Source: C# Corner (www.c-sharpcorner.com) PRINT

Article

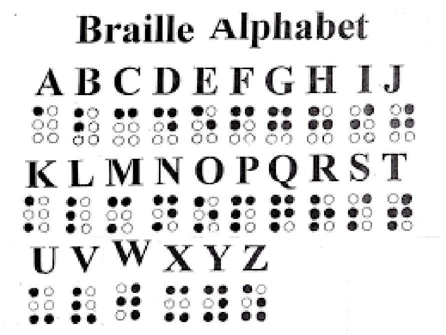
Braille Language In Raspberry Pi Using C# With Visual Studio

By **Kishore Chowdary** on **Feb 12 2017**

This article will help you in writing the code to your Raspberry Pi for a UWP using which you can apply the sequence of the alphabet that can be easily read by blind people. This language is called Braille language. Here, there will be six dots which will be touched and felt by blind people. For all the twenty-six alphabet letters in English, we have twenty-six different pattern types. These patterns can be formed with the help of just six GPIO pins in Raspberry Pi. Before we get into this, let me explain a bit more about this language.

Braille language

Braille is a tactile writing system used by people who are blind or visually impaired. It is traditionally written with embossed paper. Braille-users can read computer screens and other electronic supports thanks to refreshable braille displays. They can write braille with the original slate and stylus or type it on a braille writer, such as a portable braille note-taker, or on a computer that prints with a braille embosser. The pattern for each alphabet for this language is shown in the image below.

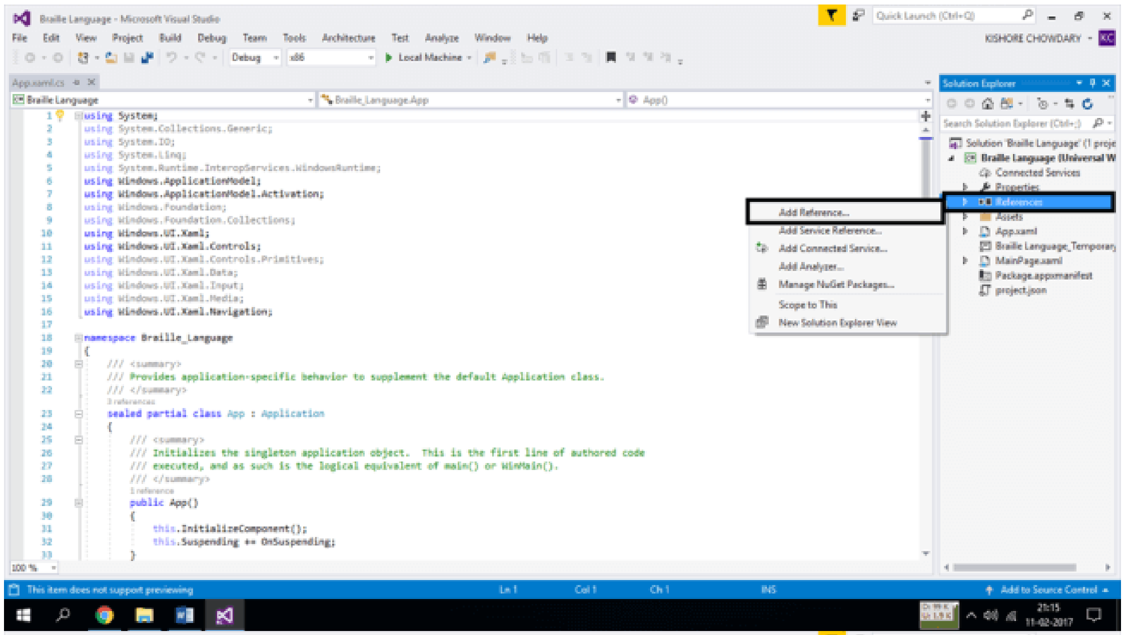


Requirements

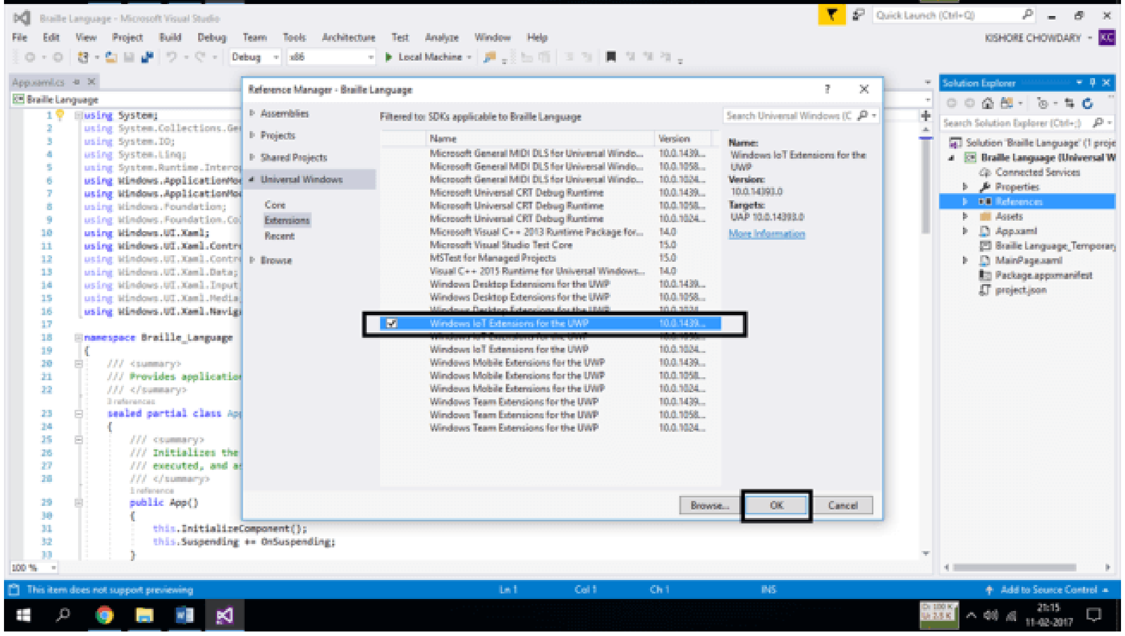
1. Solenoid Actuator – X6 nos
2. Jumping wires
3. Relay – X6 nos
4. Raspberry Pi

Creating UWP

Now, let us create a new UWP application. For this, open up Visual Studio and go to FileàNew ProjectàUniversal Windows Platform. This will create a new project. Now to make this application work with your Raspberry Pi, you need to add a reference file to your project. For this, right click on the reference files option in the solution explorer and then click on the add reference file option. This will show you a set of reference files. Now search for the reference file named “Windows IOT Extension for UWP” and add it to your project. This is the extension which you need to access your remote IOT device.



```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel;
using Windows.ApplicationModel.Activation;
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;

namespace Braille_Language
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application class.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object.  This is the first line of authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
        public App()
        {
            this.InitializeComponent();
            this.Suspending += OnSuspending;
        }
```



Reference Manager - Braille Language

Filtered to: SDKs applicable to Braille Language

Name	Version
Microsoft General MIDI DLS for Universal Windo...	10.0.1439...
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	10.0.1439...
Microsoft Universal CRT Debug Runtime	10.0.1058...
Microsoft Universal CRT Debug Runtime	10.0.1024...
Microsoft Visual C++ 2013 Runtime Package for...	14.0
Microsoft Visual Studio Test Core	15.0
MSTest for Managed Projects	15.0
Visual C++ 2015 Runtime for Universal Windows...	14.0
Windows Desktop Extensions for the UWP	10.0.1439...
Windows Desktop Extensions for the UWP	10.0.1058...
Windows Desktop Extensions for the UWP	10.0.1024...
✓ Windows IoT Extensions for the UWP	10.0.1439...
Windows IoT Extensions for the UWP	10.0.1058...
Windows IoT Extensions for the UWP	10.0.1024...
Windows Mobile Extensions for the UWP	10.0.1439...
Windows Mobile Extensions for the UWP	10.0.1058...
Windows Mobile Extensions for the UWP	10.0.1024...
Windows Team Extensions for the UWP	10.0.1439...
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: 10.0.14393.0
Targets: UAP 10.0.14393.0

Writing code

We are going to use the pin numbers 5, 6, 13, 19, 26, 21. To make all these simple let us have these pins as following.

Pin 5 = a

Pin 6 = b

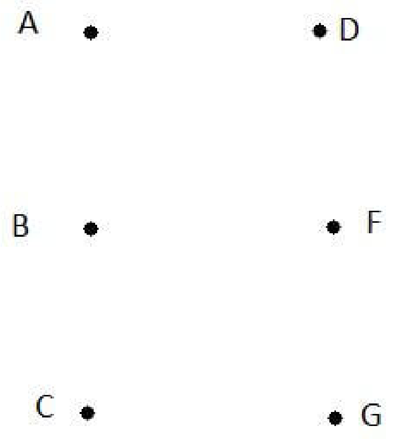
Pin 13 = c

Pin 19 = d

Pin 26 = f

Pin 21 = g

The code which we are going to write now is based on the conceptual pattern of the bailey language. Here we will be using six GPIO pins and one ground pin. For every different pattern we will be using different format of coding. Have a look over the image below to know about the patterns of the language. Look at the pattern of letter A. To make the pattern to work out, you need to make pin **a** high and rest of the pins namely **b, c, d, f, g.** I have shown the image of the pattern and the naming and pin numbers for the pattern.



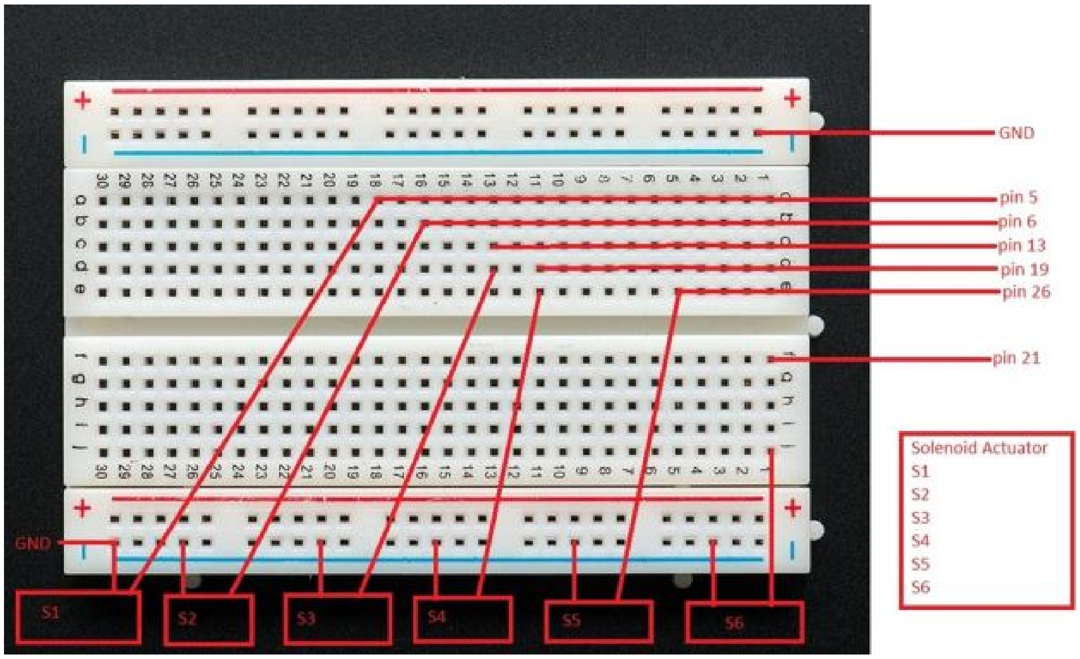
Variables assigned to the Pins

Have a look at the code below. Just copy and paste the code in the main page to get the same output for the Braille concept. In the following code, I have written only for three alphabets. You just alter the code for the rest of the alphabets and numbers.

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.Devices.Gpio;

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

namespace Braille_Language
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        GpioPin a, b, c, d, f,g;
        public MainPage()
        {
            this.InitializeComponent();
            Loaded += MainPage_Loaded;
        }

        private void MainPage_Loaded(object sender, RoutedEventArgs e)
        {

            //pin opening

            var controller = GpioController.GetDefault();
            a = controller.OpenPin(5);
            b = controller.OpenPin(6);
            c = controller.OpenPin(13);
            d = controller.OpenPin(19);
            f = controller.OpenPin(26);
            g = controller.OpenPin(21);

            //setting pin mode

            a.SetDriveMode(GpioPinDriveMode.Output);
            b.SetDriveMode(GpioPinDriveMode.Output);
            c.SetDriveMode(GpioPinDriveMode.Output);
            d.SetDriveMode(GpioPinDriveMode.Output);
            f.SetDriveMode(GpioPinDriveMode.Output);
            g.SetDriveMode(GpioPinDriveMode.Output);
            ///////////////////////////////////////////////////////////////////////
            //Enabling Solenoid Actuator for letter A

            a.Write(GpioPinValue.High);
            b.Write(GpioPinValue.Low);
            c.Write(GpioPinValue.Low);
            d.Write(GpioPinValue.Low);
            f.Write(GpioPinValue.Low);
            g.Write(GpioPinValue.Low);

            ////////////////////////////////
            //Normal position

            a.Write(GpioPinValue.Low);
            b.Write(GpioPinValue.Low);
            c.Write(GpioPinValue.Low);
            d.Write(GpioPinValue.Low);
            f.Write(GpioPinValue.Low);
            g.Write(GpioPinValue.Low);

            ////////////////////////////////////////////////////////////////////////////
            //Enabling Solenoid Actuator for letter b

            a.Write(GpioPinValue.High);
            b.Write(GpioPinValue.High);
            c.Write(GpioPinValue.Low);
            d.Write(GpioPinValue.Low);
            f.Write(GpioPinValue.Low);
            g.Write(GpioPinValue.Low);

            ////////////////////////////////
            //Normal position

            a.Write(GpioPinValue.Low);
            b.Write(GpioPinValue.Low);
            c.Write(GpioPinValue.Low);
            d.Write(GpioPinValue.Low);
            f.Write(GpioPinValue.Low);
            g.Write(GpioPinValue.Low);

            ////////////////////////////////////////////////////////////////////////////
            //Enabling Solenoid Actuator for letter c

            a.Write(GpioPinValue.High);
            b.Write(GpioPinValue.Low);
            c.Write(GpioPinValue.Low);
            d.Write(GpioPinValue.High);
            f.Write(GpioPinValue.Low);
            g.Write(GpioPinValue.Low);

            ////////////////////////////////
            //Normal position

            a.Write(GpioPinValue.Low);
            b.Write(GpioPinValue.Low);
            c.Write(GpioPinValue.Low);
            d.Write(GpioPinValue.Low);
            f.Write(GpioPinValue.Low);
            g.Write(GpioPinValue.Low);

/*            The same way you just change the value of pins from high to low based on the patterns of the Braille language
            and write the code for the rest of the aplhabets and numbers                              */


        }
    }
}
```

Connecting the Raspberry Pi

Now, it is time to connect your raspberry pi with the Solenoid Actuator with the help of the breadboard along with the relay. I will simply explain the concept of connecting the Solenoid Actuator to the raspberry pi without the relay. But it is a must to connect the relay with the solenoid actuator to get an efficient output. Connect the pins as shown in the images below.



Execution

Once you finish the connection, connect the device to your application and then upload the program to your device. Since we have written the code of the alphabets in the sequence with a small delay in there, the alphabets will be executed at an interval of 2 seconds until the letter Z. Hope this will help you in creating an IOT based braille language app. Comment below if you find any difficulties in executing this.

Thanks for reading.