UWP-002 - Creating your First Universal Windows Platform App

Near the end of the C# Fundamentals for Absolute Beginners series on Microsoft Virtual Academy I demonstrated how Events work in C#.

And so, to demonstrate that, I created two very different types of applications, but I followed the same workflow and got a similar result. First, I created a ASP.NET Web Forms application, and then second I created a Windows Presentation Foundation (WPF) application. I took the same basic steps to create those two applications, and I got essentially the same result, even though one result was presented in a web browser and the other was presented in a Windows form. I placed a button on the form, the end user clicks the button which was handled by the Click event, and the code programmatically changed the value of Label control to the words "Hello World".

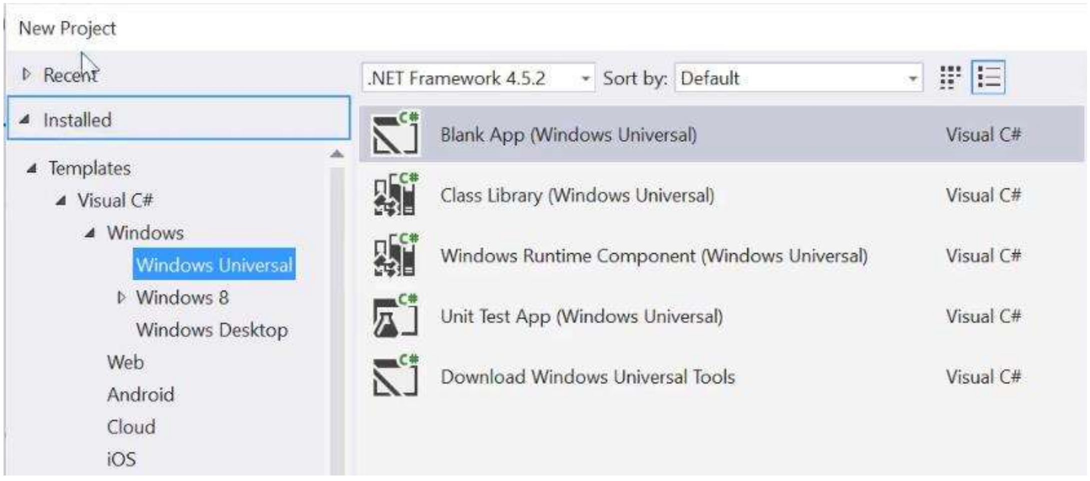
That example illustrated how C# Events work, and also to give us confidence we could leverage the same techniques and workflow to build another type of application.

The good news is that we can re-purpose that knowledge to create Universal Windows Platform apps. In this lesson I want to re-create that same application a basic "Hello World" application, but this time, I'll do it creating a simple, Universal Windows application and I encourage you to follow along.

Before getting started please make sure that you already have Visual Studio 2015 installed. Any edition will do: Express, Community, or one of the paid editions. Secondly, you're going to need to make sure that you have the Windows 10 Emulators installed, just like we talked about in the previous lesson.

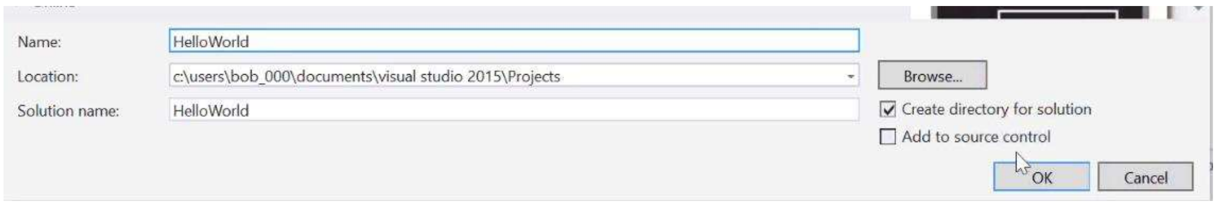
Assuming that you've got all of that installed and we're ready to go, we will get started by creating a new project. There are many ways in Visual Studio to do this. But create a new project by clicking on the New Project link in the Start page. That will open up the New Project dialog in Visual Studio.

On the left-hand side, select: Installed templates > Visual C# > Windows > Windows Universal, and then in the center a series of templates are displayed. Choose the top option, “Blank App (Windows Universal)” template.



Blank App (Windows Universal)	Visual C#
Class Library (Windows Universal)	Visual C#
Windows Runtime Component (Windows Universal)	Visual C#
Unit Test App (Windows Universal)	Visual C#
Download Windows Universal Tools	Visual C#

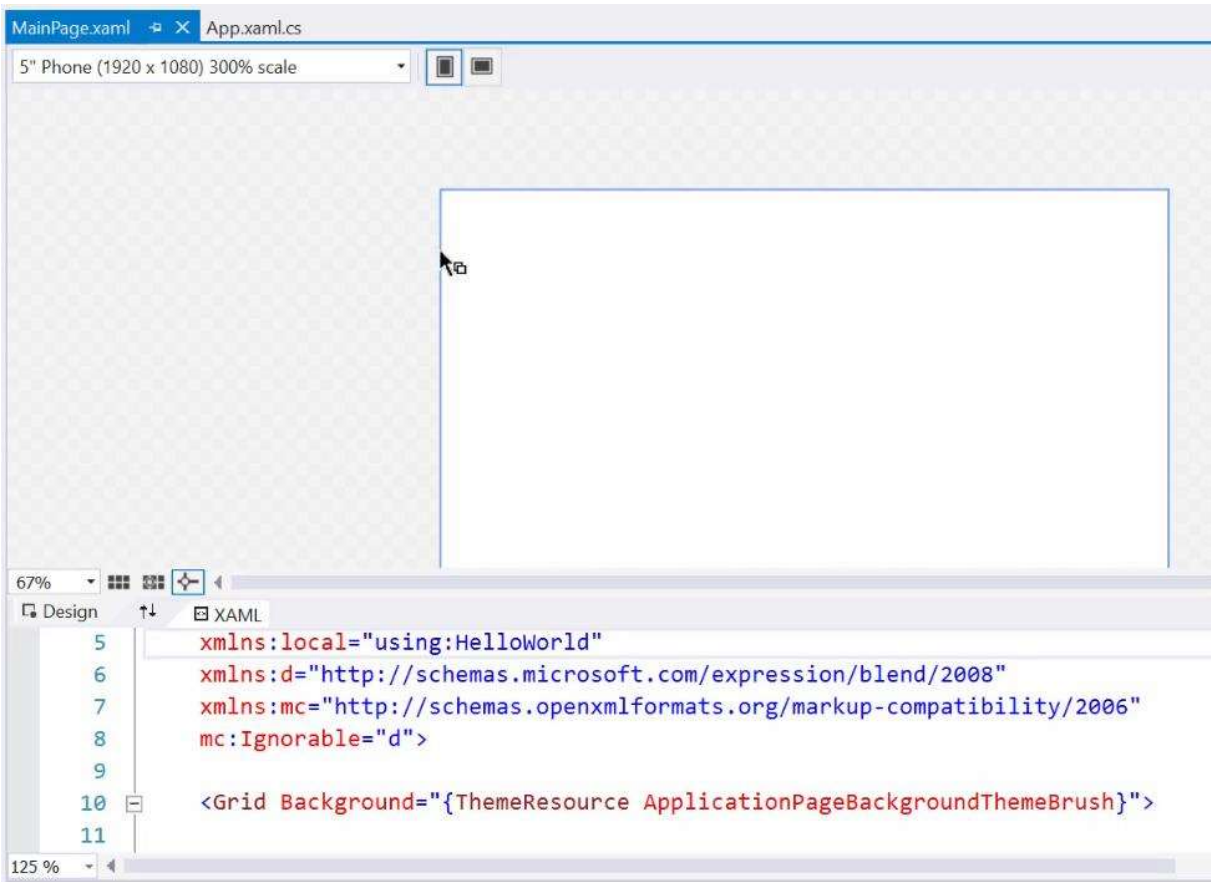
Next, change the project name from whatever the default is to simply "HelloWorld", and click the OK button in the lower right-hand corner.



Name: HelloWorld
Location: c:\users\bob_000\documents\visual studio 2015\Projects
Solution name: HelloWorld
☑ Create directory for solution
☐ Add to source control

It may take Visual Studio a few moments to set up the new project.

First, in the Solution Explorer window, double click the MainPage.xaml file. That will open up that file in a special type of designer that has two panes. The top pane is a visual designer, and the lower pane is a textual code-based designer. This designer allows us to edit the code that will be used to generate objects on screen. The lower pane displays the XAML that correlates to the upper pane. So I'll refer to this as the “XAML Editor”.

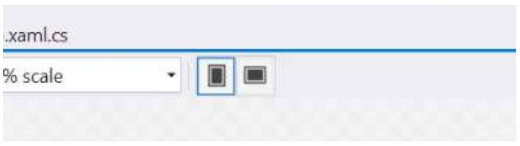


```
    xmlns:local="using:HelloWorld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

```

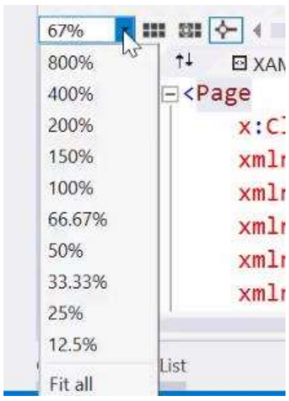
In the top pane, you can see that we get a visual representation of our application. And you can see that we're actually looking at a a rendering of how it would look if we were to design our application or run our application on a five-inch phone screen with a resolution of 1920 by 1080.

The visual designer is displaying at a 300% scale. This may be too large for our purposes and we will change that in just a moment.

Notice that we can also view what our application's user interface would look like in portrait or in landscape mode by toggling the little buttons at the top.

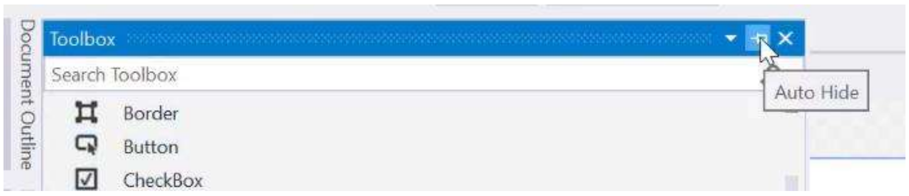


But again, this is a little bit large, we have to scroll around on the screen just to see the entire design surface. So we can make it a little bit smaller by going to the zoom window in the lower left-hand corner. And I'm just going to choose like 33%.

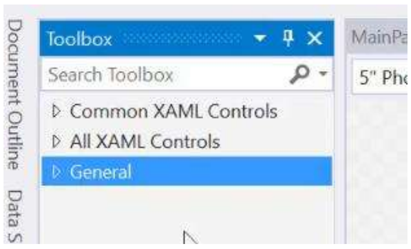


Now that won't completely compact it down for our viewing pleasure. However, it makes it much smaller and manageable.

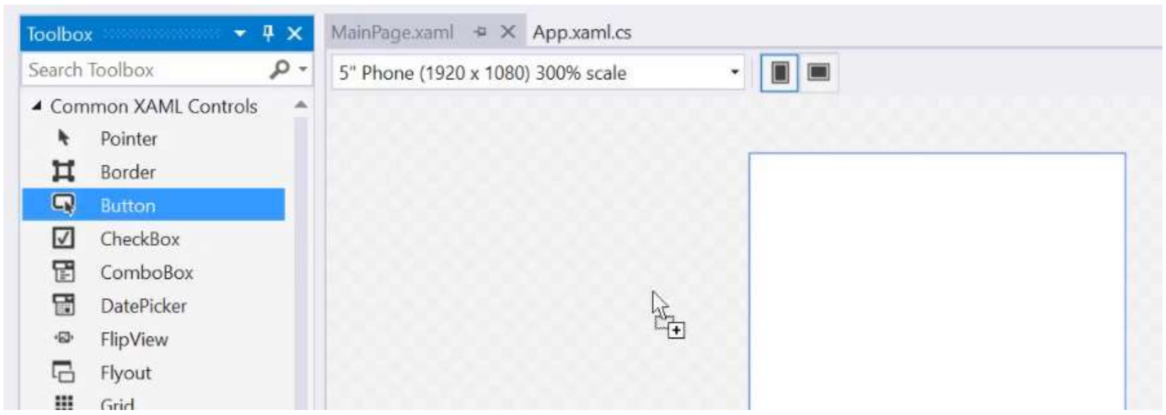
What we will do is start off by adding some controls onto this designer surface, just to break into exactly how this all works. So, over on the left-hand side, there should be a little tab sticking out called Toolbox. And if you click it, the Toolbox window will jettison out from the right-hand side. I'm actually going to use this little pin in the upper right-hand corner of that window. And I click it once, and that will turn off auto-hide. So now that Toolbox window is automatically docked over here on the left-hand side. And if you don't like the way that it's positioned, you can just, again, position it any way you want to using your mouse and the little border area between the Toolbox and the main area.



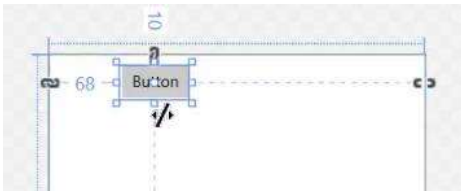
And you'll notice that if we were to roll up here to the very top, that there are a couple of categories of controls that can be added, common XAML controls, or all XAML controls. We'll discuss many of these throughout this series. We basically want to work with just some common ones to start off with.



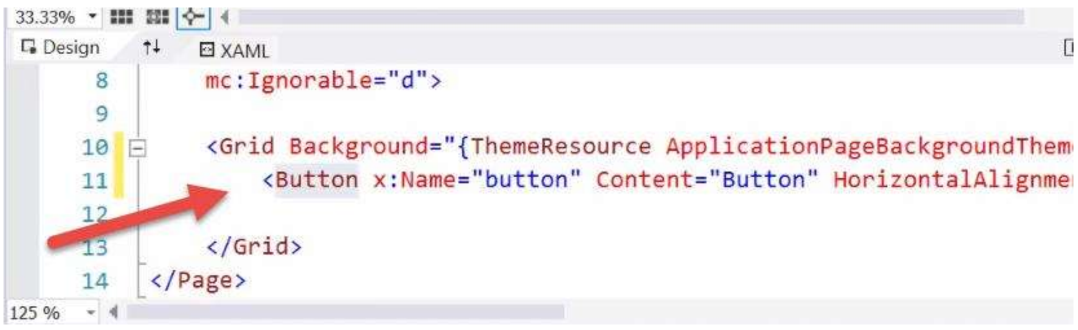
So drag and drop a Button control from the Toolbox, onto the design surface.



So here we go, dragging and dropping. And notice when I do, it drops it right where my mouse cursor was.



And, hopefully you notice also, here, in this XAML editor that it added this line of code in line number 11. It created a tag, which is an opening and closing angle bracket.

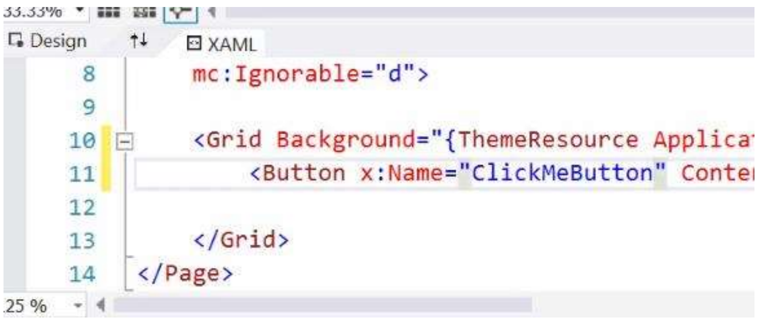


```
8        mc:Ignorable="d">
9
10       <Grid Background="{ThemeResource ApplicationPageBackgroundThem
11           <Button x:Name="button" Content="Button" HorizontalAlignme
12
13       </Grid>
14   </Page>
```

Very similar to what you would see in HTML, and yet this is not HTML. But similar in the sense that you use HTML to lay out the design of a web page. Same is true with XAML.   use the XAML programming language to easily lay out the various controls and the layout for our application.

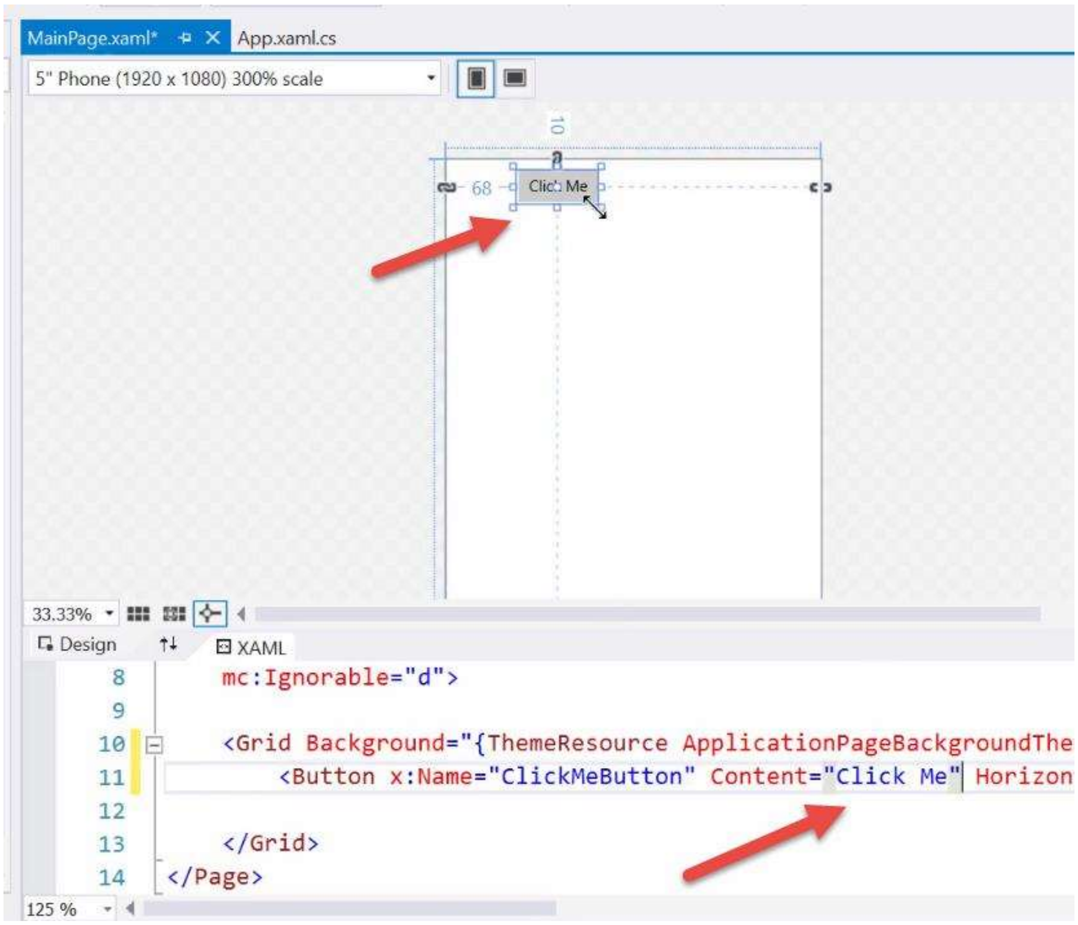
Notice that we created a button control and it has a number of different properties like the name of that control, the content of that control. Also the alignment, the horizontal alignment and the vertical alignment, and then margin. And that margin, as you can see, is based on, here, this little visual view of it. It's 10 pixels from the top and 68 pixels from the left. And so, that's where the alignments come into play, the horizontal alignment left, that's what we're aligning to and then 10 pixels from the top. And then, for the right-hand side and the bottom, well, those are set to zero because we're really not worried about those.

However, just notice that there are attributes or properties of the elements that we add in our XAML editor. So now if I wanted to change the name of this button for programmatic access in C#, I can just call this the ClickMeButton.



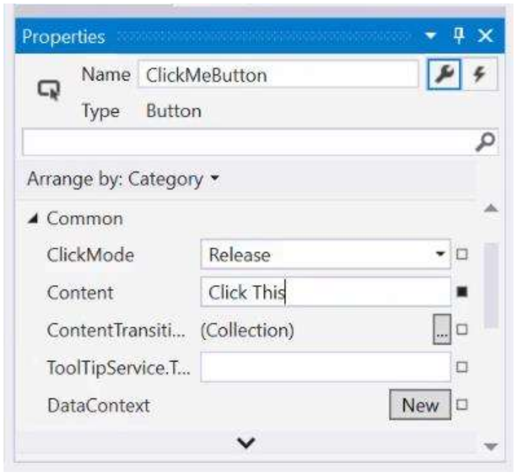
```
    mc:Ignorable="d">

    <Grid Background="{ThemeResource Applica
        <Button x:Name="ClickMeButton" Conte

    </Grid>
</Page>
```

And I can also change the content here as well, by typing in Click Me to the Content area. Notice that when I do that, it changes the visual attribute of the button to "Click Me" instead of whatever was in there before.



```
        mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThe
        <Button x:Name="ClickMeButton" Content="Click Me" Horizon

    </Grid>
</Page>
```

Now there are also a third way that I can make modifications to this object, to this button object. We see it here in the designer. We see it here in the XAML editor. But then also here in the right-hand side of the Properties window, we can make changes by finding a property that modify like, for example, the text property or some common properties like the content. And we can change it by just modifying the little Property editor and hitting enter on our keyboard.

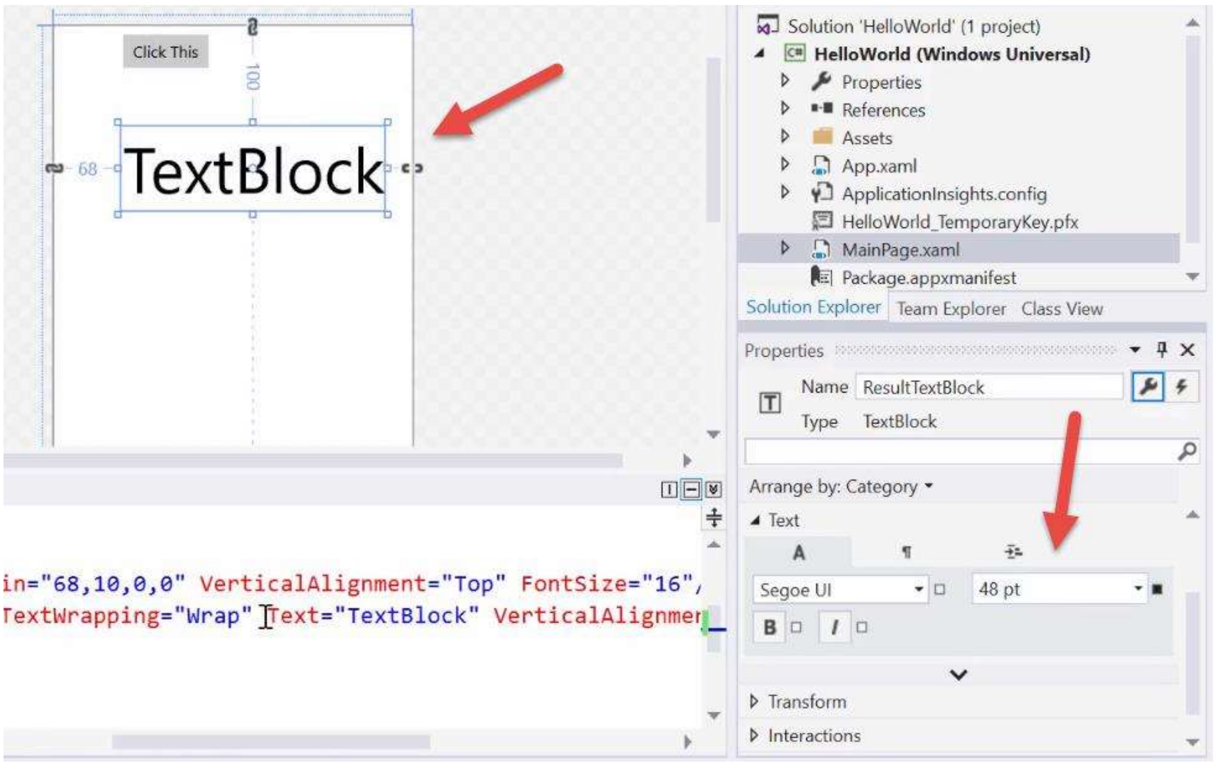


Name	ClickMeButton
Type	Button
ClickMode	Release
Content	Click This
ContentTransiti...	(Collection)
ToolTipService.T...	
DataContext	New

Notice when I did that, it modified not only what we see here in the designer, but then also what we see here in the XAML editor. Change the content property to the value "Click This".

Notice that these are essentially three different views of the same object. One is visual, one is textual, and the other is a Property editor. But we can make changes in any of these and they'll show up in the other one, so they're all connected together.

Next I'll drag and drop a TextBlock control from the Toolbox to the visual Designer surface. The XAML code is also displayed in the XAML editor. I'll change the name to "ResultTextBlock" and change the text property to 48 point.

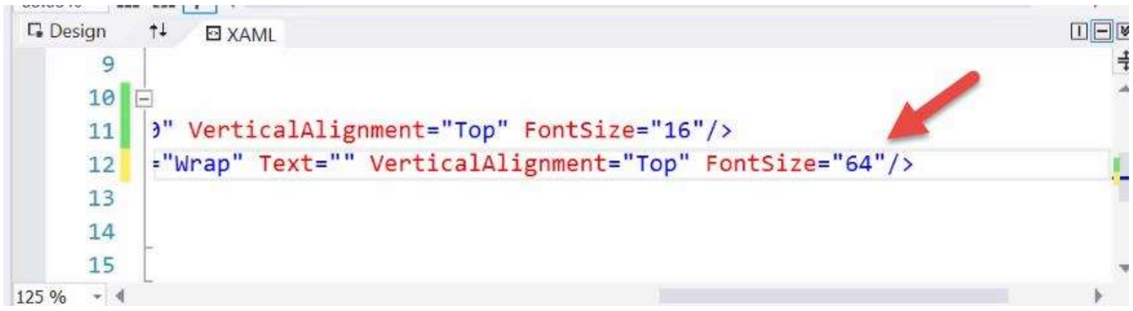


Name ResultTextBlock
Type TextBlock
Segoe UI
48 pt

```
in="68,10,0,0" VerticalAlignment="Top" FontSize="16",
TextWrapping="Wrap" Text="TextBlock" VerticalAlignmen
```

A TextBlock is a label: somewhere where we can display text.

When reviewing the XAML notice that the font size is set to 64.

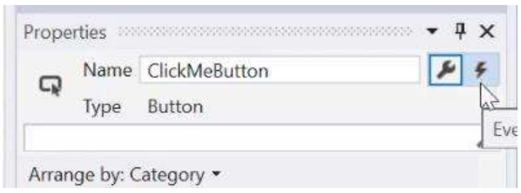


```
)" VerticalAlignment="Top" FontSize="16"/>
="Wrap" Text="" VerticalAlignment="Top" FontSize="64"/>
```

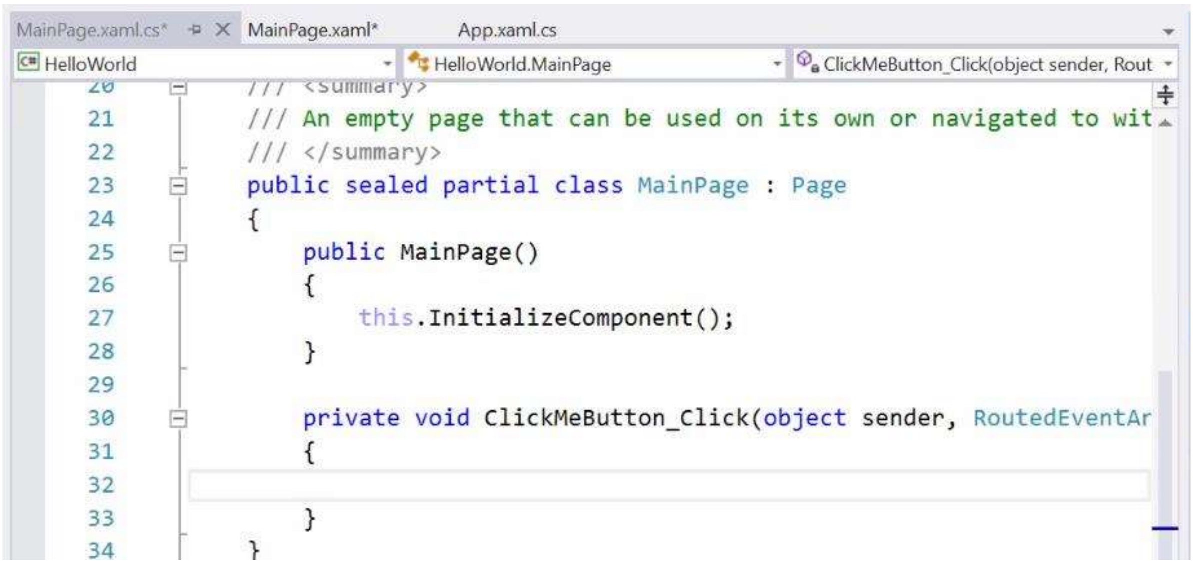
And you might say, "Wait a second here. I see that it's 48 point over here in the Properties editor, but it's 64 units here in the Font Size attribute in the XAML editor. I thought those were supposed to be the same."

Well actually, they are the same. The XAML editor is rendering values in device independent pixels, or rather DIPs. The Properties window is actually representing values in points, which are a fixed size of 1/72nd of an inch. So, it's a very complicated topic, but we will revisit this when we take into consideration the various screen sizes and resolutions on the various form factors that Windows 10 will run on as we're building our applications. As you k you can have a large screen with a low resolution, or vice versa. And so, units of measurement like inches are irrelevant. Instead we need a new unit of measurement that accounts for both screen size and resolution to determine the sizes of things like controls that we add to our apps.

Next I'll select the Button control and then, in the Properties window, I'll select the little "bolt of lightning" button to display events that can be handled for the button.

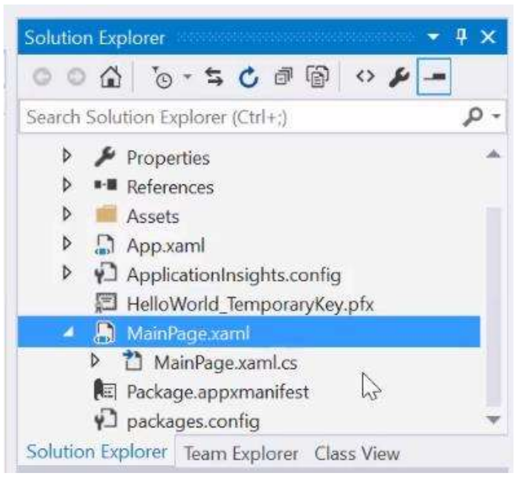


The very first event is the Click event. I'll double-click inside of that TextBox in the Property editor and open up the MainPage.xaml.cs file.

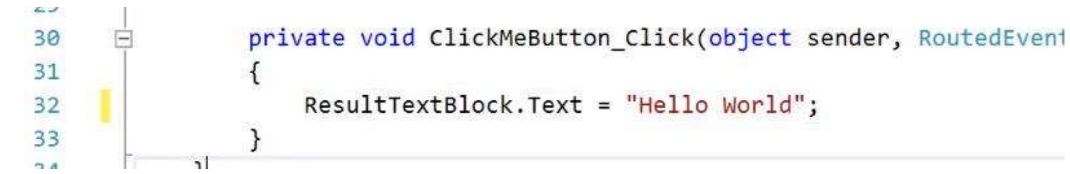


```
/// <summary>
/// An empty page that can be used on its own or navigated to wit
/// </summary>
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
    }

    private void ClickMeButton_Click(object sender, RoutedEventAr
    {

    }
}
```

Now notice here in our Solution Explorer, that the MainPage.xaml is related to the MainPage.xaml.cs file. They're actually two parts of the same puzzle. Two pieces of the same puzzle.   talk about that more later.

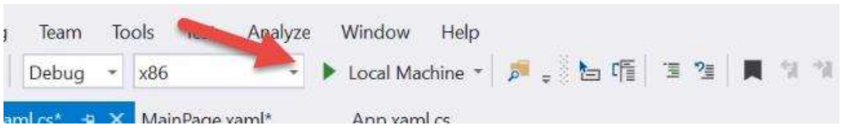


I'll write code that will display "Hello World" in the ResultTextBlock when a user clicks the ClickMeButton.

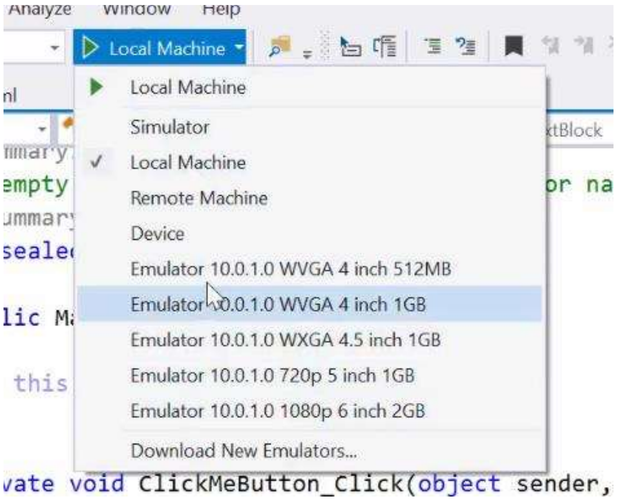


```
        private void ClickMeButton_Click(object sender, RoutedEvent
        {
            ResultTextBlock.Text = "Hello World";
        }
```

Next I'll test the application by clicking the Run Debug icon in the toolbar at the top.

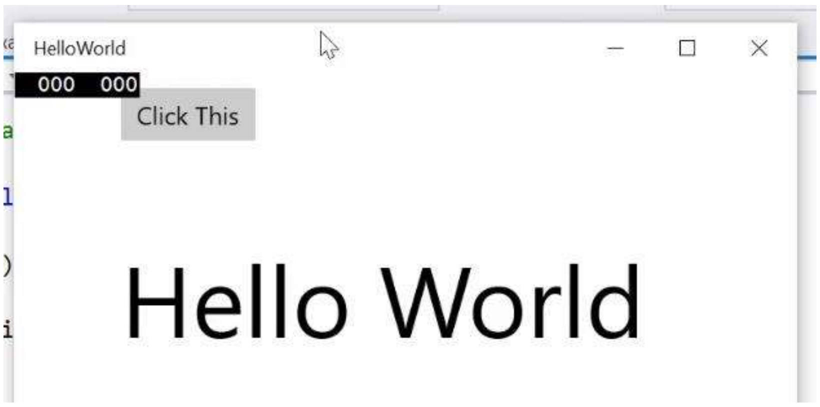


Notice that be running it on my local machine, but if I use this little drop-down arrow, you can see that there are some other options as well, including a Simulator, and an Emulator, as well as an actual physical device.


Analyze
Window
Help
Local Machine
Local Machine
Simulator
✓ Local Machine
Remote Machine
Device
Emulator 10.0.1.0 WVGA 4 inch 512MB
Emulator 10.0.1.0 WVGA 4 inch 1GB
Emulator 10.0.1.0 WXGA 4.5 inch 1GB
Emulator 10.0.1.0 720p 5 inch 1GB
Emulator 10.0.1.0 1080p 6 inch 2GB
Download New Emulators...

```
vate void ClickMeButton_Click(object sender,
```

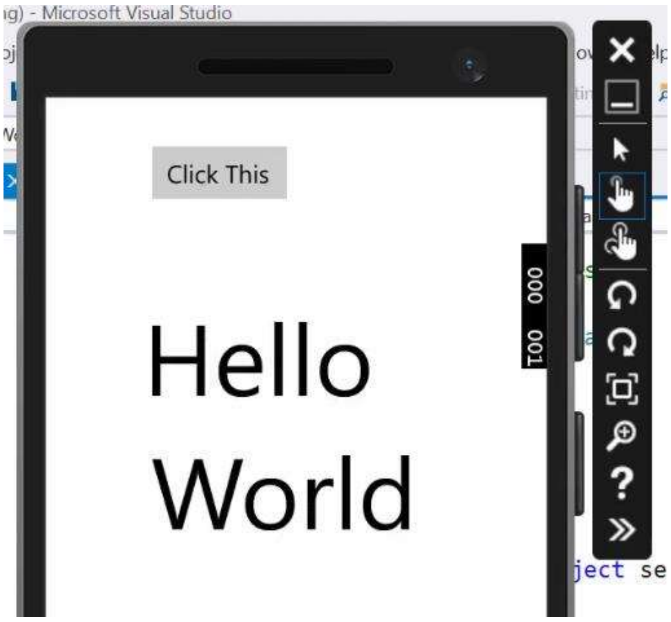
So the first time that we run this, actually just run it on the local machine. And it opens up our application, and it's actually quite large, but resize this down. And if we click the Click This button you can see the text "Hello World" appears, as we would expect.



And so, working in debug mode is great because we're able to set break points like we learned about in the C# Fundamentals for Absolute Beginners series, and inspect values as the application is currently running. We're able to also see our app, our Universal Windows Platform app running without having to deploy our app to an actual, physical Windows device. It's just running on our desktop.

Now for most of this course, be running the app on my local machine, like we've done here. Because really it's the fastest way to test what we've done. However, at times want to test on emulators. And so, an emulator is a virtual machine that emulates, or mimics, a physical device. And that physical device could be a phone or a tablet, or even an Xbox One, although we don't have that emulator just yet to work with.

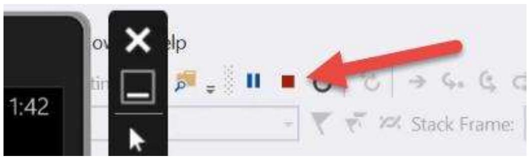
I'll run the application again, but this time choose this Emulator 10.0.1.0 WVGA. That option specifies which physical phone that we will be emulating. So, if we take a look, we can see that our application has been loaded up in this emulator that looks just like a phone.



To stop debugging using the emulator, I could click the close button in the upper right-hand corner of the emulator.

However, you should get in the habit of just leaving the emulator up and running, even when you're not debugging. There are a couple of reasons for this. First of all, the emulator is running as a virtual machine and it takes time to essentially reboot. To shut it down and then to turn it back on. And then, secondly, any app data that you save to the phone between debugging sessions will be lost whenever you shut down the simulator. So if you have data that you want to keep around between debugging sessions, do not close down the emulator.

The other option you have is to actually click the Stop Debugging button in the Toolbar.



Stop Debugging will allow the Emulator to continue running for the next debugging session.

The purpose of this lesson was to explain the relationship between the XAML editor, the design view and the Properties window. Second, the purpose was to demonstrate how C# in the code behind file can handle events and set attributes of those visual objects. Finally, how to debug the application as a desktop application or as a mobile application using the Emulators.