Windows IoT Core Application Development: Headed Blinky

Created by Rick Lesniak

https://learn.adafruit.com/windows-iot-application-development-headed-blinky

Last updated on 2023-08-29 03:11:36 PM EDT
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Add BlinkyHeaded project to Solution</td>
<td>3</td>
</tr>
<tr>
<td>GUI Designer</td>
<td>9</td>
</tr>
<tr>
<td>Main Program</td>
<td>13</td>
</tr>
<tr>
<td>Code</td>
<td>16</td>
</tr>
<tr>
<td>Running BlinkyHeaded</td>
<td>18</td>
</tr>
<tr>
<td>FAQ</td>
<td>18</td>
</tr>
</tbody>
</table>
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).

IMPORTANT: Before going forward, make sure you have successfully completed the BlinkyHeadless app. This tutorial assumes you will be building on that.

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 these 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

![Solution Explorer with Add Project option](image)

- 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
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.

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

![Ellipse in XAML Editor](image)

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`.
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:
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:

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 the code in the box underneath the GUI editing window and replace it with the following:

```xml
&lt;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"
  mc:Ignorable="d">
  &lt;Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    &lt;StackPanel Margin="373,170,371,261" Orientation="Vertical">
      &lt;Ellipse x:Name="LED" Fill="LightGray" HorizontalAlignment="Center"
        Height="100" Margin="0,25,0,0" Stroke="White" VerticalAlignment="Top" Width="100"/>
      &lt;TextBlock x:Name="DelayText" HorizontalAlignment="Center"
        Margin="0,50,0,0" Text="500 ms" VerticalAlignment="Top" FontSize="24"
        TextAlignment="Center" Width="100"/>
      &lt;TextBlock x:Name="GpioStatus" HorizontalAlignment="Center"
        Text="Waiting to initialize GPIO..." VerticalAlignment="Center"
        TextAlignment="Center" FontSize="24" Height="34" Margin="0,50,0,0" Width="270"/>
    &lt;/StackPanel&gt;
  &lt;/Grid&gt;
&lt;/Page&gt;
```

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:
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()
    {
        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:

```csharp
private DispatcherTimer timer;
private SolidColorBrush redBrush = new SolidColorBrush(Colors.Red);
private SolidColorBrush grayBrush = new SolidColorBrush(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:

```csharp
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 our **Gpio** 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:

```csharp
private SolidColorBrush redBrush = new SolidColorBrush(Colors.Red);
private SolidColorBrush grayBrush = new SolidColorBrush(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 the **GpioStatus** text block we included in the UI:

```csharp
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 = "GPI pin initialized correctly.";
}
```
We still have some syntax errors, but those will go away when include our using statement for **Windows.Devices.Gpio**

All that’s left to do is call **InitGPIO** from the **MainPage** function, start the timer, and the program is done:

```csharp
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.

```csharp
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
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.

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.