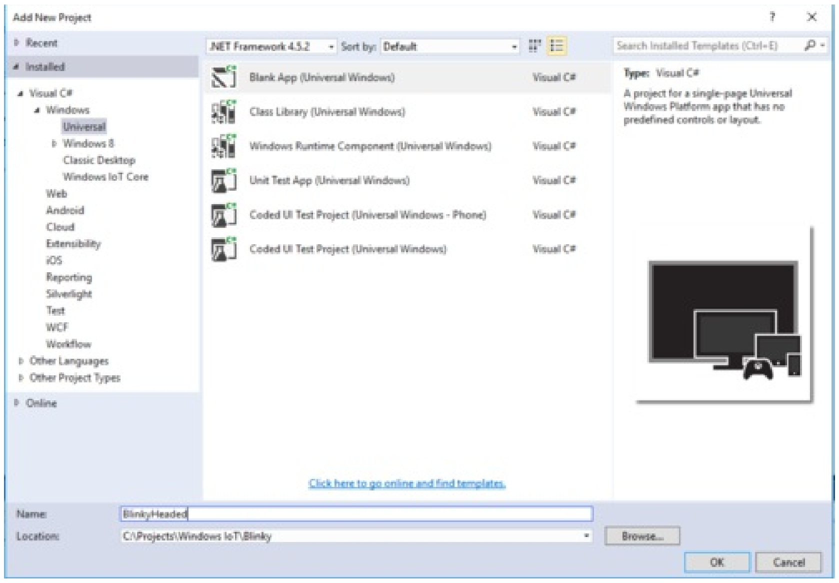
Windows IoT Core Application Development: Headed Blinky

Created by Rick Lesniak


Add New Project
Recent
Installed
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Windows IoT Core
Web
Android
Cloud
Extensibility
iOS
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Online
Blank App (Universal Windows) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Coded UI Test Project (Universal Windows - Phone) Visual C#
Coded UI Test Project (Universal Windows) Visual C#
Search Installed Templates (Ctrl+E)
Click here to go online and find templates.
Browse...
OK
Cancel

.NET Framework 4.5.2 Sort by: Default

Type: Visual C#

A project for a single-page Universal Windows Platform app that has no predefined controls or layout.

Name: BlinkyHeaded

Location: C:\Projects\Windows IoT\Blinky

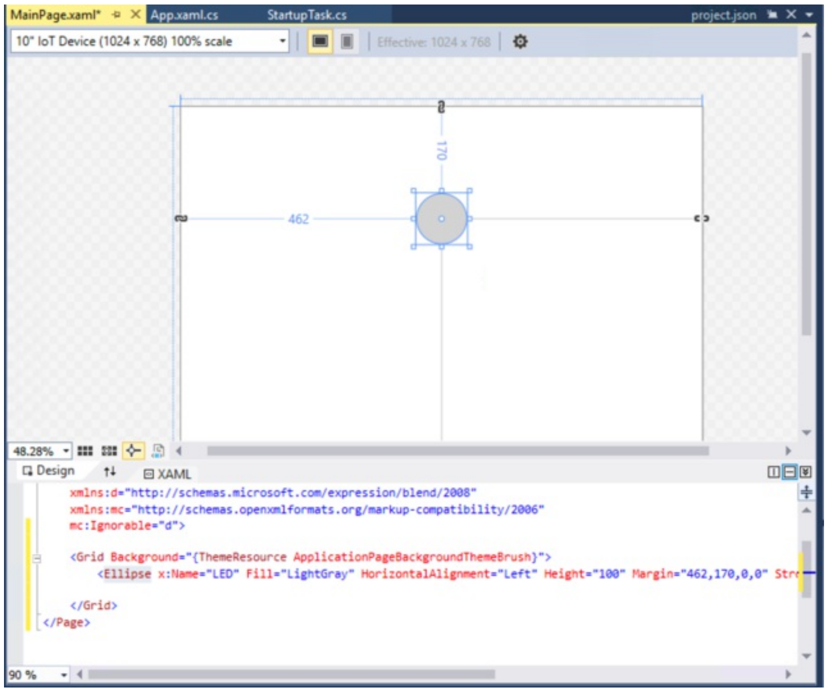
Last updated on 2016-11-02 05:29:27 PM UTC

Guide Contents

Introduction

We'll go through a simple *Headed* application. As mentioned in the previous tutorial, a Headed application differs from a Headless application in that it has a graphical user interface.

On a Pi, the user interface will appear on your HDMI monitor (currently, the Adafruit PiTFT is not supported by Windows 10 IoT Core).



```
    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}">
        <Ellipse x:Name="LED" Fill="LightGray" HorizontalAlignment="Left" Height="100" Margin="462,170,0,0" Str

    </Grid>
</Page>
```

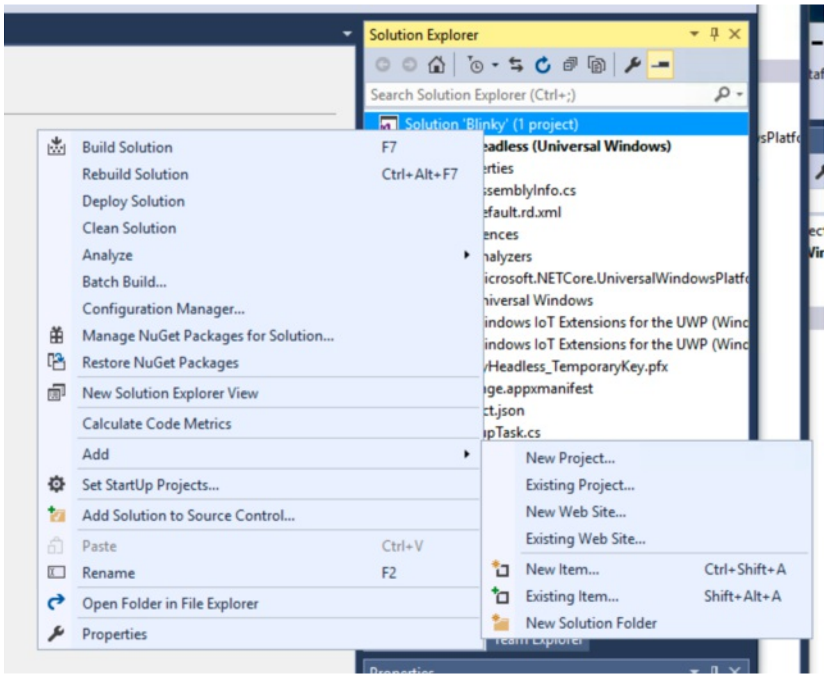
IMPORTANT: Before going forward, make sure you have successfully completed the BlinkyHeadless app. This tutorial assumes you will be building on that.
Go back to Headless Application tutorial
http://adafru.it/pcs

Add BlinkyHeaded project to Solution

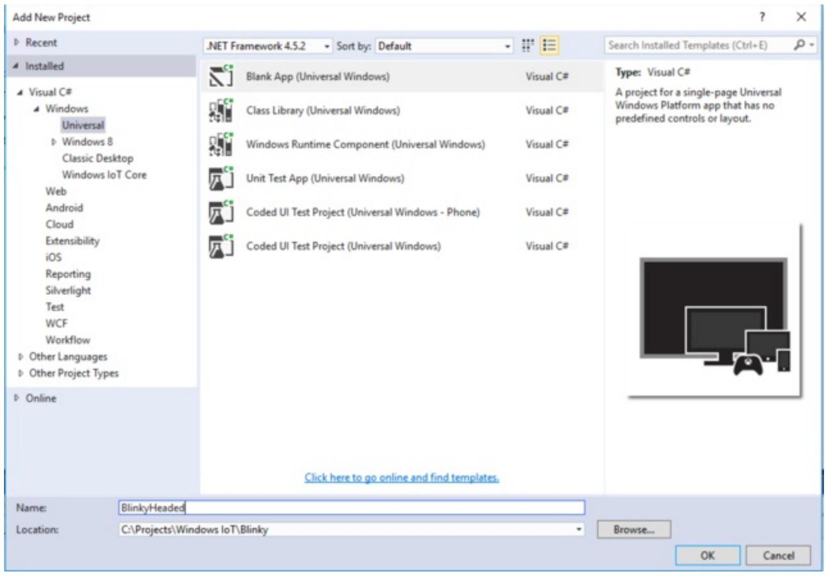
As mentioned in the Blinky Headless tutorial, you can have multiple projects inside a single solution. We will be adding the BlinkyHeaded project to the **Blinky** solution, alongside the BlinkyHeadless project.

To get started, follow thse steps:

- In the **Solution Explorer**, right-click on **Solution Blinky,** and place the cursor over **Add**. This will bring up another popup menu. Select **New Project**



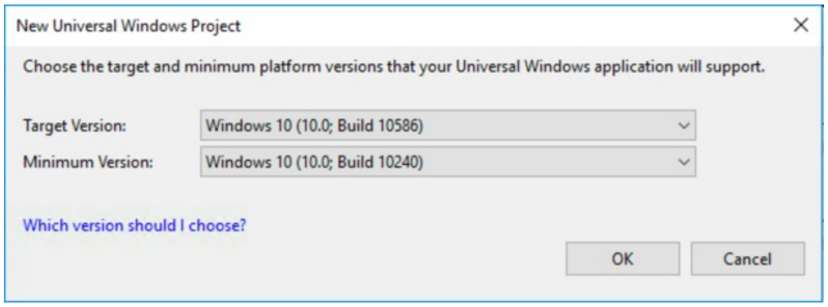
- The **Add New Project** window will open. On the left, navigate to **Installed > Visual C# > Windows > Universal**, and select **Blank App (Universal Windows)**
- Give the project a name ("BlinkyHeaded").
- Leave the Location path alone. Visual Studio will automatically put the project inside the **Blinky Solution** folder.
- Click **OK**


Add New Project
Recent
Installed
Visual C#
Windows
Universal
Windows 8
Classic Desktop
Windows IoT Core
Web
Android
Cloud
Extensibility
iOS
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Online
.NET Framework 4.5.2
Sort by: Default
Search Installed Templates (Ctrl+E)
Blank App (Universal Windows) Visual C#
Class Library (Universal Windows) Visual C#
Windows Runtime Component (Universal Windows) Visual C#
Unit Test App (Universal Windows) Visual C#
Coded UI Test Project (Universal Windows - Phone) Visual C#
Coded UI Test Project (Universal Windows) Visual C#
Type: Visual C#
A project for a single-page Universal Windows Platform app that has no predefined controls or layout.
Click here to go online and find templates.
Browse...
OK
Cancel

Name: BlinkyHeaded
Location: C:\Projects\Windows IoT\Blinky

A new dialog window will open, asking you to choose the target and minimum versions your BlinkyHeaded app will support. Choose the target and

Minimum versions to match the version of Windows IoT Core you have running on your Raspberry Pi. If you're not sure, check the main HDMI display or Device Portal to see the installed version.

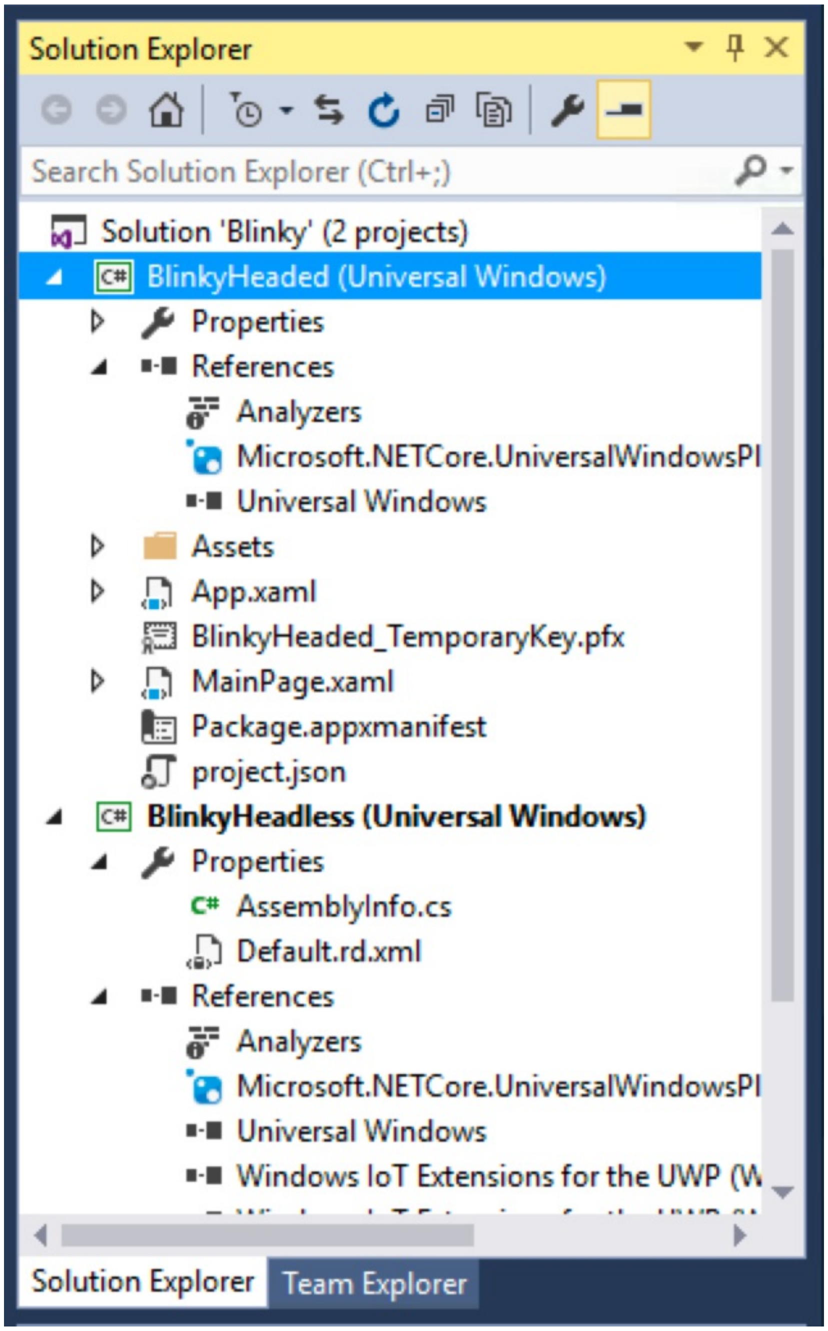

New Universal Windows Project

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

Target Version: Windows 10 (10.0; Build 10586)

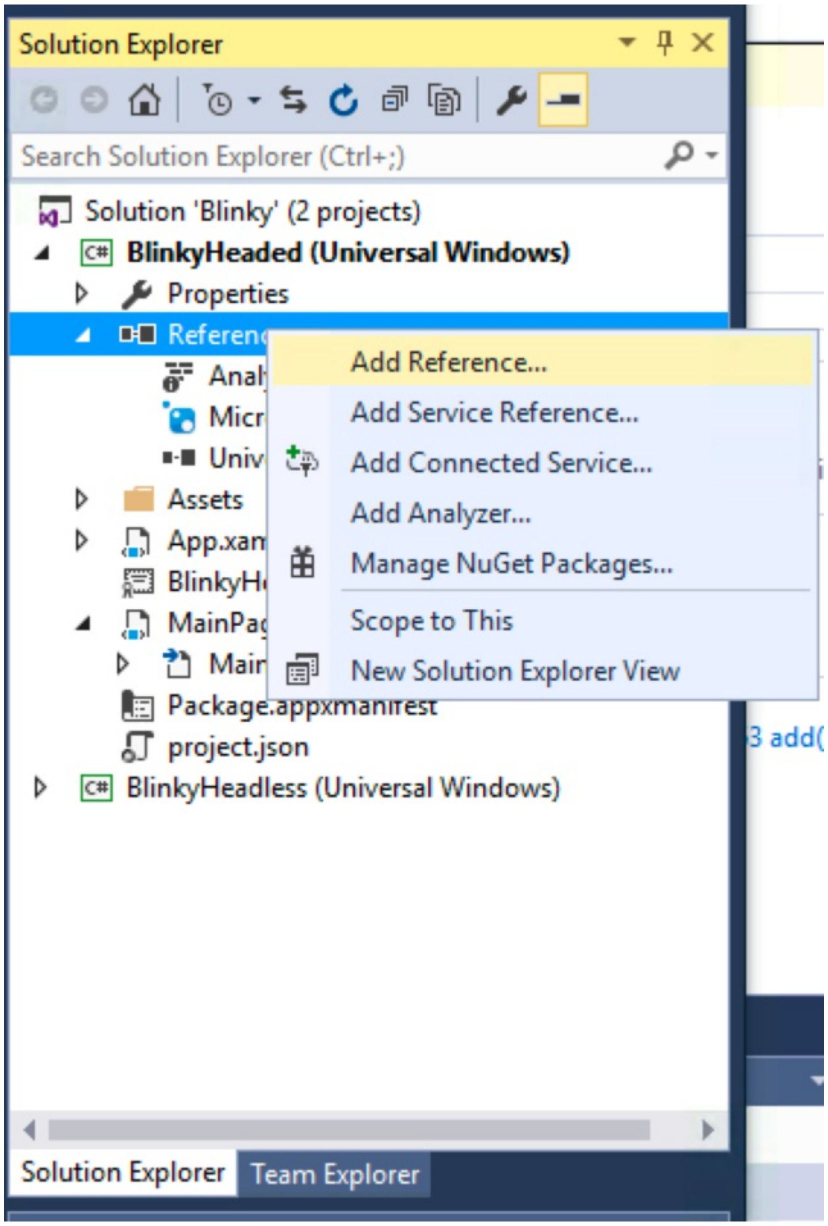
Minimum Version: Windows 10 (10.0; Build 10240)

Click **OK** and the BlinkyHeaded project will appear in the Blinky Solution:



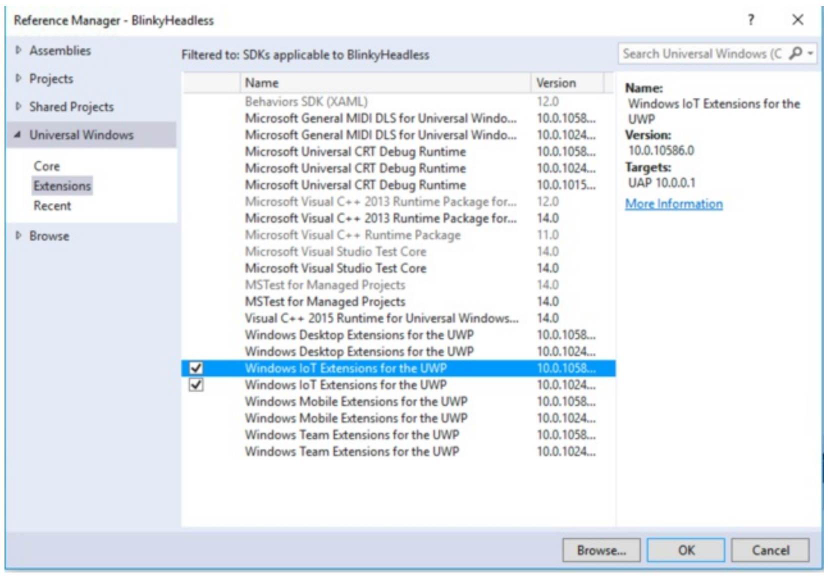
```
Solution 'Blinky' (2 projects)
  BlinkyHeaded (Universal Windows)
    Properties
    References
      Analyzers
      Microsoft.NETCore.UniversalWindowsPl
      Universal Windows
    Assets
    App.xaml
    BlinkyHeaded_TemporaryKey.pfx
    MainPage.xaml
    Package.appxmanifest
    project.json
  BlinkyHeadless (Universal Windows)
    Properties
      AssemblyInfo.cs
      Default.rd.xml
    References
      Analyzers
      Microsoft.NETCore.UniversalWindowsPl
      Universal Windows
      Windows IoT Extensions for the UWP (W
```

Even though we used a Windows IoT template for creating our solution, we still need to add the IoT framework to our BlinkyHeaded project, the same as we did in BlinkyHeadless. In the **Solution Explorer**, right click on the line that says **References**, and select **Add Reference…**



This will bring up the **References Manager**. In the **References Manager Window**, navigate to **Universal Windows > Extensions**. In the list, find **Windows IoT Extensions for the UWP**. There may be more than one item with this name. To the right of each item, you'll see a version number. Select the items that match the Target and Minimum versions you entered earlier.

When you select an item, a checkbox will appear to the left of the item. Check the box for each item you select.


Reference Manager - BlinkyHeadless

- Assemblies
- Projects
- Shared Projects
- Universal Windows
  - Core
  - Extensions
  - Recent
- Browse

Filtered to: SDKs applicable to BlinkyHeadless

Search Universal Windows (C

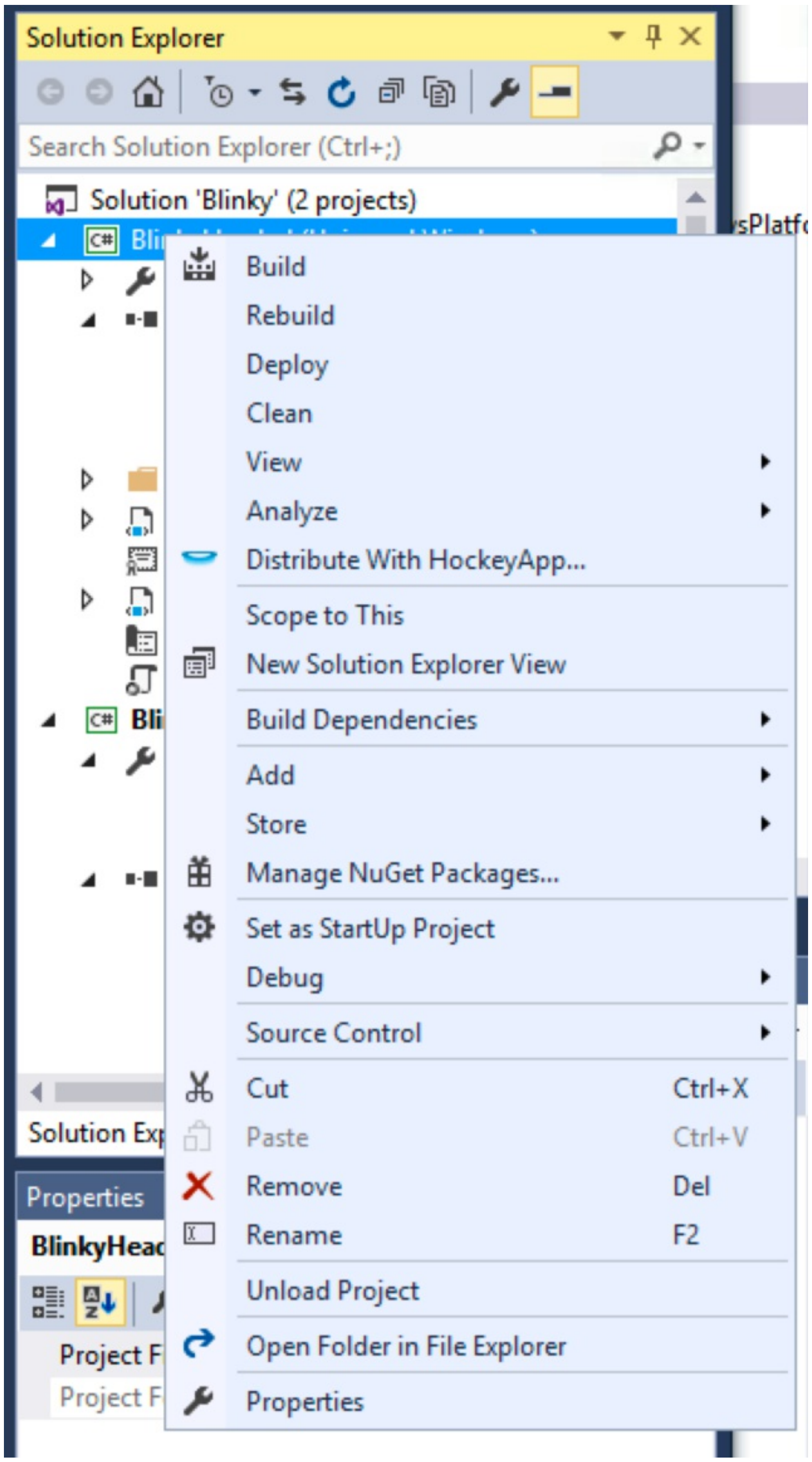
	Name	Version
	Behaviors SDK (XAML)	12.0
	Microsoft General MIDI DLS for Universal Windo...	10.0.1058...
	Microsoft General MIDI DLS for Universal Windo...	10.0.1024...
	Microsoft Universal CRT Debug Runtime	10.0.1058...
	Microsoft Universal CRT Debug Runtime	10.0.1024...
	Microsoft Universal CRT Debug Runtime	10.0.1015...
	Microsoft Visual C++ 2013 Runtime Package for...	12.0
	Microsoft Visual C++ 2013 Runtime Package for...	14.0
	Microsoft Visual C++ Runtime Package	11.0
	Microsoft Visual Studio Test Core	14.0
	Microsoft Visual Studio Test Core	14.0
	MSTest for Managed Projects	14.0
	MSTest for Managed Projects	14.0
	Visual C++ 2015 Runtime for Universal Windows...	14.0
	Windows Desktop Extensions for the UWP	10.0.1058...
	Windows Desktop Extensions for the UWP	10.0.1024...
✓	Windows IoT Extensions for the UWP	10.0.1058...
✓	Windows IoT Extensions for the UWP	10.0.1024...
	Windows Mobile Extensions for the UWP	10.0.1058...
	Windows Mobile Extensions for the UWP	10.0.1024...
	Windows Team Extensions for the UWP	10.0.1058...
	Windows Team Extensions for the UWP	10.0.1024...

Name:
Windows IoT Extensions for the UWP
Version:
10.0.10586.0
Targets:
UAP 10.0.0.1
More Information

Browse... OK Cancel

Click **OK** to add the references to your project.

One more thing to do: We need to tell Visual Studio that we are now working on BlinkyHeaded. Right-click on BlinkyHeaded, and select **Set as StartUp Project** in the popup.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Blinky' (2 projects)
Build
Rebuild
Deploy
Clean
View
Analyze
Distribute With HockeyApp...
Scope to This
New Solution Explorer View
Build Dependencies
Add
Store
Manage NuGet Packages...
Set as StartUp Project
Debug
Source Control
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename F2
Unload Project
Open Folder in File Explorer
Properties

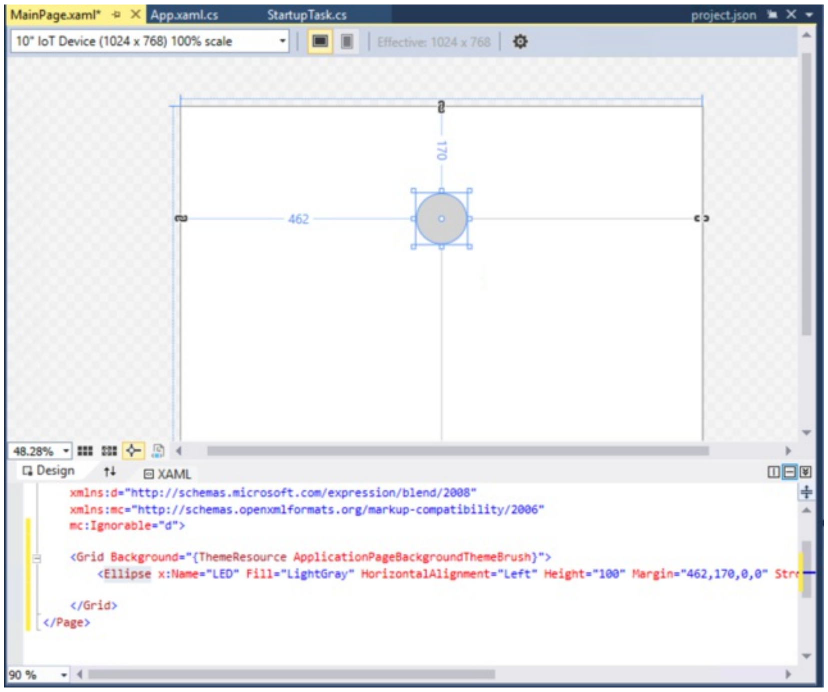
GUI Designer

When the BlinkyHeaded project is added to the solution, Visual Studio will automatically generate several files, and it will open one of these files: App.xaml.cs

This file sets up the application and we really don't need to change anything in it. You can close the tab and open the MainPage.xaml file instead. In Solution Explorer, find MainPage.xaml and double-click on it. This brings up an editing window where you can graphically design your user interface.

The Raspberry Pi is a 10" IoT device, so select that from the box at the upper left of the graphics-editing window.

Now, locate the **Ellipse** control in the **Toolbox**, under **All XAML Controls**. Drag and drop an ellipse on to the edit window. Drag it around until it's centered in the upper half of the panel:

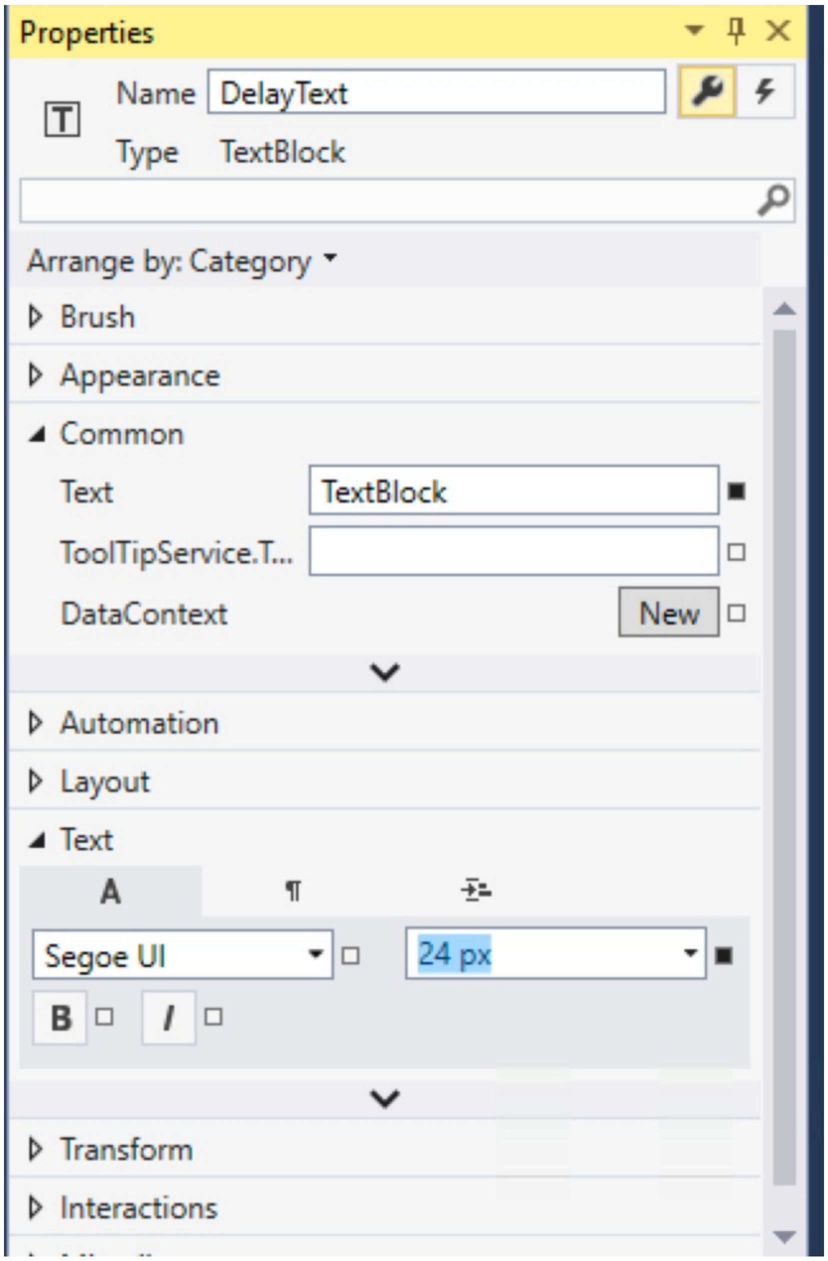


```
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Ellipse x:Name="LED" Fill="LightGray" HorizontalAlignment="Left" Height="100" Margin="462,170,0,0" Str

    </Grid>
</Page>
```

We're going to 'blink' the ellipse on the display, along with blinking the real LED.

In the **Properties** panel, give the ellipse a name, LED. In the BlinkyHeader program, we're planning to blink a real LED, and also blink this ellipse to match the real LED.

Now, drag a text block from the toolbox, and drop it underneath the ellipse. In **Properties**, give it a name, DelayText. In the **Common** box of **Properties**, change the text to 500ms. In the **Text** box of **Properties**, change the font size to 24 px.


Properties

Name DelayText

Type TextBlock

Arrange by: Category

Brush

Appearance

Common

Text: TextBlock

ToolTipService.T...

DataContext: New

Automation

Layout

Text

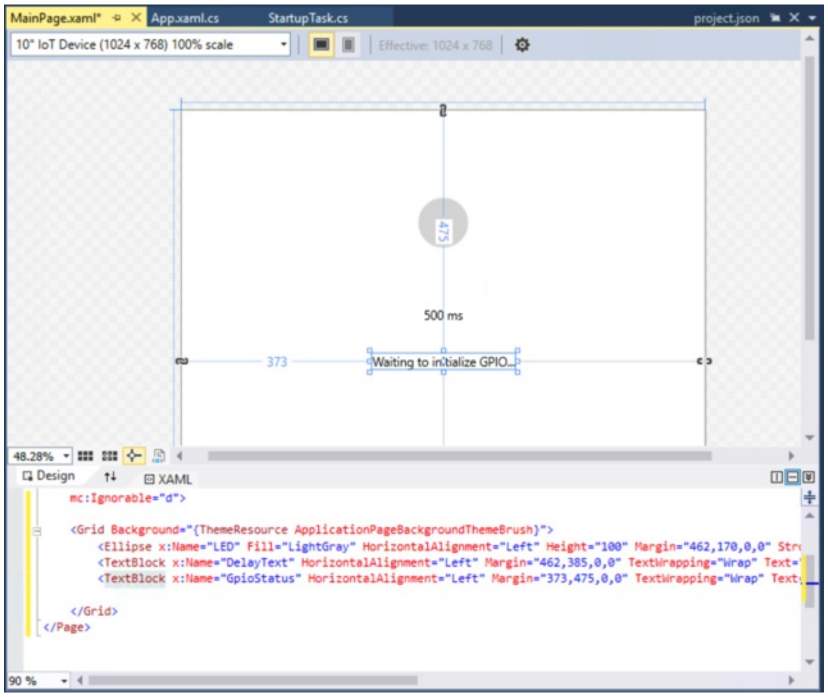
Segoe UI, 24 px

Transform

Interactions

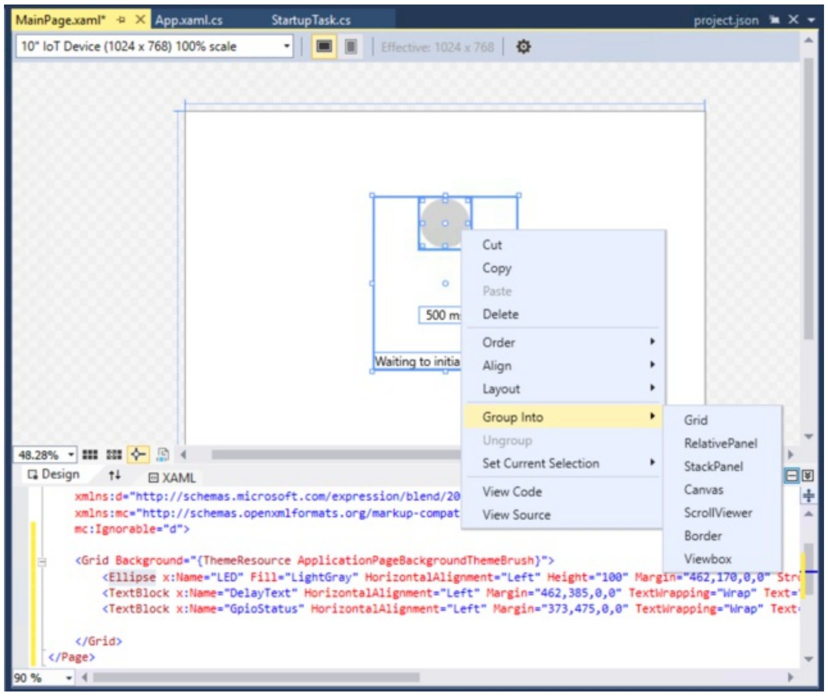
Now, center the text block under the ellipse.

Add one more **TextBlock** underneath the DelayText block. Name it GpioStatus, and change the text to Waiting to initialize GPIO... Make the font size 24 px, and center the box:



```
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Ellipse x:Name="LED" Fill="LightGray" HorizontalAlignment="Left" Height="100" Margin="462,170,0,0" Str
        <TextBlock x:Name="DelayText" HorizontalAlignment="Left" Margin="462,385,0,0" TextWrapping="Wrap" Text="
        <TextBlock x:Name="GpioStatus" HorizontalAlignment="Left" Margin="373,475,0,0" TextWrapping="Wrap" Text

    </Grid>
</Page>
```

Finally, we're going to group our ellipse and text blocks into a**StackPanel**. Select all three objects by dragging over them with the cursor. Right-click on the ellipse, and move the cursor to **Group Into**. From the popup, select**StackPanel**:



```
    xmlns:d="http://schemas.microsoft.com/expression/blend/20
    xmlns:mc="http://schemas.openxmlformats.org/markup-compat
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Ellipse x:Name="LED" Fill="LightGray" HorizontalAlignment="Left" Height="100" Margin="462,170,0,0" Str
        <TextBlock x:Name="DelayText" HorizontalAlignment="Left" Margin="462,385,0,0" TextWrapping="Wrap" Text="
        <TextBlock x:Name="GpioStatus" HorizontalAlignment="Left" Margin="373,475,0,0" TextWrapping="Wrap" Text

    </Grid>
</Page>
```

You'll have to play around with the GUI editor to get things right. Don't be afraid to explore the properties and experiment with items from the toolbox (and don't be afraid to make mistakes!).

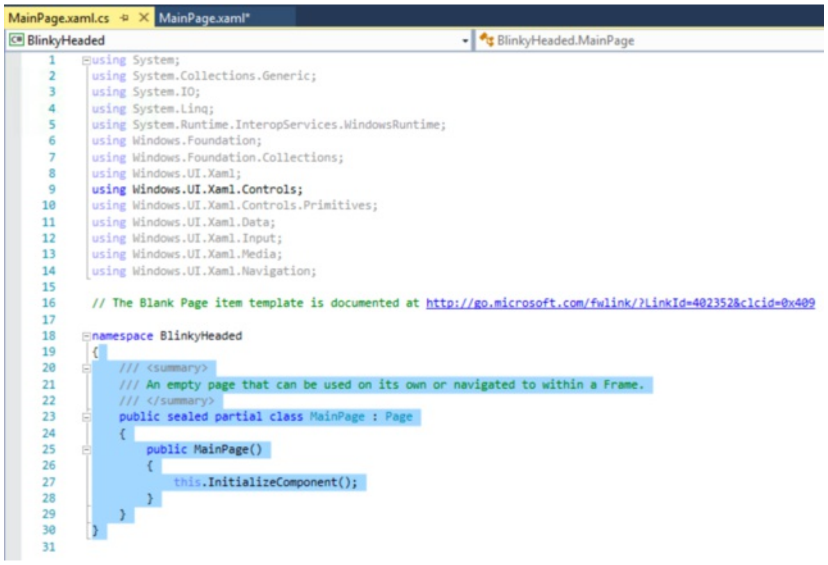
If you want to get back to the gui design used in this tutorial, simply delete thecode in the box underneath the GUI editing window and replace it with the following

```
<Page
    x:Class="BlinkyHeaded.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:BlinkyHeaded"
    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}">
        <StackPanel Margin="373,170,371,261" Orientation="Vertical">
            <Ellipse x:Name="LED" Fill="LightGray" HorizontalAlignment="Center" Height="100" Margin="0,25,0,0" Stroke="White" VerticalAlignment="Top" Width="100"/>
            <TextBlock x:Name="DelayText" HorizontalAlignment="Center" Margin="0,50,0,0" Text="500 ms" VerticalAlignment="Top" FontSize="24" TextAlignment="Center" Width="10
            <TextBlock x:Name="GpioStatus" HorizontalAlignment="Center" Text="Waiting to initialize GPIO..." VerticalAlignment="Center" TextAlignment="Center" FontSize="24" Heigh

        </StackPanel>

    </Grid>
</Page>
```

Main Program

For the purposes of this tutorial, we're going to be using screen captures to illustrate the code in this section. Just follow along and try to understand each step. The full code is available in the next section.

To open the graphics editor, we double-clicked on MainPage.xaml in the **Solution Explorer**. If you single-click on it instead, MainPage.xaml.cs will be shown below. Double-click on MainPage.xaml.cs The file will open in a new edit window, and this will be our main program file:



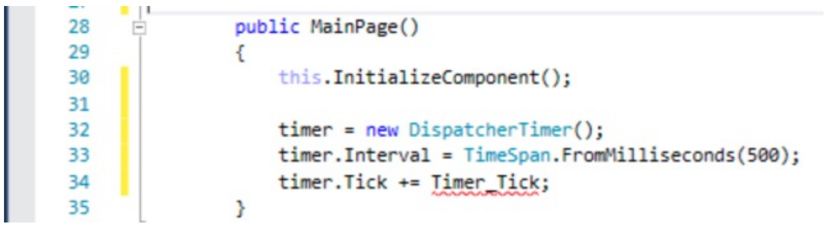
```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

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

Headed applications support threading, just like headless applications. But, there's a detail to be aware of: user interface operations must always occur on the user-interface thread. The MainPage function is originally called on the UI thread, so we need to arrange things such that our UI updates always happen on this same thread.

Although we don't need to update the real LED on the UI thread, we do need to update the virtual LED on the screen from the main thread.

In BlinkyHeadless, we started a new thread for the timer. In BlinkyHeaded, we will use a DispatcherTimer instead. This timer is guaranteed to execute on the main UI thread. It is said to be "synchronous". Meaning that our Timer-Tick routine will be called when the UI thread isn't doing anything else. The BlinkyHeadless timer was "asynchronous", that is, it could happen at any time, regardless of what the main thread is doing.

Let's add the timer to the function MainPage:

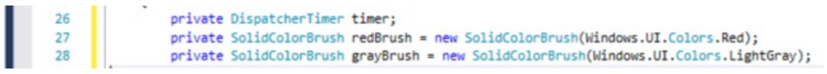


```
        public MainPage()
        {
            this.InitializeComponent();

            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(500);
            timer.Tick += Timer_Tick;
        }
```

Note the error line underneath Timer_Tick. That's because we haven't declared the Timer_Tick function yet. Let's do that next.

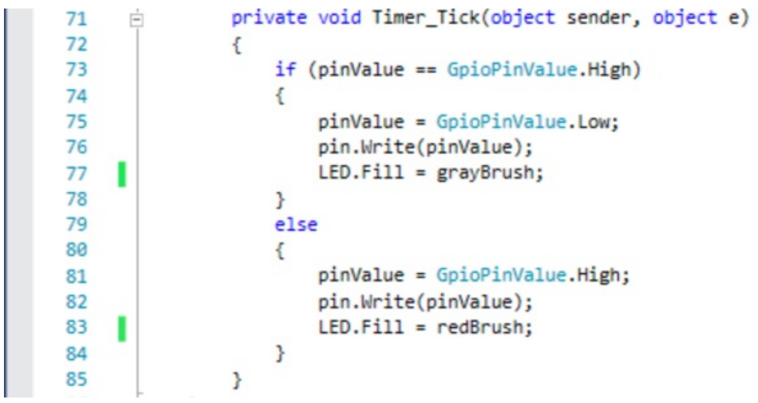
Timer_Tick is going to look just about the same as Timer_Tick from BlinkyHeadless. But we're going to add the code to "blink" the LED on screen. We do that by filling the ellipse with either red or gray color.

Our colors will be controlled by the **Brush** object. We'll need a gray_brush object and a red_brush object. These will be declared as global objects, and we'll put them right after the DispatcherTimer global object:



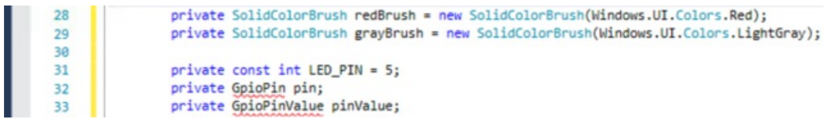
```
        private DispatcherTimer timer;
        private SolidColorBrush redBrush = new SolidColorBrush(Windows.UI.Colors.Red);
        private SolidColorBrush grayBrush = new SolidColorBrush(Windows.UI.Colors.LightGray);
```

Recall that we named the ellipse LED. Filling the ellipse object is straightforward; we just call the Fill method of the LED object. Here's our Timer_Tick function:



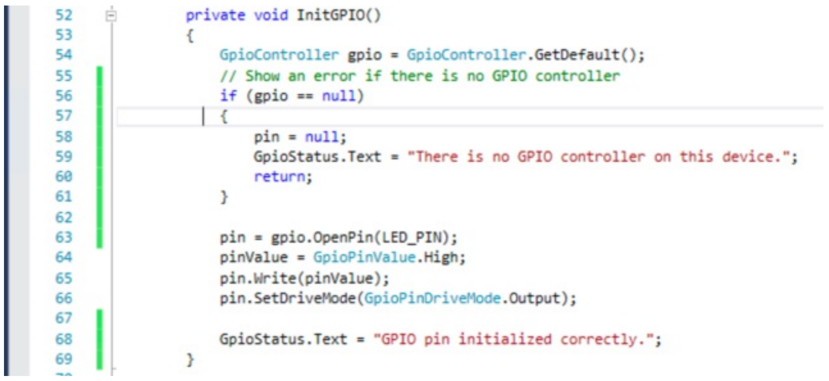
```
        private void Timer_Tick(object sender, object e)
        {
            if (pinValue == GpioPinValue.High)
            {
                pinValue = GpioPinValue.Low;
                pin.Write(pinValue);
                LED.Fill = grayBrush;
            }
            else
            {
                pinValue = GpioPinValue.High;
                pin.Write(pinValue);
                LED.Fill = redBrush;
            }
        }
```

Of course, we have some errors because we haven't declared ourGpio objects yet. We'll need a InitGPIO function, just like we had in BlinkyHeadless. We'll also need the same set of Gpio variables we had in BlinkyHeadless:



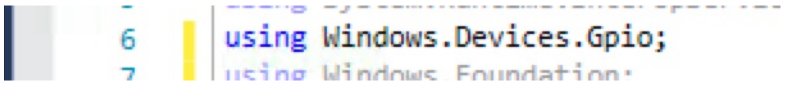
```
        private SolidColorBrush redBrush = new SolidColorBrush(Windows.UI.Colors.Red);
        private SolidColorBrush grayBrush = new SolidColorBrush(Windows.UI.Colors.LightGray);

        private const int LED_PIN = 5;
        private GpioPin pin;
        private GpioPinValue pinValue;
```

We'll just copy the InitGPIO function over from BlinkyHeadless, and add some messages to take advantage of theGpioStatus text block we included in the UI:



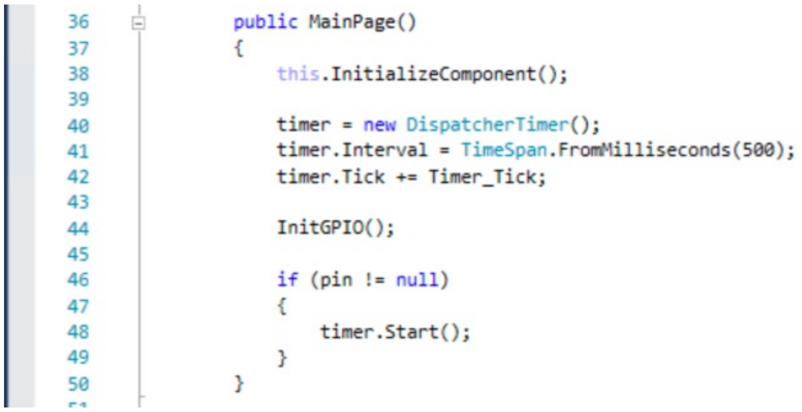
```
        private void InitGPIO()
        {
            GpioController gpio = GpioController.GetDefault();
            // Show an error if there is no GPIO controller
            if (gpio == null)
            {
                pin = null;
                GpioStatus.Text = "There is no GPIO controller on this device.";
                return;
            }

            pin = gpio.OpenPin(LED_PIN);
            pinValue = GpioPinValue.High;
            pin.Write(pinValue);
            pin.SetDriveMode(GpioPinDriveMode.Output);

            GpioStatus.Text = "GPIO pin initialized correctly.";
        }
```

We still have some syntax errors, but those will go away when include our*using* statement for Windows.Devices.Gpio



```
using Windows.Devices.Gpio;
```

All that's left to do is callInitGPIO from the MainPage function, start the timer, and the program is done:



```
        public MainPage()
        {
            this.InitializeComponent();

            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(500);
            timer.Tick += Timer_Tick;

            InitGPIO();

            if (pin != null)
            {
                timer.Start();
            }
        }
```

Code

Here's the final version of MainPage.xaml.cs There are other source code files in the BlinkyHeaded project, but they are automatically generated by Visual Studio, and we don't need to be concerned with them now.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Devices.Gpio;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace BlinkyHeaded
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    ///
    public sealed partial class MainPage : Page
    {
        private DispatcherTimer timer;

        private SolidColorBrush redBrush = new
SolidColorBrush(Windows.UI.Colors.Red);
        private SolidColorBrush grayBrush = new
SolidColorBrush(Windows.UI.Colors.LightGray);

        private const int LED_PIN = 5;
        private GpioPin pin;
        private GpioPinValue pinValue;

        public MainPage()
        {
            this.InitializeComponent();

            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(500);
            timer.Tick += Timer_Tick;

            InitGPIO();

            if (pin != null)
            {
                timer.Start();
            }
        }

        private void InitGPIO()
        {
            GpioController gpio = GpioController.GetDefault();
            // Show an error if there is no GPIO controller
            if (gpio == null)
            {
                pin = null;
                GpioStatus.Text = "No GPIO controller on device.";
                return;
            }

            pin = gpio.OpenPin(LED_PIN);
            pinValue = GpioPinValue.High;
            pin.Write(pinValue);
            pin.SetDriveMode(GpioPinDriveMode.Output);

            GpioStatus.Text = "GPIO pin initialized.";
        }

        private void Timer_Tick(object sender, object e)
        {
            if (pinValue == GpioPinValue.High)
            {
                pinValue = GpioPinValue.Low;
                pin.Write(pinValue);
                LED.Fill = grayBrush;
            }
            else
            {
                pinValue = GpioPinValue.High;
                pin.Write(pinValue);
                LED.Fill = redBrush;
            }
        }
    }
}
```

Running BlinkyHeaded

Running the application from Visual Studio is the same as running BlinkyHeadless. Set the run button to **Remote Machine**, and click.

Be prepared to wait several minutes for BlinkyHeaded to start up the first time. Be patient!

Remember, you can set breakpoints and single-step in the debugger, just as we did in BlinkyHeadless.

Next: Set Your Application to Run at Startup
http://adafru.it/psD

FAQ

I get a deployment error from Visual Studio: DEP0001 : Unexpected Error: -2145615869

This error typically happens after you reboot your Pi. For whatever reason, Visual Studio is no longer able to communicate with it. The solution is to simply close and restart Visual Studio. Do this every time you reboot your Pi.