UWP-020 - Common XAML Controls - Part 1

This lesson will cover some of the most basic input controls that we haven't already talked about up to this point. We will need some of these input controls for future projects. We will cover seven in rapid fire in this lesson.

I've created a new project named: “ControlsExamplePart1”.

To setup this example I’ll drag and drop an image from my desktop into the Assets folder. This image that should be included with the zipped folder associated with this lesson.

There are really three things that I want to learn about each control:

First of all, how do I display it on screen?

Secondly, how do I handle the major events that it will emit whether it be a tapped or a selection changed, whatever the case might be?

Third, how do I retrieve the value out of that control, the current selection, whatever has been typed in, whatever has been checked or unchecked, etc.?

I’ll begin by adding a Grid, RowDefinitions and ColumnDefinitions:

```
10      <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}" Margin="10,10,0,0">
11          <Grid.RowDefinitions>
12              <RowDefinition Height="Auto" />
13              <RowDefinition Height="Auto" />
14              <RowDefinition Height="Auto" />
15              <RowDefinition Height="Auto" />
16              <RowDefinition Height="Auto" />
17              <RowDefinition Height="Auto" />
18              <RowDefinition Height="Auto" />
19              <RowDefinition Height="Auto" />
20              <RowDefinition Height="Auto" />
21              <RowDefinition Height="Auto" />
22          </Grid.RowDefinitions>
23          <Grid.ColumnDefinitions>
24              <ColumnDefinition Width="Auto" />
25              <ColumnDefinition Width="*" />
26          </Grid.ColumnDefinitions>
```

Next, I’ll add the first input control is the CheckBox, a very simple “yes/no” control represented as a box that is empty (no / false) or has a checkmark inside of it (yes / true).

The CheckBox has a Content that will be displayed in the label next to the box. There’s also a Tapped event that we will handle with C#.

```
28          <TextBlock Grid.Row="0" Text="CheckBox" VerticalAlignment="Center" />
29          <StackPanel Grid.Column="1"
30                      Margin="20,10,0,10"
31                      Orientation="Horizontal">
32              <CheckBox Name="MyCheckBox"
33                        Content="Agree?"
34                        Tapped="MyCheckBox_Tapped" />
35              <TextBlock Name="CheckBoxResultTextBlock" />
36          </StackPanel>
```

In the Tapped event we'll display the current value of the CheckBox by retrieving the IsChecked property:

```
30          private void MyCheckBox_Tapped(object sender, TappedRoutedEventArgs e)
31          {
32            CheckBoxResultTextBlock.Text = MyCheckBox.IsChecked.ToString();
33          }
```

When I run the app, you can see the result of tapping the checkbox:

Next, the RadioButton is similar to a CheckBox however it will allow you to add multiple potential values each grouped together by a GroupName property. In other words, all the radio buttons that belong together must have the same GroupName.

```
        <TextBlock Grid.Row="2"
                   Text="RadioButton"
                   VerticalAlignment="Center" />
        <StackPanel Grid.Row="2"
                    Grid.Column="1"
                    Orientation="Horizontal"
                    Margin="20,10,0,10">
            <RadioButton Name="YesRadioButton"
                         Content="Yes"
                         GroupName="MyGroup"
                         Checked="RadioButton_Checked" />
            <RadioButton Name="NoRadioButton"
                         Content="No"
                         GroupName="MyGroup"
                         Checked="RadioButton_Checked" />
            <TextBlock Name="RadioButtonTextBlock" />
        </StackPanel>
```

I can handle the Checked event, where I use the tertiary operator to display whether the Yes or No RadioButton IsChecked:

```
        private void RadioButton_Checked(object sender, RoutedEventArgs e)
        {
            RadioButtonTextBlock.Text = (bool)YesRadioButton.IsChecked ? "Yes" : "No";
        }
```

When I run the app and select the Yes RadioButton:

You can have two groups of radio buttons, and the selections will be unique within each of those groups. And we could add as many RadioButtons as we want but it typically only works well with four or five selections at most. If you have more than that, then you'll probably want to move onto a different type of control for input.

Next, the ComboBox which allows you to create a number of ComboBoxItems, each with their own value.

```
57          <TextBlock Grid.Row="3"
58                     Text="ComboBox"
59                     Name="MyComboBox"
60                     VerticalAlignment="Center" />
61          <StackPanel Orientation="Horizontal"
62                      Grid.Row="3"
63                      Grid.Column="1"
64                      Margin="20,10,0,10">
65              <ComboBox SelectionChanged="ComboBox_SelectionChanged" >
66                  <ComboBoxItem Content="Fourth" />
67                  <ComboBoxItem Content="Fifth" />
68                  <ComboBoxItem Content="Sixth" IsSelected="True" />
69              </ComboBox>
70              <TextBlock Name="ComboBoxResultTextBlock" />
71          </StackPanel>
72
```

We're handling the SelectionChanged event so whenever somebody makes a different selection in the ComboBox, handle that and then take whatever the new selection was and display that in the label below or in the TextBlock below.

```
40          private void ComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
41          {
42              if (ComboBoxResultTextBlock == null) return;
43
44              var combo = (ComboBox)sender;
45              var item = (ComboBoxItem)combo.SelectedItem;
46              ComboBoxResultTextBlock.Text = item.Content.ToString();
47          }
```

When I run the app and select an ComboBoxItem:

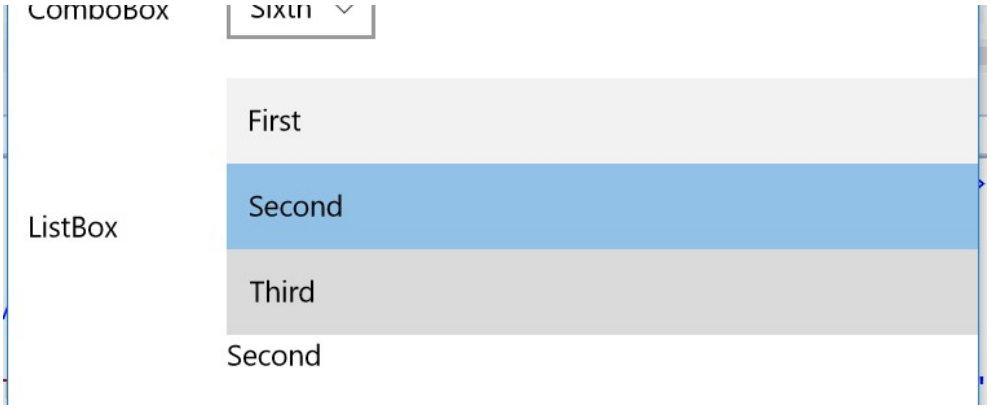
Next, the ListBox which will be a pretty important control when we are creating our hamburger navigation. There are a bunch of different ways that we can create that hamburger navigation that's popular in Windows 10. I choose the ListBox for reasons that I'll demonstrate in an upcoming lesson to follow up on a promise that I made earlier.

```
74          <TextBlock Grid.Row="4" Text="ListBox" VerticalAlignment="Center" />
75          <StackPanel Grid.Row="4" Grid.Column="1"  Margin="20,10,0,10">
76              <ListBox Name="MyListBox"
77                       SelectionMode="Multiple"
78                       SelectionChanged="ListBox_SelectionChanged">
79                  <ListBoxItem Content="First" />
80                  <ListBoxItem Content="Second" />
81                  <ListBoxItem Content="Third" />
82              </ListBox>
83              <TextBlock Name="ListBoxResultTextBlock" />
84          </StackPanel>
85
```

Notice that I set the SelectionMode to Multiple, meaning that the user can select more than one option. Then, I handle the SelectionChanged event and I get an array of selected items with a clever LINQ statement and join the items of the array together separated by a comma:

```
49          private void ListBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
50          {
51              var selectedItems = MyListBox.Items.Cast<ListBoxItem>()
52                                       .Where(p => p.IsSelected)
53                                         .Select(t => t.Content.ToString())
54                                           .ToArray();
55
56              ListBoxResultTextBlock.Text = string.Join(", ", selectedItems);
57          }
```

When I run the app, I can select one or more items to list them to see the selections in the TextBlock below.



Next, the Image control will be used in an upcoming exercise so I want to explain one of its most important properties. The Image control displays an image.

```
86          <TextBlock Grid.Row="5" Text="Image" VerticalAlignment="Center" />
87          <Image Source="Assets/logo.png"
88                 HorizontalAlignment="Left"
89                 Width="250"
90                 Height="50"
91                 Grid.Row="5"
92                 Grid.Column="1"
93                 Stretch="UniformToFill"
94                 Margin="20,10,0,10" />
95
```

First, I set the Source. In this case the Source is going to be relative to the root of our project. So we will look in the Assets folder for a logo.png. We could also supply it a location across the internet or some other folder in our application.

Next, I set the Stretch property. And there are a couple different options.

When the Stretch property is set to None, no stretching will be used. The image will be cropped to the available width and height of the control.

When the Stretch property is set to Fill, the image will be stretched to use all available width and height of the control.

When the Stretch property is set to Uniform, the image will be resized to fit completed in the available width and height meaning that it will be constrained by the smaller of the two dimensions so that there's no distorting the image:

Finally, when the Stretch property is set to UniformToFill, the image will be resized to fit the in the available width OR height, but not both. In this case, the image is constrained by the height but it is displayed in full width.

Next, the ToggleButton which is a “state-ful” Button, so you can click it on and you can click it off. There is also an option to make it a ThreeState button, so you can click it and give it no value, click it on or click it off. And so I've set the IsThreeState="True" so that we can see it at work.

```
96          <TextBlock Grid.Row="7" Text="ToggleButton" VerticalAlignment="Center"  />
97          <StackPanel Orientation="Horizontal"
98                      Grid.Row="7"
99                      Grid.Column="1"
100                     Margin="20,10,0,10" >
101             <ToggleButton Name="MyToggleButton"
102                           Content="Premium Option"
103                           IsThreeState="True"
104                           Click="MyToggleButton_Click" />
105             <TextBlock Name="ToggleButtonResultTextBlock" />
106         </StackPanel>
```

Furthermore I handle the Click event where I grab the current value of IsChecked and display it in the TextBlock:

```
59          private void MyToggleButton_Click(object sender, RoutedEventArgs e)
60          {
61              ToggleButtonResultTextBlock.Text = MyToggleButton.IsChecked.ToString();
62          }
63      }
```

When I run the application I can toggle through the various states, such as the “on” state:

Then the last control that we will look at is similar to the ToggleButton that except it's called the ToggleSwitch. The ToggleSwitch allows us to toggle between two states. And we see this control used often in the Settings panel in Windows 10. What you can do that's interesting is actually set a Content

area for what should be displayed whenever the Toggle is in an off position or an on position. You can also handle other events but this will be primarily how it's used to display the current state of the toggle.

```
108         <TextBlock Grid.Row="8"
109                    Text="ToggleSwitch"
110                    VerticalAlignment="Center" />
111         <StackPanel Grid.Row="8"
112                     Grid.Column="1"
113                     Margin="20,10,0,10" >
114             <ToggleSwitch>
115                 <ToggleSwitch.OffContent>
116                     <TextBlock Text="I'm off right now." />
117                 </ToggleSwitch.OffContent>
118                 <ToggleSwitch.OnContent>
119                     <TextBlock Text="I'm on!" />
120                 </ToggleSwitch.OnContent>
121             </ToggleSwitch>
122         </StackPanel>
123
```

When I run the application and toggle the ToggleSwitch the TextBlock is set to either "I'm off right now." or "I'm on!"