



.NET Technologies using C#

C # --- Introduction to UWP

Instructor: Mahboob Ali

Email: mahboob.ali@sheridancollege.ca

Course: PROG32356

Adapted from: Gursharan Singh Tatla

Introduction

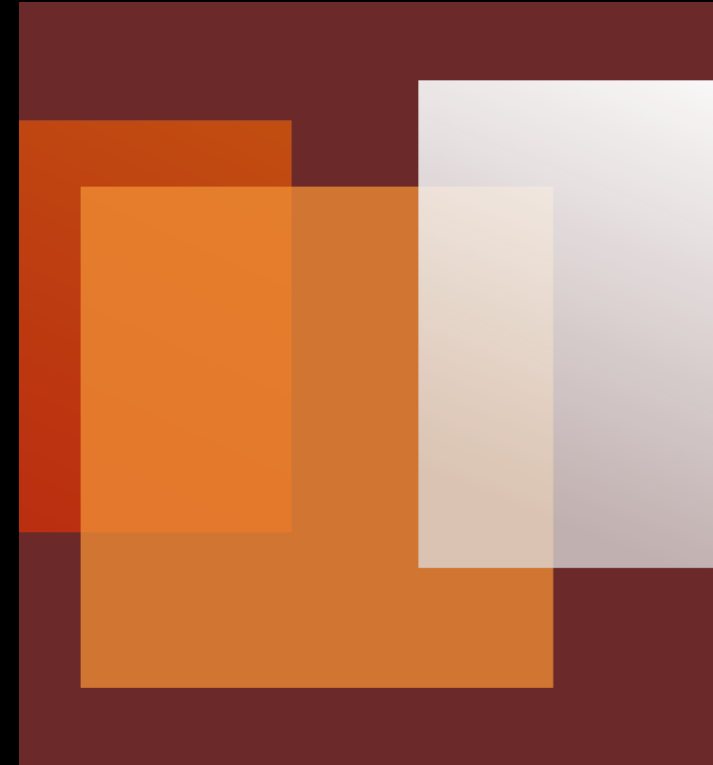
- UWP stands for Universal Windows Platform.
- UWP is Microsoft's latest technology solution to build applications for Microsoft's Windows suite of operating systems.
- It provides a guaranteed API layer across multiple device types.
- You can create a single app package that can be uploaded to a single store to be distributed to reach all the device types your app can run on.
- These devices include Windows 10 desktops, laptops, and tablets, the Xbox One and later video game systems, and Mixed Reality Headsets like Microsoft HoloLens.
- UWP provides standard mechanisms to detect the capabilities of the current device and then activate additional features of your app to fully take advantage of them.
- UWP with XAML provides layout panels that adapt how they display their child controls to make the most of the device they are currently running on.
- It is the Windows app equivalent of web page responsive design.
- They also provide visual state triggers to alter the layout based on dynamic changes, such as the horizontal or vertical orientation of a tablet.

Understanding Fluent Design System

- Microsoft's **Fluent Design System** will be delivered in multiple waves, rather than as a "Big Bang" all in one go, to help developers slowly migrate from traditional styles of user interface to more modern ones.
- Wave 1, available in Windows 10 Fall Creators Update, released in October 2017 and refined in the subsequent waves as part of biannual Windows 10 updates, included the following features:
 - Acrylic material
 - Connected animations
 - Parallax views
 - Reveal lighting

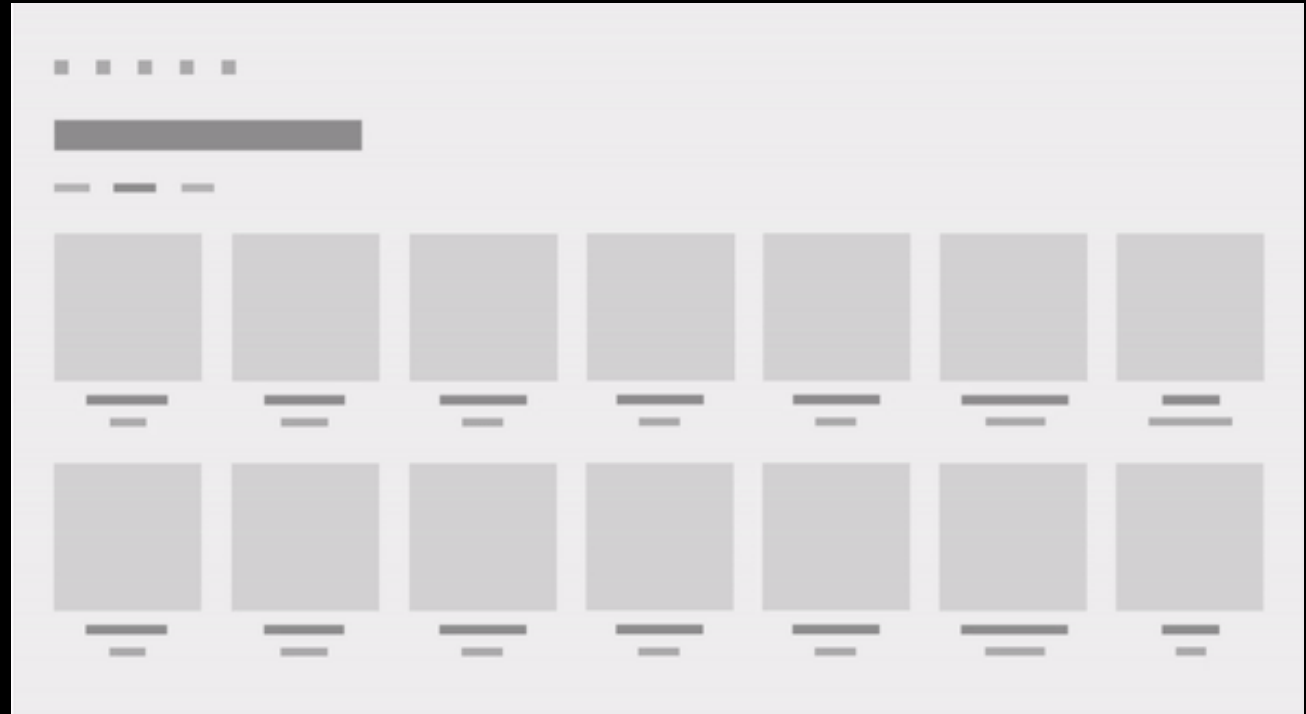
Filling User Interface Elements with Acrylic Brushes

- Acrylic material is a semi-transparent blur-effect brush that can be used to fill user interface elements to add depth and perspective to your apps.
- Acrylic can show through what is in the background behind the app, or elements within the app that are behind a pane.
- Acrylic material can be customized with varying colors and transparencies.
- More information:
 - <https://docs.microsoft.com/en-us/windows/uwp/design/style/acrylic>



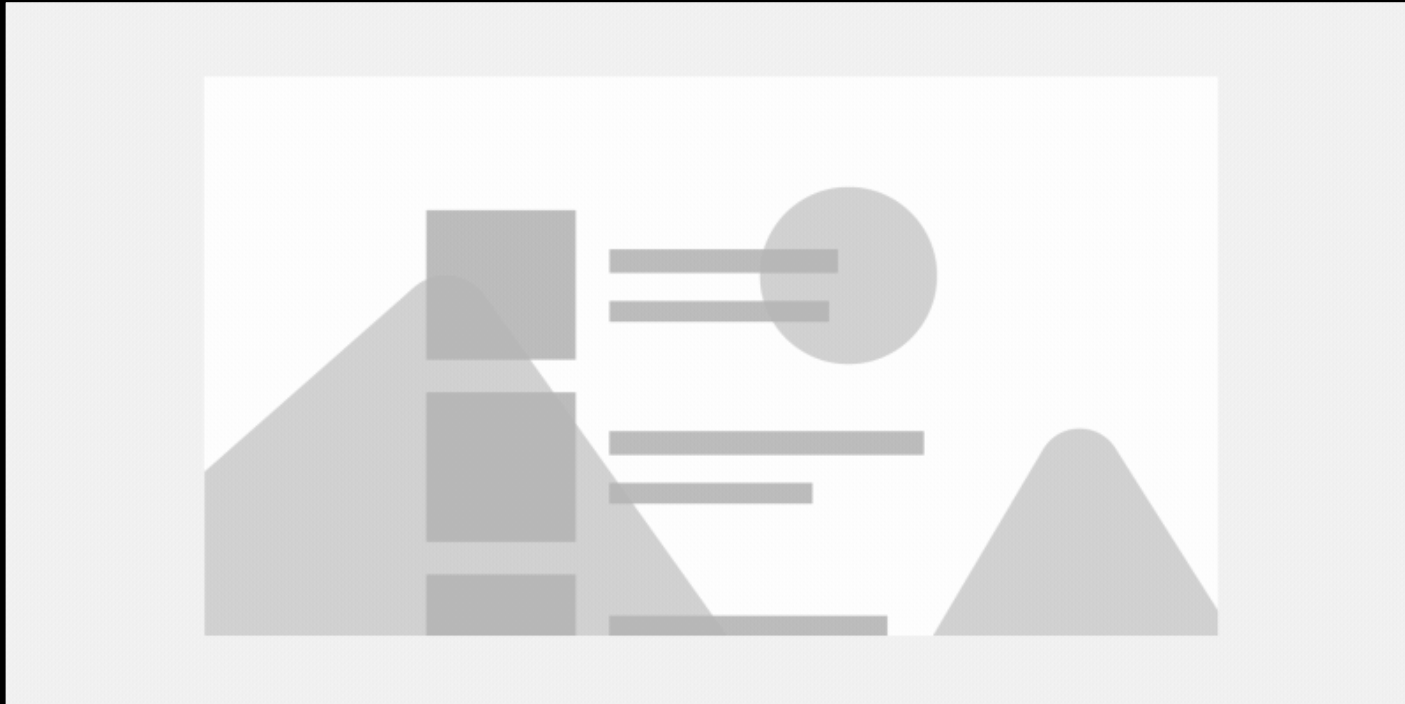
Connecting User Interface Elements with Animations

- When navigating around a user interface, animating elements to draw connections between screens helps users to understand where they are and how to interact with your app.
- More information:
 - <https://docs.microsoft.com/en-us/windows/uwp/design/motion/connected-animation>



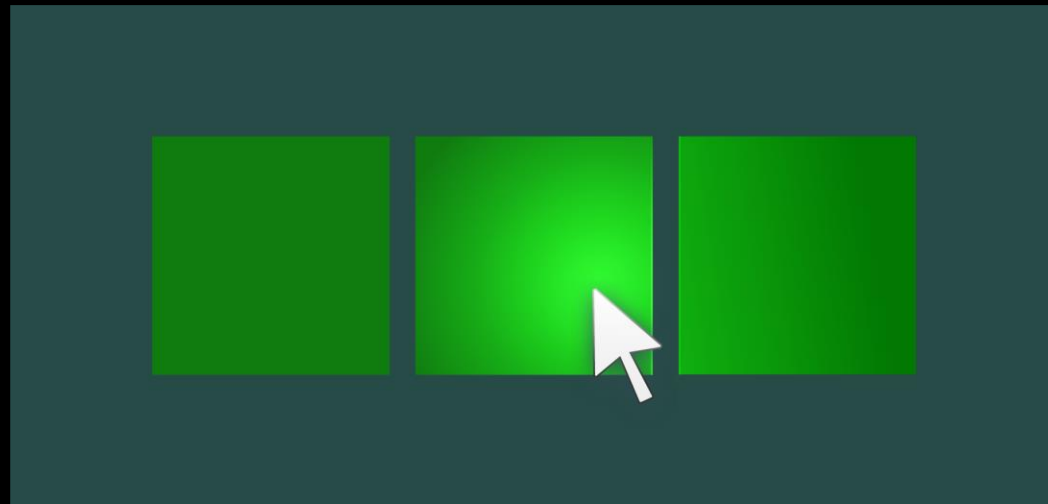
Parallax Views and Reveal Lighting

- **Parallax views** are backgrounds, often images, that move at a slower rate than the foreground during scrolling to provide a feeling of depth and give your apps a modern feel.
- More information:
 - <https://docs.microsoft.com/en-us/windows/uwp/design/motion/parallax>



Parallax Views and Reveal Lighting

- **Reveal lighting** helps the user understand which of the visual elements in the user interface is an interactive element by lighting up as the user moves their mouse cursor over that element to draw their focus.
- More information:
 - <https://docs.microsoft.com/en-us/windows/uwp/design/style/reveal>



Understanding XAML Standard

- In 2006, Microsoft released WPF, which was the first technology to use XAML.
- Silverlight, for web and mobile apps, quickly followed.
- WPF is still used today to create Windows desktop applications; for example, Microsoft Visual Studio 2019 is partially built using WPF.
- XAML can be used to build parts of the following apps:
 - **UWP Apps** for Windows 10 devices, Xbox One, and Mixed Reality Headsets.
 - **WPF Apps** for the Windows desktop, including Windows 7 and later.
 - **Xamarin Apps** for mobile and desktop devices including Android, iOS, and macOS.

Choosing Common Controls

- There are lots of predefined controls that you can choose from for common user interface scenarios.

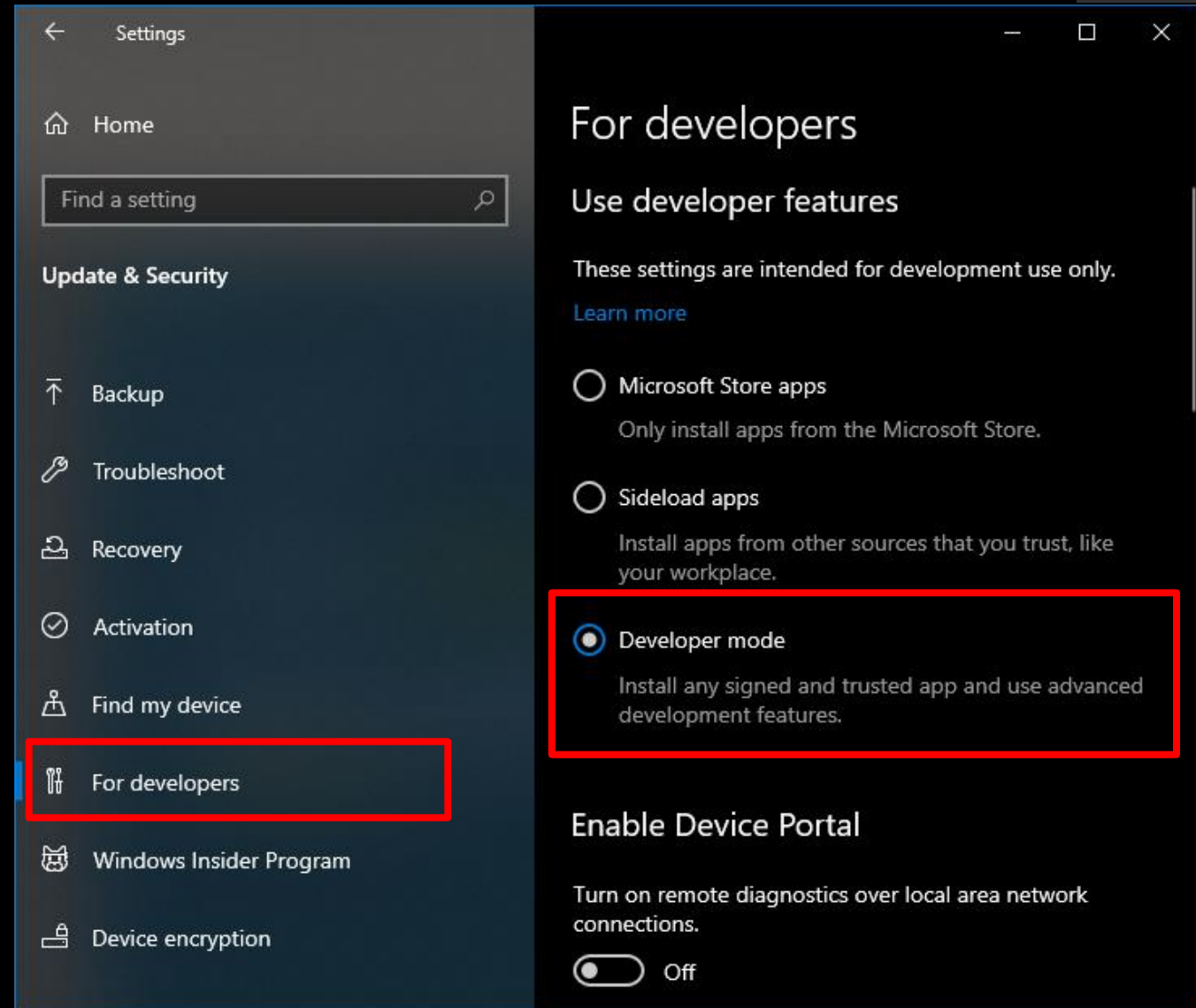
Controls	Description
Button, Menu, Toolbar	Executing actions
CheckBox, RadioButton	Choosing options
Calendar, DatePicker	Choosing dates
ComboBox, ListBox, ListView, TreeView	Choosing items from lists and hierarchical trees
Canvas, DockPanel, Grid, StackPanel, WrapPanel	Layout containers that affect their children in different ways
Label, TextBlock	Displaying read-only text
RichTextBox, TextBox	Editing text
Image, MediaElement	Embedding images, videos, and audio files
DataGrid	Viewing and editing data as quickly and easily as possible
Scrollbar, Slider, StatusBar	Miscellaneous user interface elements

Creating a Modern Windows App

- We will start by creating a simple UWP app, with some common controls and modern features of Fluent Design like acrylic material.

Enabling Developer Mode (Windows 10)

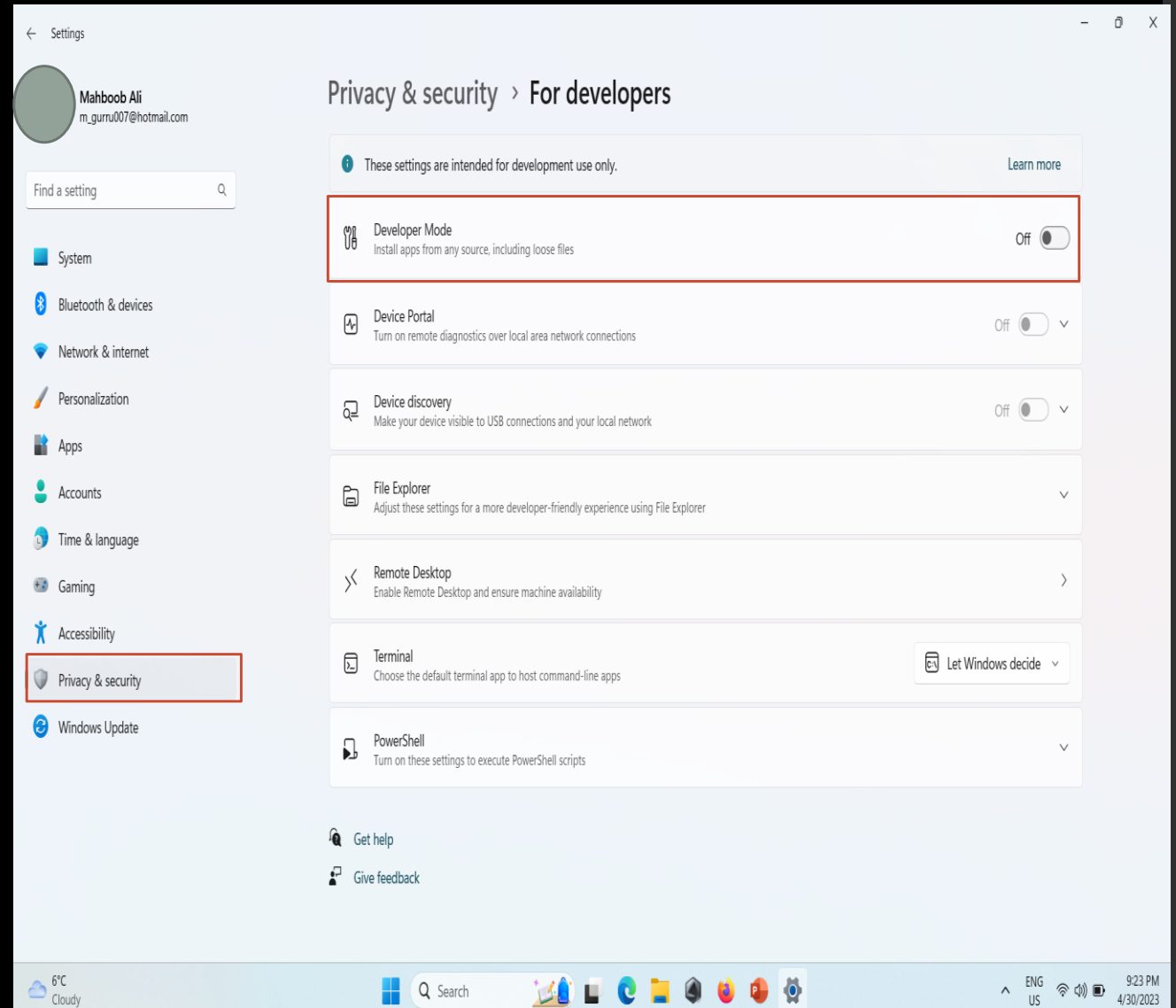
- To be able to create apps for UWP, you must enable developer mode in Windows 10.
1. Navigate to **Start | Settings | Update & Security | For developers**, and then click on **Developer mode**.
 2. Accept the warning about how it "could expose your device and personal data to security risk or harm your device," and then close the **Settings app**.



Enabling Developer Mode – Windows 11

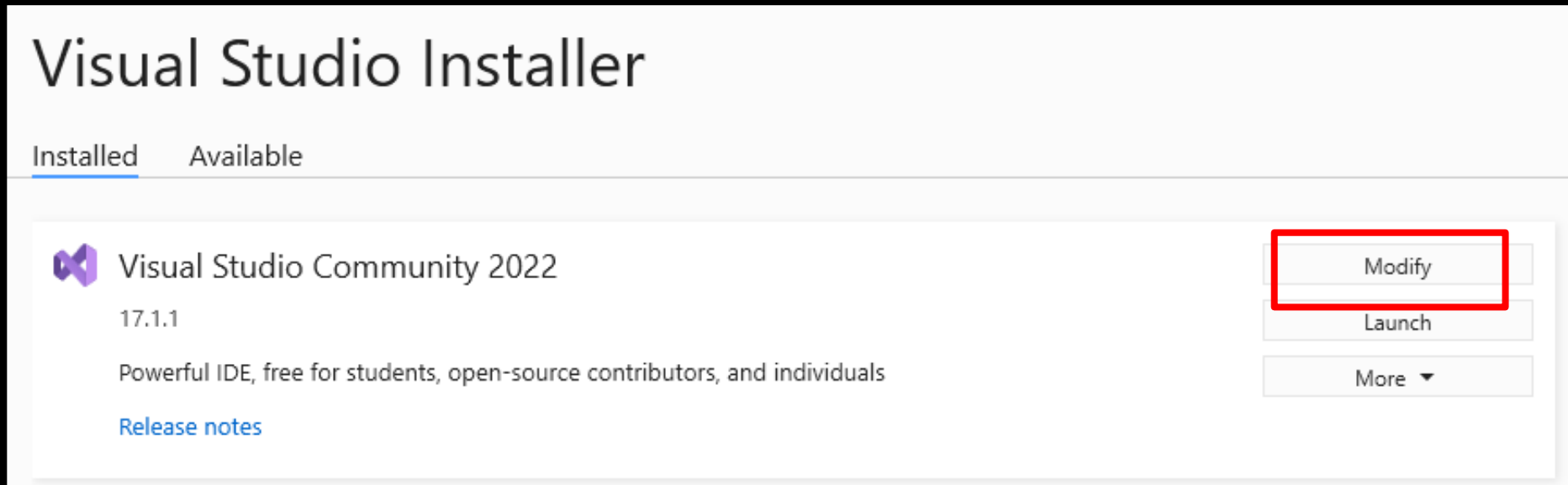
- For Windows 11:

1. Navigate to **Start | Settings | Privacy & Security | For developers**, and then enable the **Developer mode**.
2. Accept the warning about how it "could expose your device and personal data to security risk or harm your device," and then close the **Settings app**.



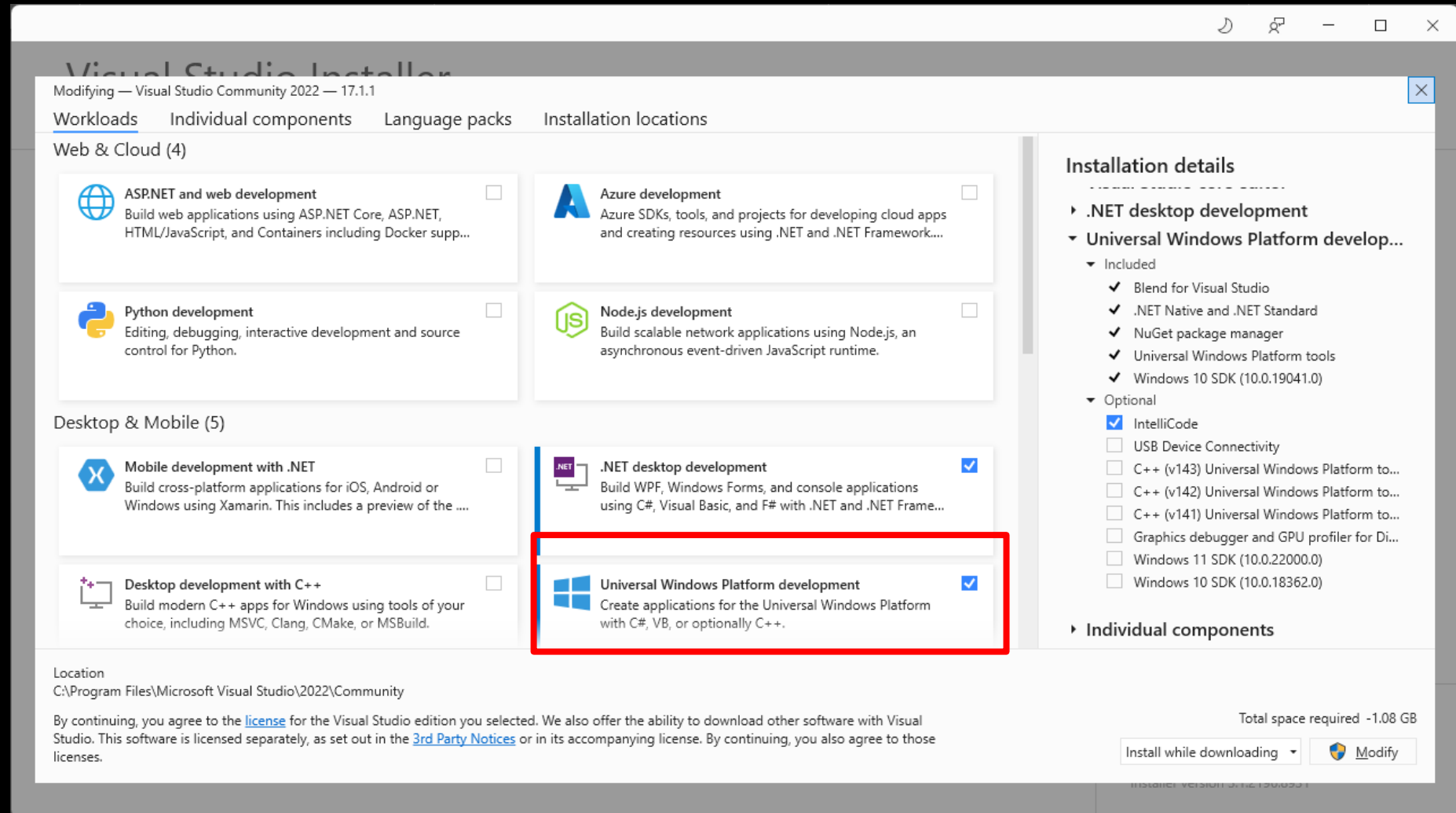
Installing Universal Windows Platform Development Workload

- You may not have the **Universal Windows App Development Tools** installed.
- To install, you must run the **Visual Studio Installer** again.
- You can do this from the **Start** menu by typing **Visual Studio Installer**.
- When you are presented with the list of installed products, click **Modify**.



Installing Universal Windows Platform Development Workload

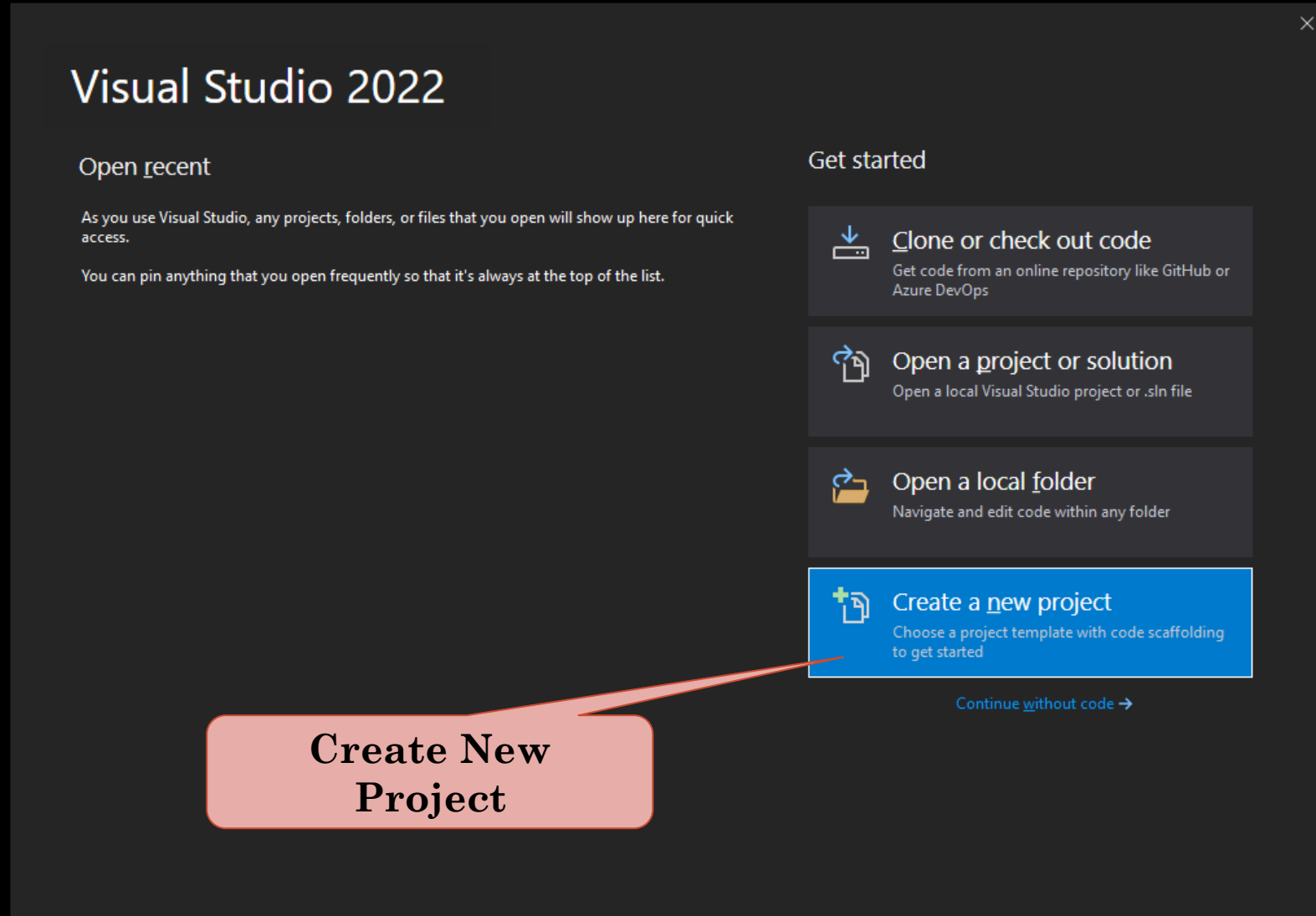
- Then check the **Universal Windows Platform development** option before clicking **Modify** again to install the selected options.



Creating a UWP Project

- Now you will create a new UWP app project.
- In Visual Studio 2019, create a new project.
- Enter **uwp** in search box, select the **Blank App (Universal Windows)** template for C#, and then click Next.
- For the Project name, enter **FluentUwpApp** and then click on Create.
- In the **New Universal Windows Platform Project dialog**, choose the **latest version of Windows 10** for Target Version and Minimum Version and click on OK.

Creating a UWP Project



Creating a UWP Project

The screenshot shows the 'Create a new project' dialog in Visual Studio. On the left, under 'Recent project templates', there is a list of templates: 'WPF Application' (C#), 'WPF App (.NET Framework)' (C#), 'Console App' (C#), and 'Blank App (Universal Windows)' (C#). The main area displays search results for 'uwp'. The first result is 'Blank App (Universal Windows)', which is highlighted. Below this result, the 'UWP' platform is selected. A callout points to the search bar containing 'uwp'. Another callout points to the 'Blank App (Universal Windows)' result. A third callout points to the 'UWP' platform selection. At the bottom, there are 'Back' and 'Next' buttons.

Create a new project

Recent project templates

- WPF Application C#
- WPF App (.NET Framework) C#
- Console App C#
- Blank App (Universal Windows) C#

Search for "uwp"

Select "Blank App (Universal Windows)"

Shows you list of most recent project templates

Back Next

Creating a UWP Project

Configure your new project

Blank App (Universal Windows) C# XAML Windows Xbox UWP Desktop

Project name

FluentUwpApp

Location

D:\OneDrive - Sheridan College\Visual Studio 2022 Projects\

Solution name ⓘ

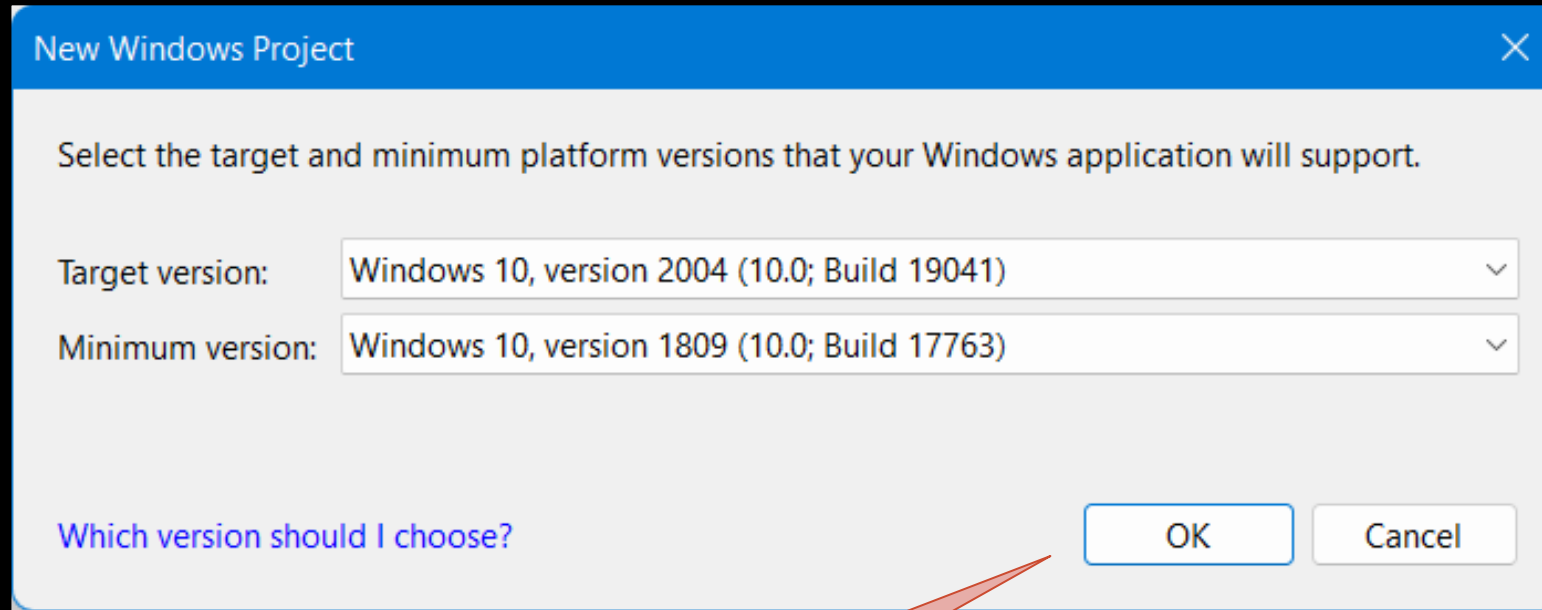
FluentUwpApp

☐ Place solution and project in the same directory

Back Create

Give your project a
name and location

Creating a UWP Project



New Windows Project

Select the target and minimum platform versions that your Windows application will support.

Target version: Windows 10, version 2004 (10.0; Build 19041)

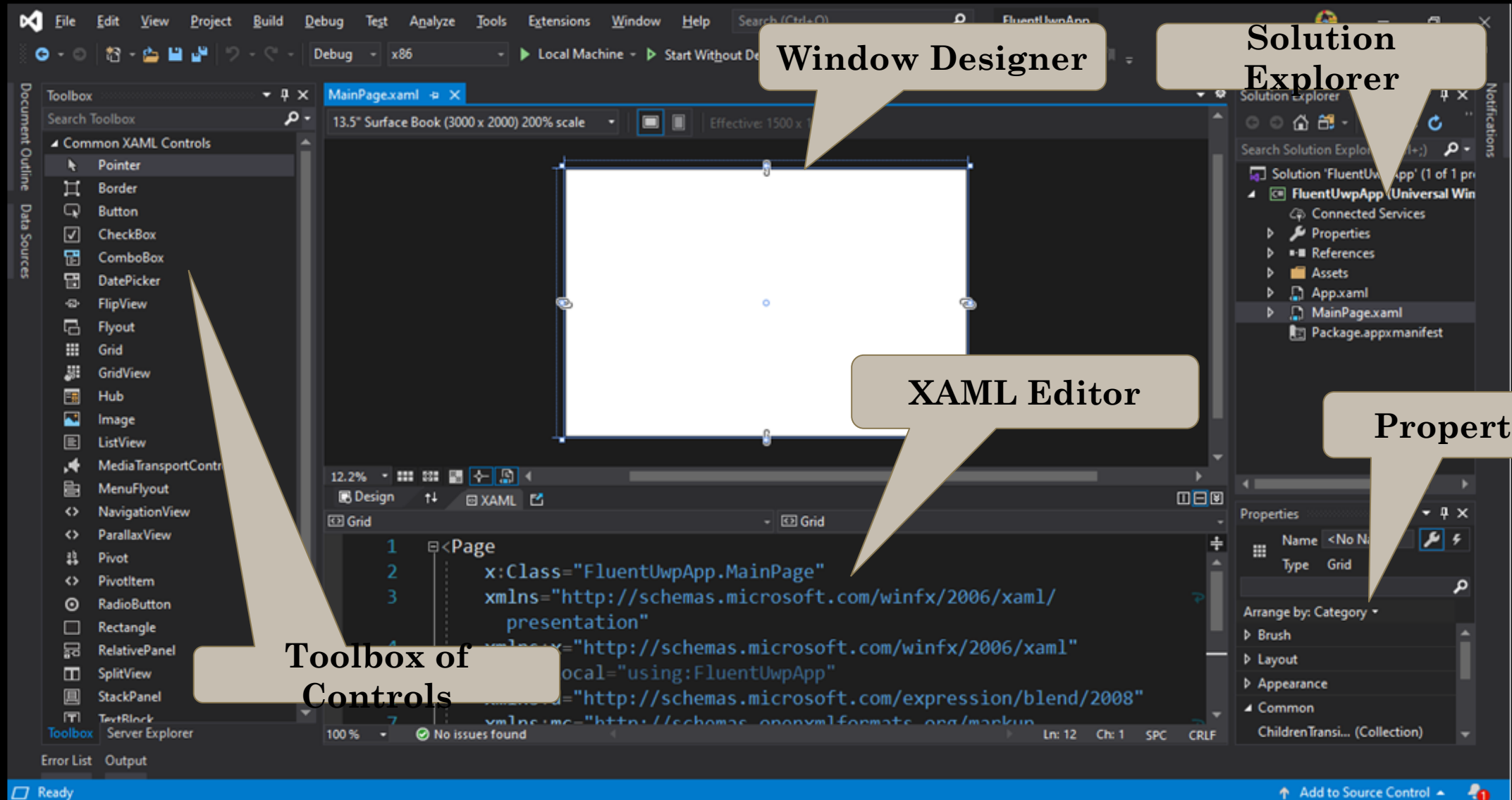
Minimum version: Windows 10, version 1809 (10.0; Build 17763)

[Which version should I choose?](#)

OK Cancel

Leave defaults and click OK

Creating a UWP Project

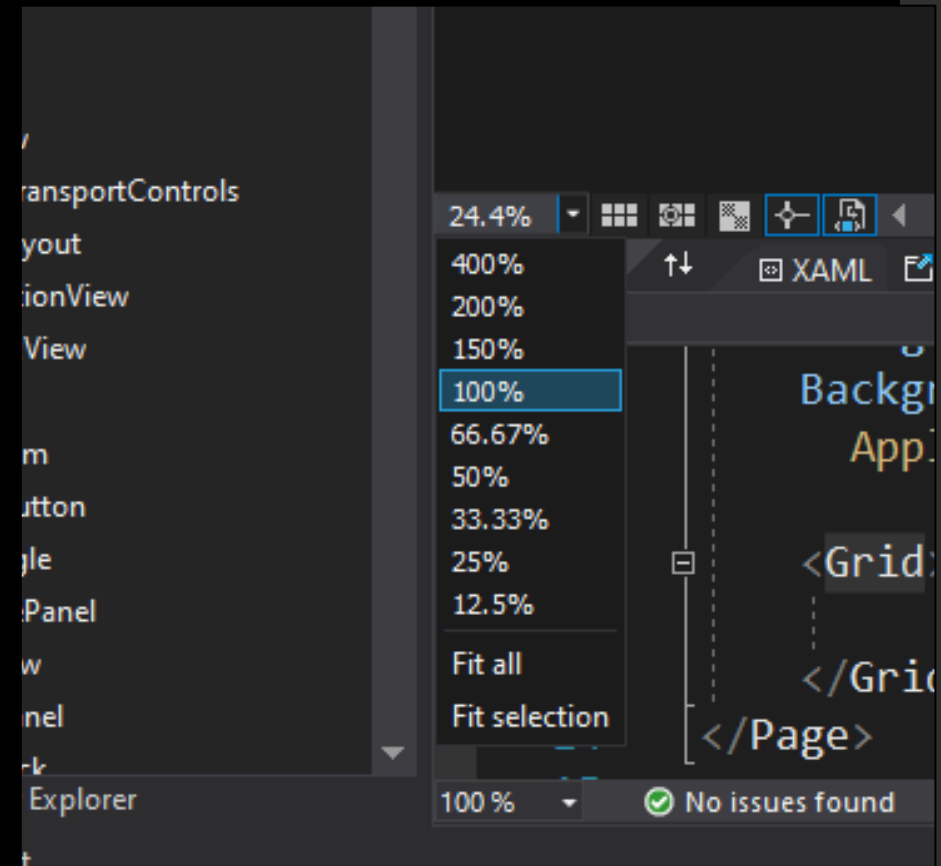
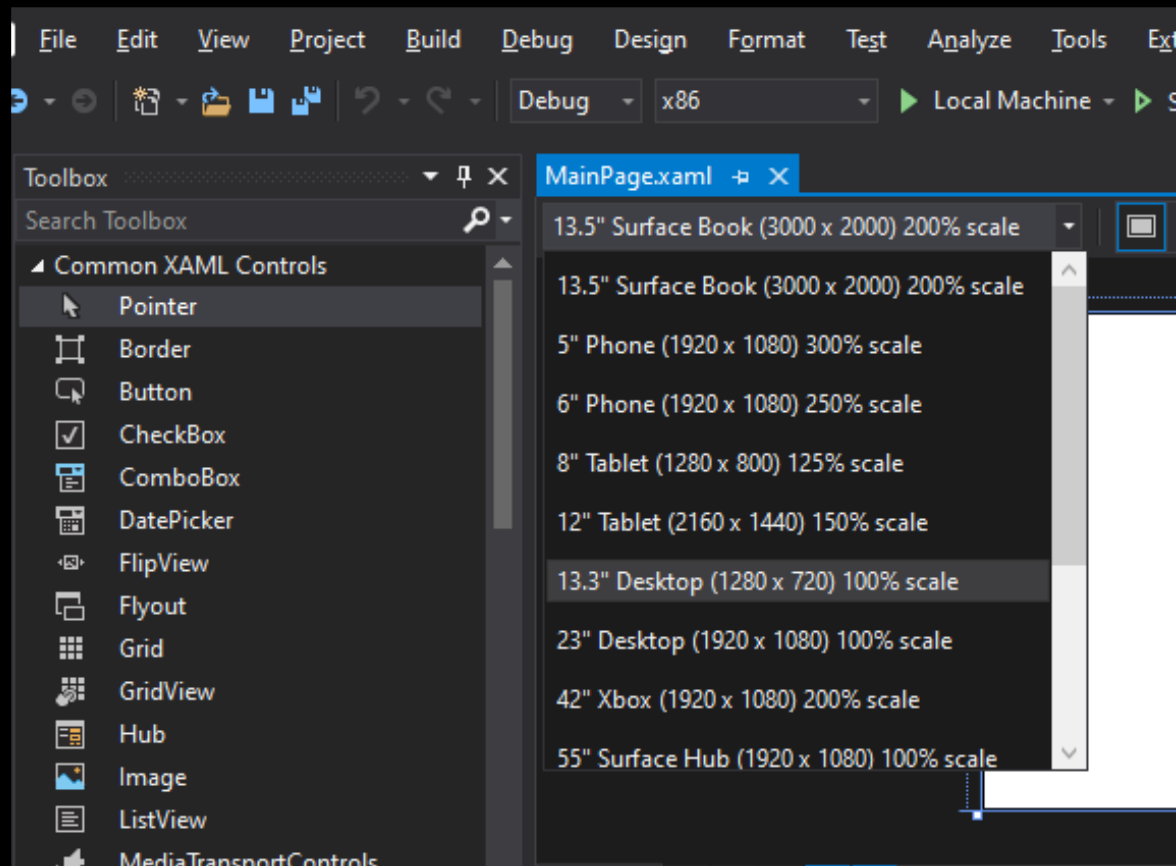


Creating a UWP Project

- In Solution Explorer, double-click on the **MainPage.xaml** file to open it for editing.
- You will see the XAML design window showing a graphical view and a XAML view.

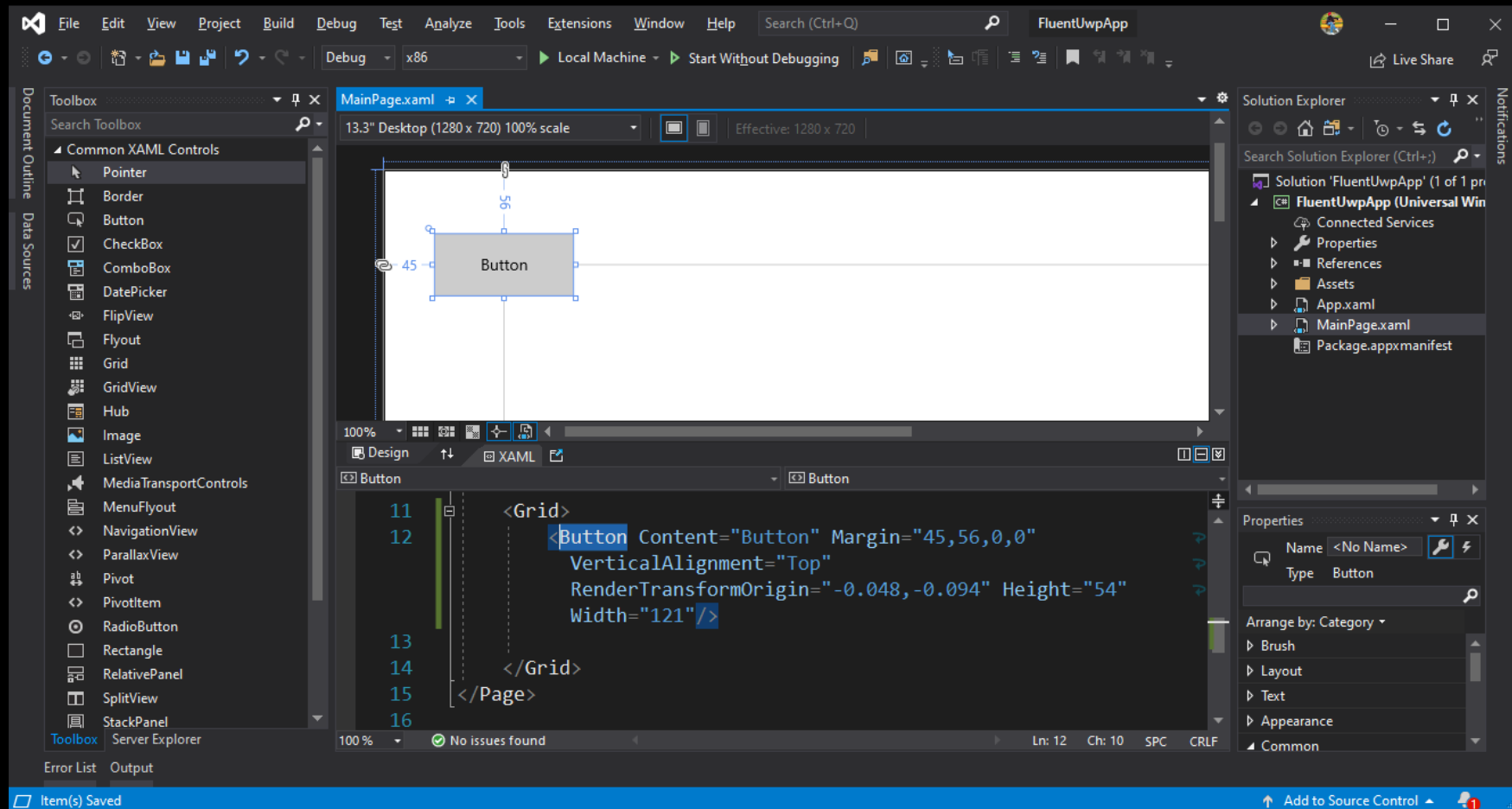
Creating a UWP Project

- For Design view, change the device to **13.3" Desktop (1280 x 720) 100% scale** and zoom to **100%**.



Creating a UWP Project

- Drag and drop the Button control from the toolbox onto the Design view.
- Resize it by clicking, holding, and dragging any of the eight square-shape resize handles on each edge and in each corner.



Creating a UWP Project

- Although you can drag and drop controls, it is better to use the XAML view for layout so that you can position items relatively and implement more of a responsive design.

```
<Grid>
    <Button Content="Button"
        Margin="30,40,0,0"
        VerticalAlignment="Top"
        Height="40" Width="150"/>
</Grid>
```

Creating a UWP Project

- In the XAML view, find the Button element and delete it.
- In the XAML view, inside the Grid element, enter the following markup:

```
<Button Content="Click Me"  
Margin="6"  
Padding="6"  
Name="ClickMeButton" />
```

- Change the zoom to 50% and note that the button is automatically sized to its content, Click Me, aligned vertically in the center and aligned horizontally to the left, even if you toggle between vertical and horizontal phone layouts.

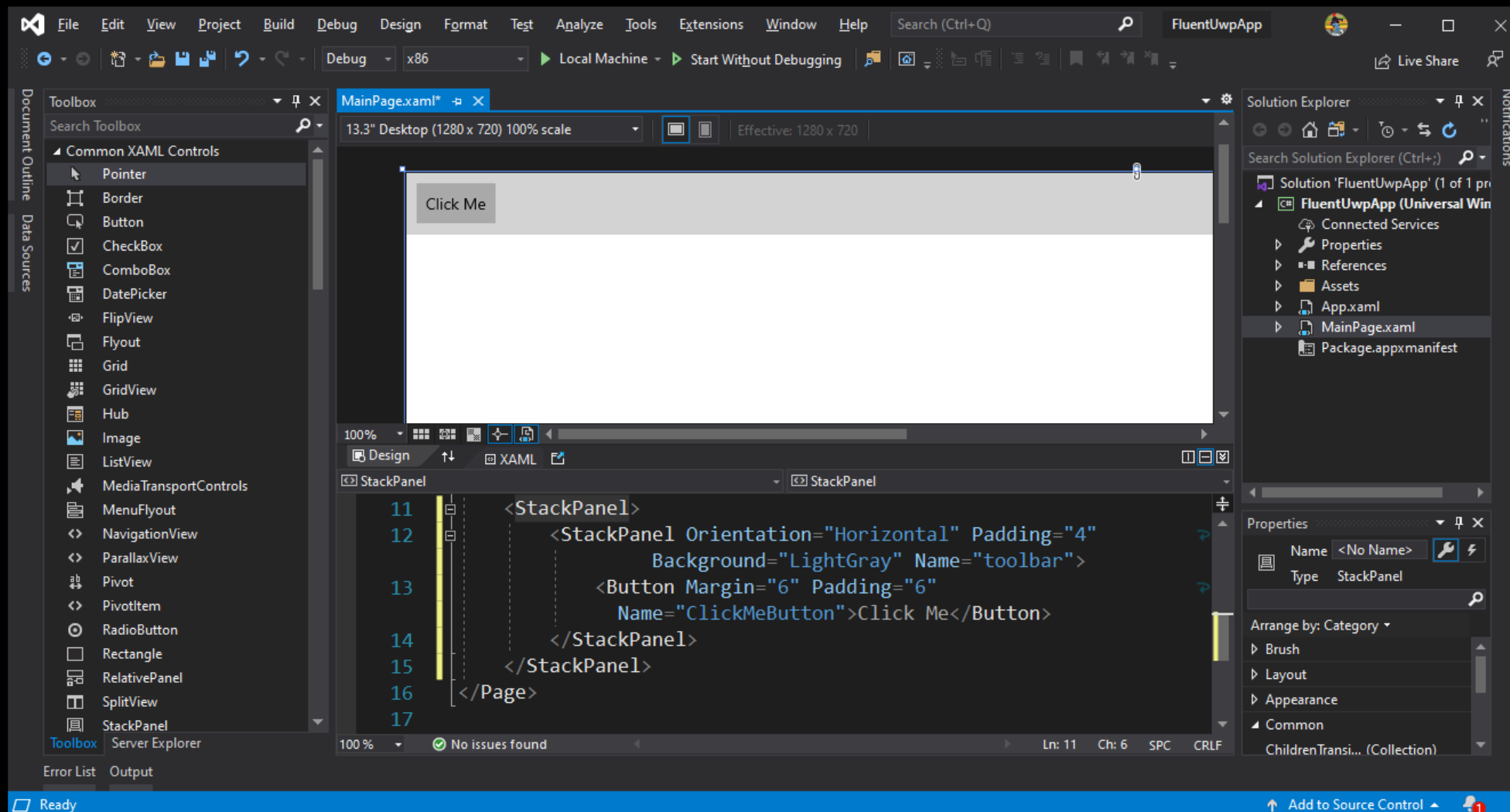
Creating a UWP Project

- In the XAML view, delete the `Grid` element, and modify the XAML to wrap the `Button` element inside a horizontally-orientated `StackPanel` with a light gray background that is inside a vertically orientated (by default) `StackPanel`, and note the change in its layout to be in the top-left of the available space, as shown in the following code:

```
<StackPanel>
    <StackPanel Orientation="Horizontal"
        Padding="4"
        Background="LightGray"
        Name="toolbar">
        <Button Margin="6"
            Padding="6"
            Name="ClickMeButton">Click Me</Button>
    </StackPanel>
</StackPanel>
```

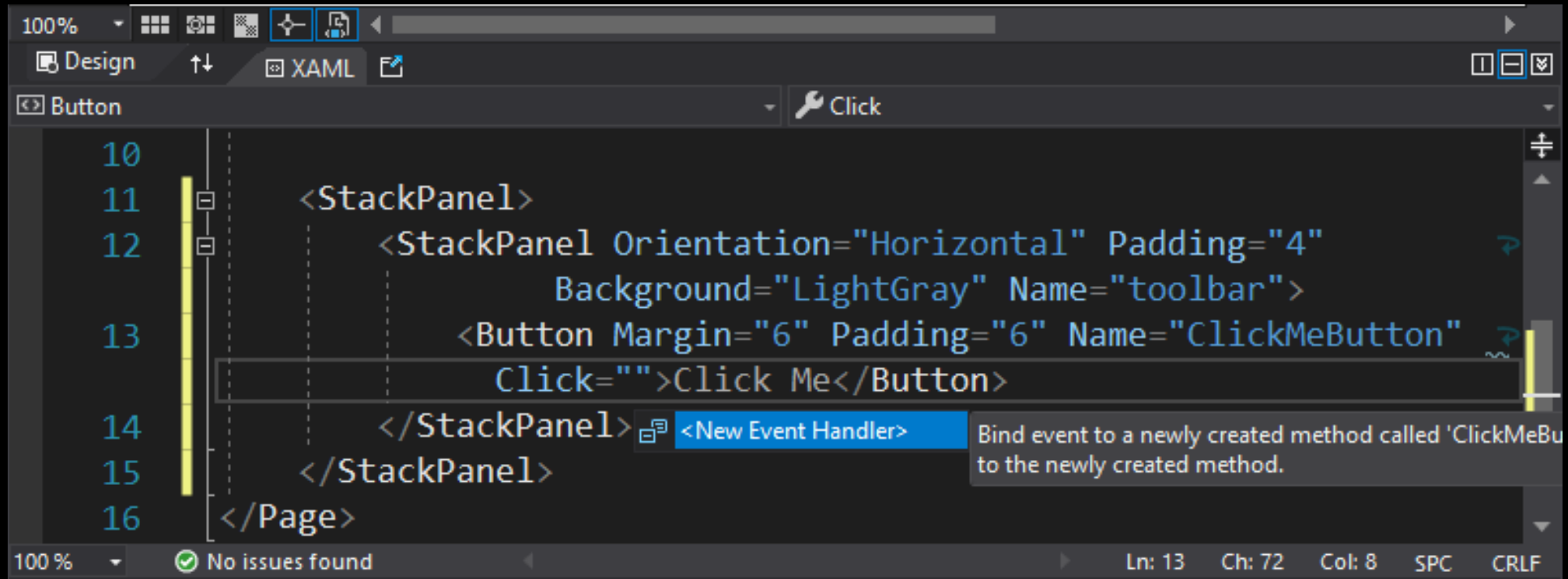
Creating a UWP Project

- This is how it'll look:



Creating a UWP Project

- Modify the Button element to give it a new event handler for its Click event.
- Press Enter to generate the event handler's name.



Creating a UWP Project

- Right-click the event handler name and select **Go To Definition** or press **F12**.
- Add a statement to the event handler method that sets the content of the button to the current time:

```
private void ClickMeButton_Click (object sender, RoutedEventArgs e)
{
    ClickMeButton.Content = DateTime.Now.ToString("hh:mm:ss");
}
```

- Run the application by navigating to **Debug | Start Without Debugging** or pressing **Ctrl + F5**.
- Click on the Click Me button and note that every time you click on the button, the button's content changes to show the current time.

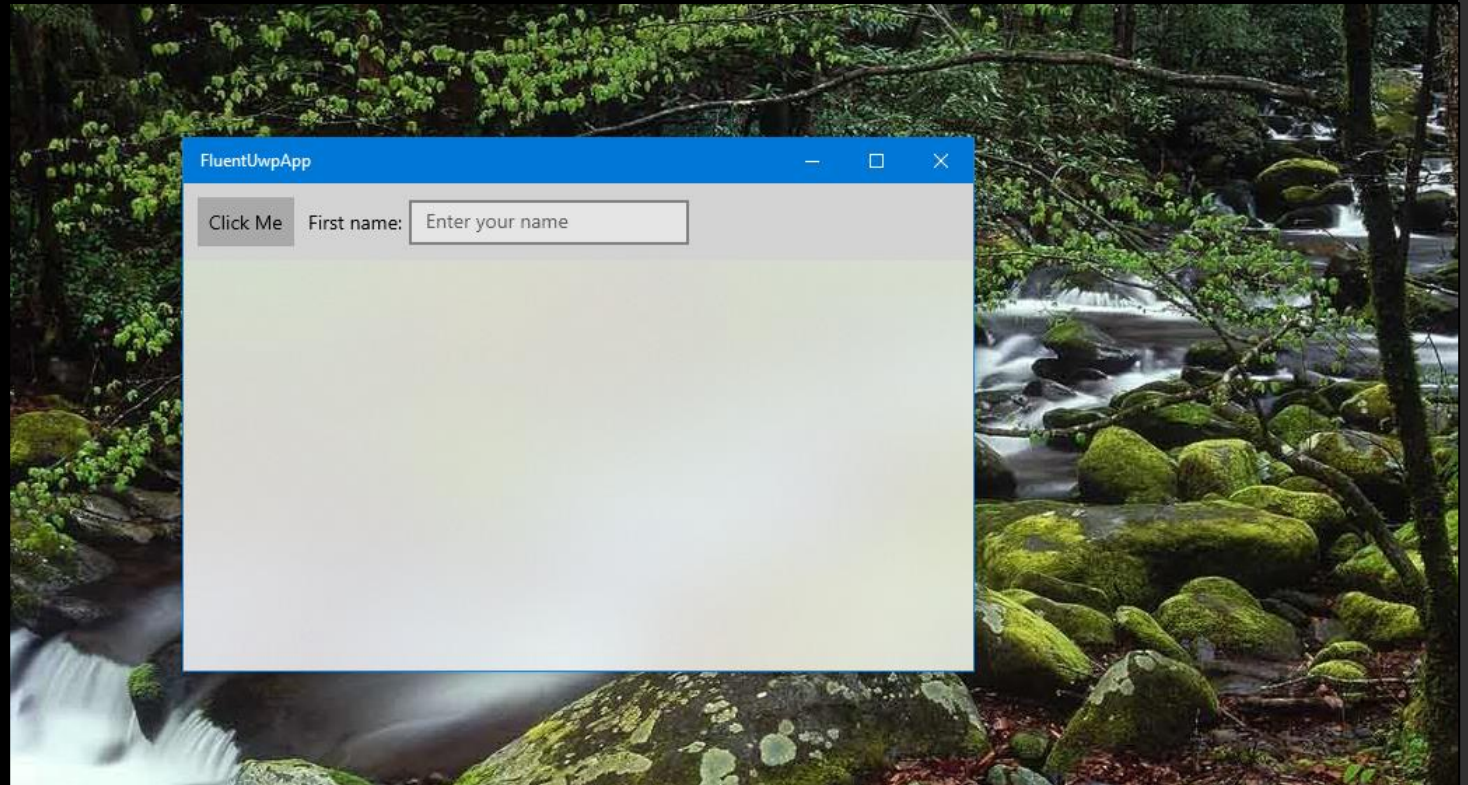
Exploring Common Controls and Acrylic Brushes

- Open **MainPage.xaml**, set the stack panel's background to use the acrylic system window brush.
- Add some elements to the stack panel after the button for the user to enter their name, as shown highlighted in the following markup:

```
<StackPanel Background="{ThemeResource SystemControlAcrylicWindowBrush}">  
    <StackPanel Orientation="Horizontal" Padding="4"  
        Background="LightGray" Name="toolbar">  
  
        <Button Margin="6" Padding="6" Name="ClickMeButton"  
            Click="ClickMeButton_Click">Click Me</Button>  
  
        <TextBlock Text="First name:" VerticalAlignment="Center" Margin="4" />  
  
        <TextBox PlaceholderText="Enter your name"  
            VerticalAlignment="Center" Width="200" />  
    </StackPanel>  
</StackPanel>
```

Exploring Common Controls and Acrylic Brushes

- Run the application by navigating to **Debug | Start Without Debugging**, and note the tinted acrylic material showing the desktop background through the app window background.
- **Note:** Acrylic uses a lot of system resources, so if an app loses the focus, or your device is low on battery, then acrylic is disabled automatically.



Exploring Reveal

- Reveal is built-in for some controls, such as `ListView` and `NavigationView`.
- For other controls, you can enable it by applying a theme style.
- First, we will add some XAML to define a calculator user interface made up of a grid of buttons.
- Then, we will add an event handler for the page's `Loaded` event so that we can apply the Reveal theme style and other properties by enumerating the buttons instead of having to manually set attributes for each one in XAML.

Exploring Reveal

- Open **MainPage.xaml**, add a new horizontal stack panel under the one used as a toolbar, and add a grid with buttons to define a calculator.
- Follow the example for complete code... (or refer to the lecture recording)

Exploring Reveal

- In the Page element, add a new event handler for Loaded, as shown highlighted in the following markup:

```
<Page  
  
...  
  
Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"  
Loaded="Page_Loaded">
```

- Right-click Page_Loaded and select **Go To Definition** or press **F12**.

Exploring Reveal

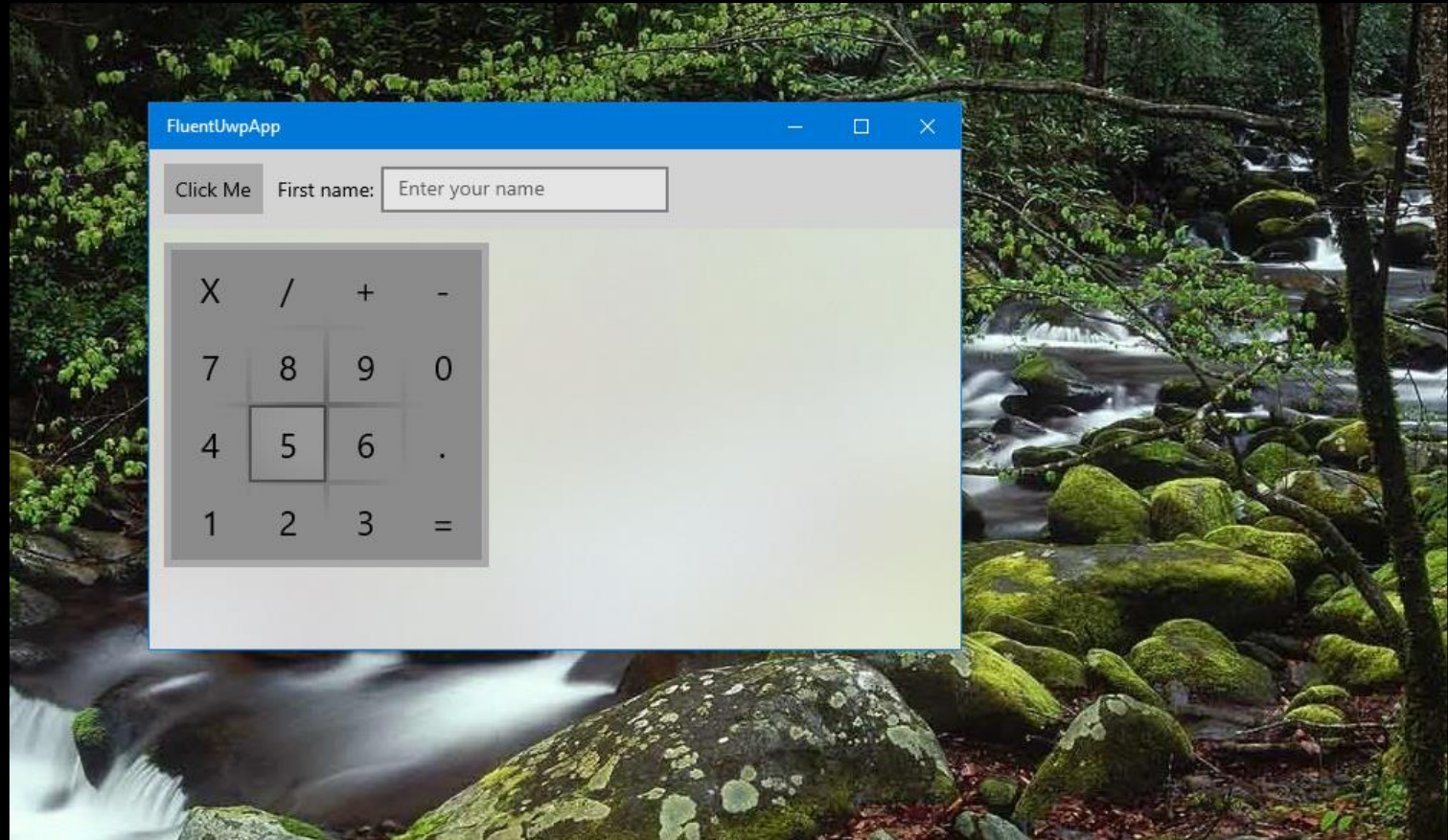
- Add statements to the Page_Loaded method to loop through all of the calculator buttons, setting them to be the same size, and apply the Reveal style.

```
private void Page_Loaded (object sender, RoutedEventArgs e)
{
    Style reveal = Resources.ThemeDictionaries["ButtonRevealStyle"] as Style;

    foreach (Button btn in gridCalculator.Children.OfType<Button> ())
    {
        btn.FontSize = 24;
        btn.Width = 54;
        btn.Height = 54;
        btn.Style = reveal;
    }
}
```


Exploring Reveal

- Run the application by navigating to **Debug | Start Without Debugging** and note the calculator buttons start with a flat gray user interface.
- When the user moves their mouse pointer over the bottom-right corner of the 8 button, we see that Reveal lights it up, and parts of the surrounding buttons light up too.



Using Resources and Templates

- When building graphical user interfaces, you will often want to use a resource, such as a brush to paint the background of controls or an instance of a class to perform custom conversions.
- These resources can be defined in a single place and shared throughout the app.

Sharing Resources

- A good place to define shared resources is at the app level.
- In Solution Explorer, open the **App.xaml** file.
- Add the following markup inside the existing Application element to define a linear gradient brush with a key of rainbow.

```
<Application
  x:Class="FluentUwpApp.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:FluentUwpApp">

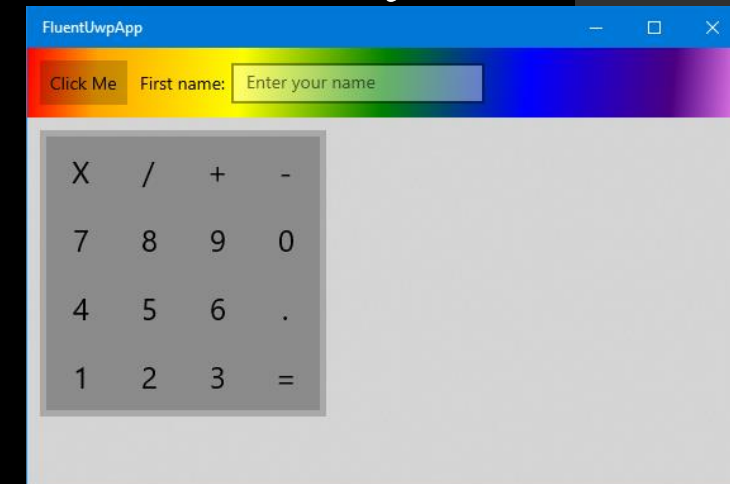
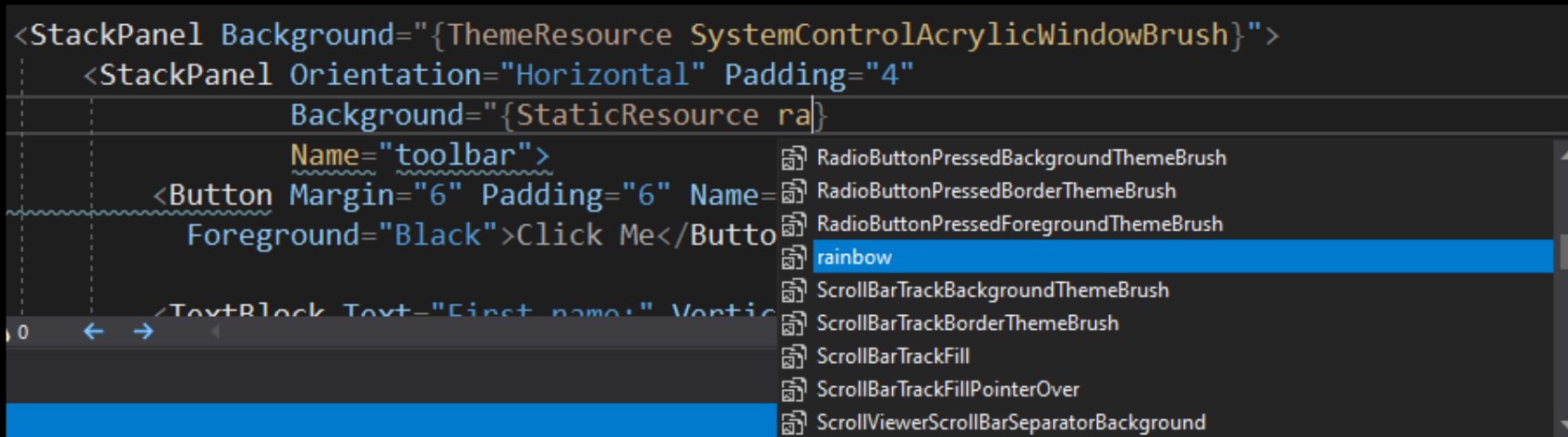
  <Application.Resources>
    <LinearGradientBrush x:Key="rainbow">
      <GradientStop Color="Red" Offset="0" />
      <GradientStop Color="Orange" Offset="0.1" />
      <GradientStop Color="Yellow" Offset="0.3" />
      <GradientStop Color="Green" Offset="0.5" />
      <GradientStop Color="Blue" Offset="0.7" />
      <GradientStop Color="Indigo" Offset="0.9" />
      <GradientStop Color="Violet" Offset="1" />
    </LinearGradientBrush>
  </Application.Resources>
</Application>
```

Using Resources and Templates

- Navigate to **Build | Build FluentUwpApp**.
- In **MainPage.xaml**, modify the StackPanel element named toolbar to change its background from LightGray to the static resource rainbow brush, as shown in the following markup:

```
<StackPanel Orientation="Horizontal" Padding="4"  
            Background="{StaticResource rainbow}" Name="toolbar">
```

- As you enter the reference to a static resource, IntelliSense will show your



Using Resources and Templates

- **Good Practice:**

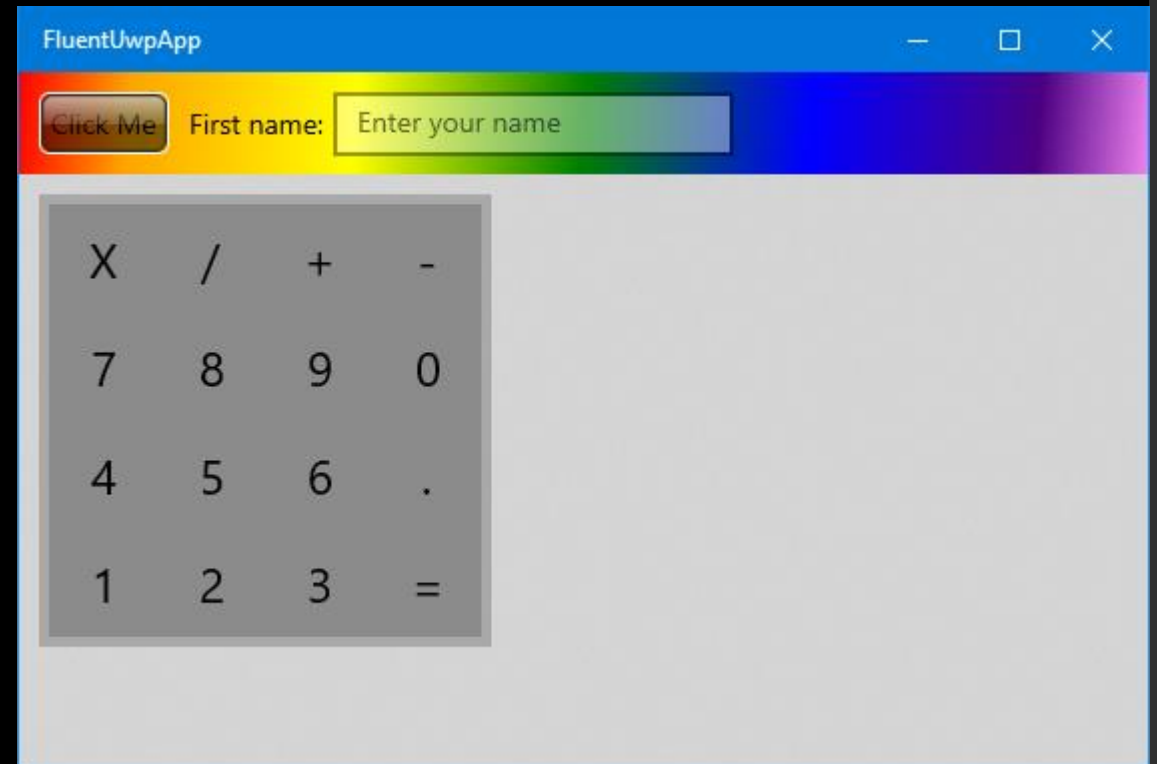
- A resource can be an instance of any object.
 - To share it within an application, define it in the **App.xaml** file and give it a unique key.
 - To set an element's property with a resource, use `{StaticResource key}`.
-
- Resources can be defined and stored inside any element of XAML, not just at the app level.
 - For example, if a resource is only needed on **MainPage**, then it can be defined there.
 - You can also dynamically load XAML files at runtime.
 - More Information: <https://docs.microsoft.com/en-ca/windows/uwp/app-resources/>

Replacing a Control Template

- You can redefine how a control looks by replacing its default template.
- The default control template for a button is flat and transparent.
- One of the most common resources is a style that can set multiple properties at once.
- If a style has a unique key then it must be explicitly set, like we did earlier with the linear gradient.
- If it doesn't have a key, then it will be automatically applied based on the `TargetType` property.

Replacing a Control Template

- In **App.xaml**, define a control template inside the `Application.Resources` element.
 - Note that the `Style` element will automatically set the `Template` property of all controls that are `TargetType` (buttons in this case) to use the defined control template.
 - The code is on the next slide.
-
- Run the application and note the black glass effect on the button in the toolbar.
 - The calculator buttons are not affected at runtime by this black glass effect because we replace their styles using code after the page has loaded.



Replacing a Control Template

```
<Application.Resources>
  <LinearGradientBrush x:Key="rainbow">
    ...
  </LinearGradientBrush>
  <ControlTemplate x:Key="DarkGlassButton" TargetType="Button">
    <Border BorderBrush="#FFFFFF" BorderThickness="1,1,1,1" CornerRadius="4,4,4,4">
      <Border x:Name="border" Background="#7F000000" BorderBrush="#FF000000" BorderThickness="1,1,1,1" CornerRadius="4,4,4,4">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition Height="*" /> <RowDefinition Height="*" />
          </Grid.RowDefinitions>
          <Border Opacity="0" HorizontalAlignment="Stretch" x:Name="glow" Width="Auto" Grid.RowSpan="2" CornerRadius="4,4,4,4">
          </Border>
          <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" Width="Auto" Grid.RowSpan="2" Padding="4" />
          <Border HorizontalAlignment="Stretch" Margin="0,0,0,0" x:Name="shine" Width="Auto" CornerRadius="4,4,0,0">
            <Border.Background>
              <LinearGradientBrush EndPoint="0.5,0.9" StartPoint="0.5,0.03">
                <GradientStop Color="#99FFFFFF" Offset="0" /> <GradientStop Color="#33FFFFFF" Offset="1" />
              </LinearGradientBrush>
            </Border.Background>
          </Border>
        </Grid>
      </Border>
    </Border>
  </ControlTemplate>
  <Style TargetType="Button">
    <Setter Property="Template" Value="{StaticResource DarkGlassButton}" />
    <Setter Property="Foreground" Value="White" />
  </Style>
</Application.Resources>
```



Thank You