.AlignVerticalCenterWithPanel="True"
                       Margin="0,0,10,0" />
            <TextBlock Text="Title"
                       FontSize="24"
                       FontWeight="Bold"
                       Margin="10,0,0,0" />
        </RelativePanel>
```

Next in the third row a status bar creating using control called a Border and a Border. The Border just creates a little layout control that provides us the ability to style the background color and the stroke around the background color.

```
<Border BorderThickness="3" Background="#FFE0E0E0" Grid.Row="2" BorderBrush="#FFD2D2D2">
    <RelativePanel>
        <TextBlock Name="ItemsTextBlock"
                   Text="Items:"
                   RelativePanel.AlignLeftWithPanel="True"
                   Margin="10,5,0,5" />
        <TextBlock Text="10"
                   RelativePanel.RightOf="ItemsTextBlock"
                   Margin="5,5,0,5" />

        <TextBlock  Text="Version:"
                    RelativePanel.LeftOf="VersionTextBlock"
                    Margin="0,5,5,5" />
        <TextBlock Name="VersionTextBlock"
                   Text="1.0.0"
                   RelativePanel.AlignRightWithPanel="True"
                   Margin="0,5,10,5" />
```

So we're just putting our RelativePanel inside of the Border and then we will do our work inside of the RelativePanel itself.

There are three sections to the RelativePanel. The first TextBlock on the left-hand side will just contain the text “Items:”. In this example, “items” could represent items that need to be addressed. It uses AlignLeftWithPanel. The next TextBlock aligns itself to the right of that first TextBlock and would contain a numeric value.

The right-hand side works the same way except with the text “Version”.

And then the third case – the middle --I actually create a StackPanel because if you were to set one of the TextBoxes using the RelativePanel.AlignedHorizontalCenterWithPanel, then it will be in the absolute center and anything else that you need to the left of it or to the right of it would then be off center. Since we need both TextBlocks to be centered I put them in a StackPanel which ensures the TextBlocks are centered no matter the width of either one.

```
<StackPanel RelativePanel.AlignHorizontalCenterWithPanel="True"
            Orientation="Horizontal">
    <TextBlock Text="User:"
               Margin="0,5,5,5" />
    <TextBlock Text="Bob"
               Margin="0,5,0,5" />
</StackPanel>
```

One note of caution regarding the RelativePanel; you could potentially get yourself into a circular reference. Example: I want object number one to the left of object number two, object number three to the left of object number two, object number four to the left of object number three, object number one to the left of number four.

Furthermore, you can set multiple relationships that target the same edge of the element and when you do that, you might have conflicting relationships in your layout as a result.

So whenever this happens, there are actually an order of events just like whenever you are working in a math problem or even in code that parentheses can dictate the order of operation. There is an order of operation with how these relationships are deciphered in this order.

- So first priority will be panel alignment, so align me to the left or the right of the panel.
- Then the second one will be Sibling Alignment relationship, so align me with the top of this control, align me with the left of that control,
- Then the third and the lowest priority would be Sibling Positional relationships. Set me above this control, below this control, to the left of this control, okay.

UWP-018 - XAML Layout with the SplitView

The second new layout control in the Universal Windows Platform API is the SplitView which allows us to create a panel that can be displayed or hidden. The SplitView features an animation that provides a sliding effect into view and out of view. The most practical use of the SplitView is to implement Windows 10 hamburger style navigation.

There are two parts to a SplitView. One might be optional. The other is required. The part that's hidden by default that you display, that's called the “Pane”. The part that's already displayed and can either be overlapped, or it can be pushed over, that's called the “Content”.

Since the SplitView is a layout control it is intended to host other XAML Controls in the Pane and the Content sections. In the SplitView Pane, you would create the hamburger navigation style by adding an icon for a section or feature of your application and add text next to it.

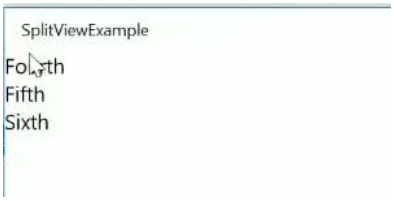
To demonstrate its usage, I created a project named "SplitViewExample." I add a SplitView and inside the SplitView I create a Pane, and a Content area.

```
10      <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
11          <SplitView >
12              <SplitView.Pane>
13
14              </SplitView.Pane>
15              <SplitView.Content>
16
17              </SplitView.Content>
18          </SplitView>
19
20      </Grid>
21  </Page>
```

Next, in both the Pane and Content I’ll add a StackPanel to contain a series of TextBlocks. I’ll set the Text to values that will help us visually distinguish where the Pane ends and the Content begins.

```
10      <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
11          <SplitView >
12              <SplitView.Pane>
13                  <StackPanel>
14                      <TextBlock Text="First" />
15                      <TextBlock Text="Second" />
16                      <TextBlock Text="Third" />
17                  </StackPanel>
18              </SplitView.Pane>
19              <SplitView.Content>
20                  <StackPanel>
21                      <TextBlock Text="Fourth" />
22                      <TextBlock Text="Fifth" />
23                      <TextBlock Text="Sixth" />
24                  </StackPanel>
```

At this early stage if I were to run the app we would only see the Content area:



How do we display the pane? We do this programmatically using C# in response to some event such as a button's click event. To access the Pane programmatically we will have to first of all give our SplitView a name. So I'll name it "MySplitView".

Next I'll add a Button with the content "Click Me" and handle the Click event by adding the attribute Click="Button_Click".

```
25                </SplitView.Content>
26            </SplitView>
27            <Button Content="Click Me" Click="Button_Click" />
28        </Grid>
29    </Page>
```

I put my mouse cursor inside the text "Button_Click" and press F12 on my keyboard which opens the MainPage.xaml.cs code behind and puts my mouse cursor inside of the Button_Click event handler method stub. I add one line of code that will set the SplitView's IsPaneOpen property to the opposite of its current value, so:

```
29
30        private void Button_Click(object sender, RoutedEventArgs e)
31        {
32            MySplitView.IsPaneOpen = !MySplitView.IsPaneOpen;
33        }
34    }
```

Back in the MainPage.xaml, I'll change this outermost Grid to a StackPanel.

```
1   <Page
2       x:Class="SplitViewExample.MainPage"
3       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
4       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
5       xmlns:local="using:SplitViewExample"
6       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
7       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
8       mc:Ignorable="d">
9
10      <StackPanel Background="{ThemeResource ApplicationPageBackgroundThemeBrus
11          <SplitView Name="MySplitView">
12              <SplitView.Pane>
13                  <StackPanel>
14                      <TextBlock Text="First" />
```

Now when I run my app in debug mode I can click the "Click Me" button to show and hide the Panel.

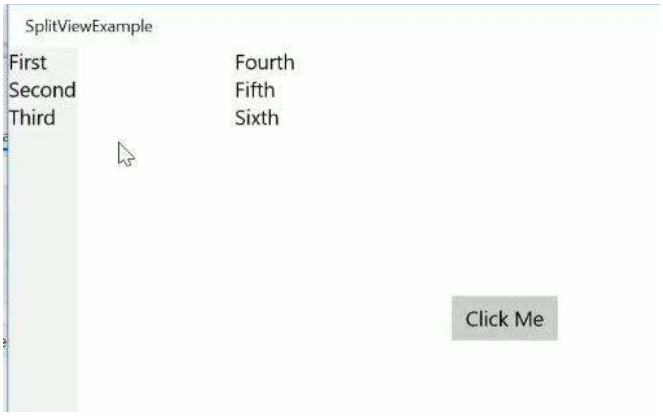
We're not finished configuring the SplitView's Panel. First, I'll modify the StackPanel's Orientation="Horizontal".

Next, I'll set the SplitView's DisplayMode. The DisplayMode is one of the most important properties that we can set on the SplitView, because it will dictate how it actually operates. There are four options here.

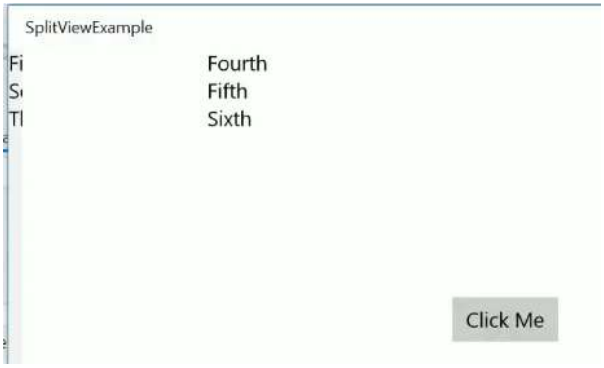
I'll set it to Inline then run the application.

When set to Inline the Pane pushes the Content over. If I were to change the DisplayMode to Overlay the Pane would completely cover up the Conetnt.

Next, I will set the CompactPaneLength to a relative small value, like 50. Just enough to display an icon. Then I'll set the DisplayMode to CompactOverlay. CompactOverlay is a variation on Overlay that will display just a little sliver of what's underneath in the Pane.



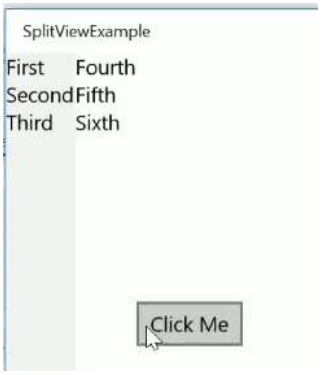
Let's change that even to just 10 pixels, just so we can barely see it creeping out.



You can just barely see First Second and Third creeping through there on the very most left side. Then when we click "Click Me," the Pane covers up whatever we see in the Content area.

CompactInline is exactly like Inline except it shows a sliver of Pane area as well. Just like Inline it will push the Content area over when it is opened.

We can also adjust a property called OpenPaneLength. If we were to set it to 50 pixels it would restrict the width of the Pane when it is in an opened state:



As you can see you're given a lot of latitude to build out your SplitView how you want to implement your Pane. In this case I just used TextBlocks however that's not the best choice. We would want to either use a Button or something that's clickable so that when we click it, we navigate to something in the main area (Content) of our application -- some feature or some different information. In a later lesson I'll demonstrate a simple implementation of the hamburger navigation featuring the SplitView.