John P Biehle

Kinderlert

Department of Electrical and Computer Engineering Technology

Purdue School of Engineering and Technology

Indiana University - Purdue University Indianapolis

December 3, 2018

Dr David Goodman

TABLE OF CONTENTS

Abstract

This document will explain the aspects of the Kinderlert system. This document will encompass the specification of the system along with the programming layout. The system hardware will be discussed along with an overview of some of the key software components. This document will end with the detail software programming for both the Kinderlert application and the Kinderlert device programming.

REVISION HISTORY

Version	Date	Revised by	Description
1.0	10 November, 2018	John P. Biehle	Initial version
2.0	25 November, 2018	John P. Biehle	Add abstract text
3.0	2 December, 2018	John P. Biehle	Add Coding, Introduction, and software enhancement

I. Introduction

Throughout history, children stray from their parents. When this happens, the parents are left in a panic and look frantically for the lost child. The Kinderlert system will notify a parent if a child wonders too far from the parent. For this document, all aspects of the Kinderlert device will be outlined. The reference documents will outline items found outside of this manual. The system overview will highlight the product specification, software flowcharts, and what software is needed to program the Kinderlert system. The hardware section will outline the devices used to create the Kinderlert system. The software section will discuss some highlights for both the Android programming and the Kinderlert device programming followed by the functional testing for the Kinderlert system. The last sections found in the Appendixes will give the detailed programming needed for the Kinderlert system. If this document is followed, detailed knowledge of the Kinderlert system will be outlined.

II. Reference Documents

Table 1: Reference DocumentsTitle	**Document Reference Number**	**Comment**
System Requirement Specification	SRS001 rev.1	Submitted 03/15/2017
Process Flow Diagram	PFD001 rev.3	Submitted 11/27/2018
Kinderlert App Code	KNDR001 rev 1	Submitted 10/28/2018
Kinderlert Dev Code	KNDRDEV rev 4	Submitted 11/27/2018
System Operational Manual	OM001 rev.1	In Draft

III. Specification

A. The phone system will need to be able to interact with the Kinderlert device. The minimum software version will be API23. The Android-based program written will provide a software link between the phone and the child detection device. When the child detection device loses contact with the phone or the distance is farther than what is programmed, the alert system will be activated by vibrating the out of balance motor on the Android device. A noise notification setting can be set as well within the Android program.

The Kinderlert system is capable of processing connection between the Kinderlert system and the Android device. Adafruit Feather is an open source design. The Adafruit Feather is able activate the audible alert system when a predetermined state is achieved. The processor will also able to reset the notification system and return to a monitoring system when the operator so chooses.

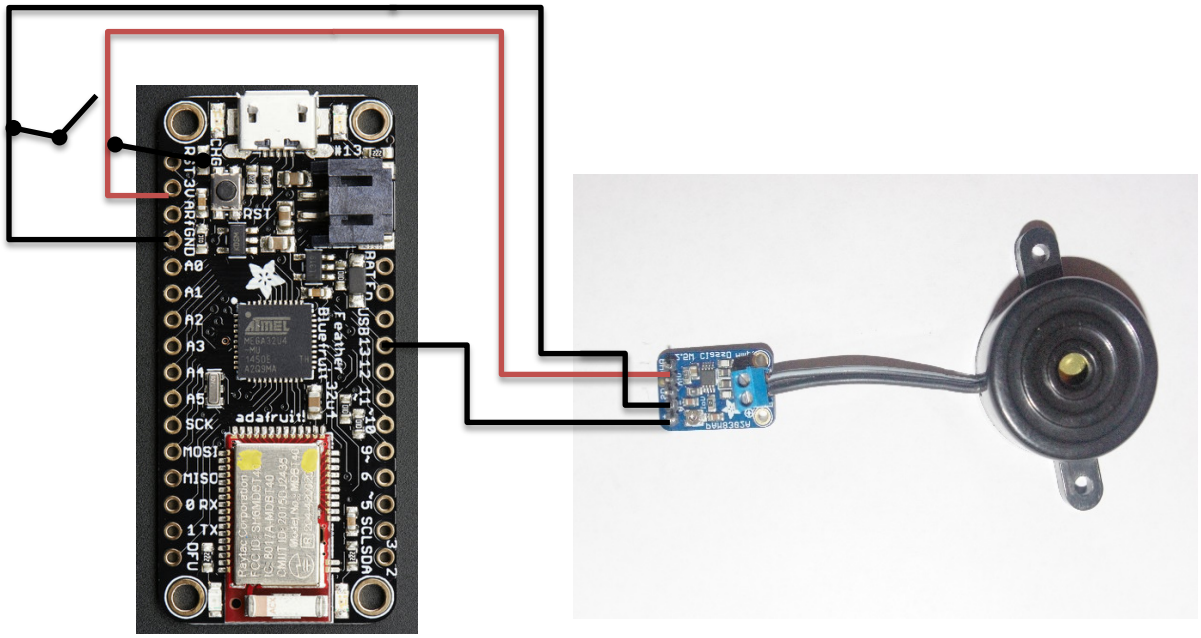
The wireless interface is a Bluetooth Low Energy (BLE) design allowing a low power connection between the phone and the Kinderlert device with enough signal strength to maintain a stable connection. The Android device will automatically pair to make initial connection with Kinderlert system utilizing the Kinderlert device name. Using direct connect method would prevent an unauthorized user from connecting and disabling the child detection system. For the audio portion of the device, the trigger would come from the microprocessor in the form of a various tones dependent on the distance from the host device. The Kinderlert device will sound tones between 1 KHZ, and 10 KHZ. To drive the signal will be an audio amplifier circuit of greater than 2.5 Watts.

There are two possible methods to reset the Kinderlert system. One method would be a reset button on the Kinderlert device. The second method would be using the Android application present on the phone in which a button press would send a reset command through the communication portion of the system to the Kinderlert device.

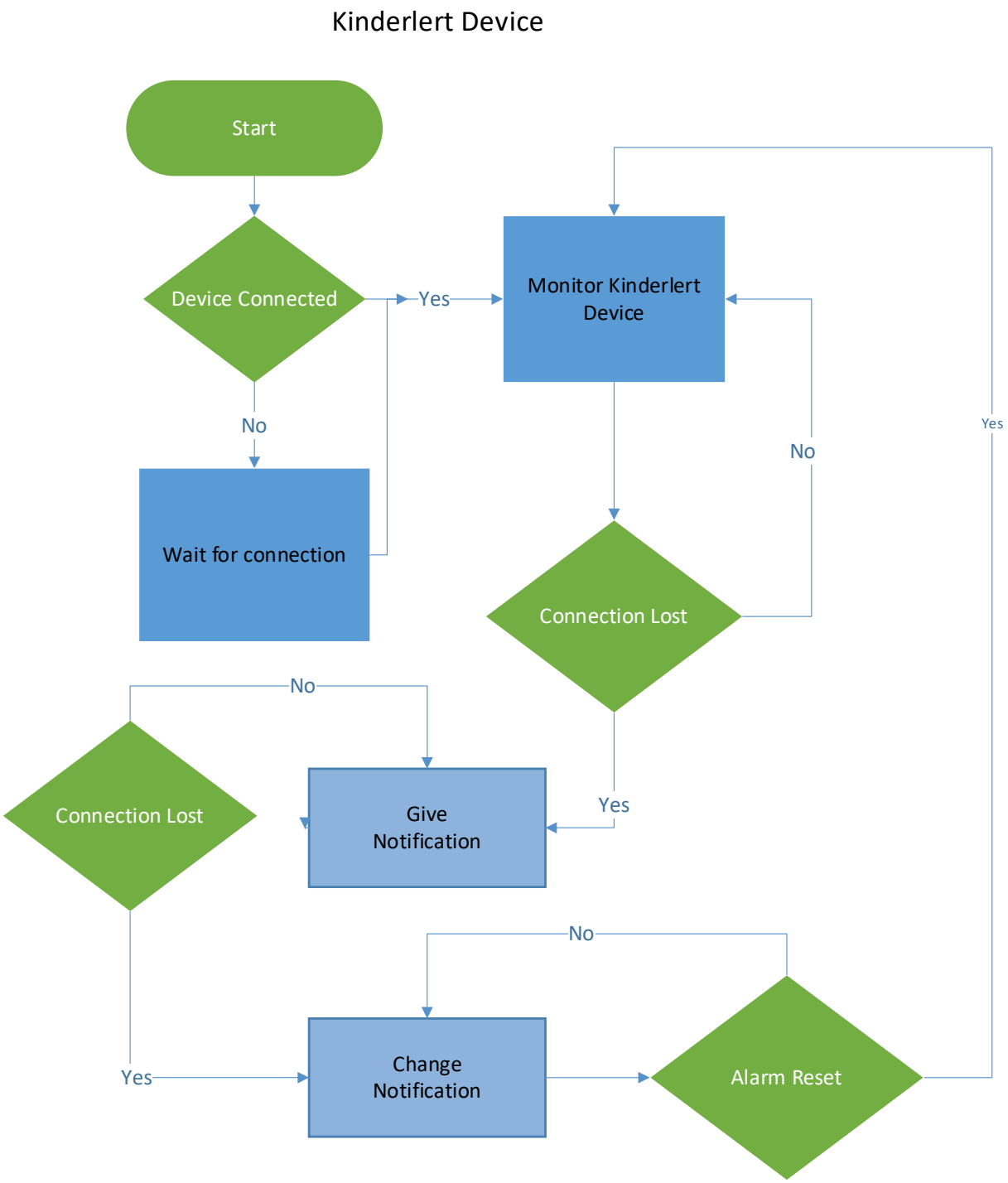
For the power of the Kinderlert device, a rechargeable lithium ion battery will allow recharging through a micro USB connector. A battery level sensing circuit will monitor the starting and stopping of the charging of the Kinderlert device. A charge indicator allows the user to visualize whether the Kinderlert device is charging or has a full charge. Programming within the Kinderlert device allows a user to hear audible tones indicating the charge level of the Kinderlert device on power up of the Kinderlert device.

The Kinderlert system has the ability to detect distance using the RSSI signal strength. Depending on the distance, the audible alarm changes to allow the parent to determine how far they are from the child. With the broad range of values for RSSI, a range of values will determine which audible alarm will be triggered.

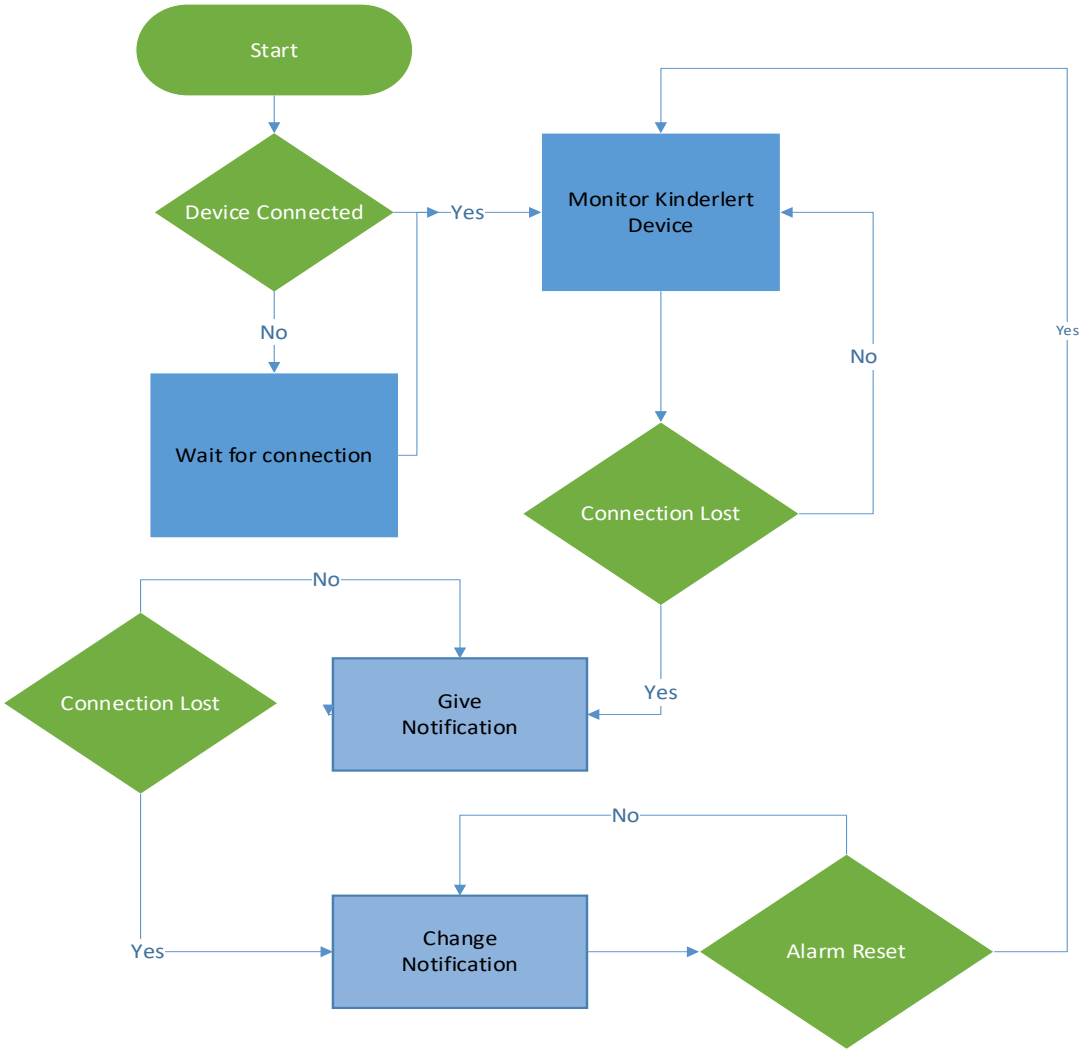
B. Block Diagram



C. Programming Flowchart



Kinderlert Device



D. Software Interface

To interface the Android mobile phone, the Xamarin portion of Visual Studios 2017 was used for coding and debugging. Xamarin allowed for the completed code to be loaded to the Android mobile phone. Using the debug portion of Xamarin, the program was able to step through the code using breakpoints to ensure the code operated as expected and to make changes to the code if the expected results were not executed.

To program the Kinderlert device, the Arduino software program allowed for coding and debugging. The coding was setup and downloaded to the Kinderlert device. After the program was downloaded, the serial connection within the Arduino program allowed for debugging and making correction to the Kinderlert device.

IV. HARDWARE

A. Adafruit Feather 32U4 Bluefruit LE

"The Adafruit Feather 32U4 Bluefruit LE- our take on an 'all-in-one' Arduino-compatible + Bluetooth Low Energy with built in USB and battery charging. It's an Adafruit Feather 32u4 with a BLE module.

Bluetooth Low Energy is the hottest new low-power, 2.4GHz spectrum wireless protocol. It's the only wireless protocol that you can use with iOS without needing special certification and it's supported by all modern smart phones.

At the Feather 32u4's heart is at ATmega32u4 clocked at 8 MHz and at 3.3V logic. This chip has 32K of flash and 2K of RAM, with built in USB so not only does it have a USB-to-Serial program & debug capability built in with no need for an FTDI-like chip, it can also act like a mouse, keyboard, USB MIDI device, etc.

To make it easy to use for portable projects, we added a connector for any of our 3.7V Lithium polymer batteries and built in battery charging." (ada, 2018)

Adafruit Feather 32U4 Bluefruit LE detailed specifications

- Measures 2.0" x 0.9" x 0.28" (51mm x 23mm x 8mm) without headers soldered in
- Light as a (large?) feather - 5.7 grams
- ATmega32u4 @ 8MHz with 3.3V logic/power
- 3.3V regulator with 500mA peak current output
- USB native support, comes with USB bootloader and serial port debugging
- You also get tons of pins - 20 GPIO pins
- Hardware Serial, hardware I2C, hardware SPI support
- 7 x PWM pins
- 10 x analog inputs
- Built in 100mA lipoly charger with charging status indicator LED
- Pin #13 red LED for general purpose blinking
- Power/enable pin
- 4 mounting holes
- Reset button

B. Android mobile device

The mobile device will need to connect with Bluetooth Low Energy technology. The Kinderlert software requires an android mobile device running Android version of 6(Marshmallow) API 23. Older versions of Android and mobile devices that do not support at least Bluetooth version 4 are not compatible with Bluetooth Low Energy will not interact with the Kinderlert device.

IV. Software

A. Android Programming

Appendix A has the complete Android programming. I will touch on a few highlights of the Android code.

Line 44 of the code: “BluefruitLEManager myConnection = new BluefruitLEManager();” setups the link between the main class for the program and the Bluefruit LE Manager class. The Bluefruit LE Manager class setups the BLE scanner for the Android. application. When the “connect” button is depressed in the Android application, scanning begins for the device “Adafruit Bluefruit LE” as found in line 307 if the Android application. Upon discovery of the Bluefuit LE device, the Android application then attempts to make a Bluetooth connection with the Kinderlert device. After the connection has completed the BLE scanner then stops.

Another section of code that is necessary is the coding for the audible alert embedded in the Android program. The code starts at line 444 with the program call of “AlertTone”. This routine sets up for an audible alert one of approximately 8K HZ. When the Bluetooth connection is lost. The Android application calls up the 8K HZ tone and will remain on until a “reset” button is depressed by the user.

B. Adafruit Feather 32U4 Bluiefruit LE Programming

The complete Kinderlert device code is located in Appendix B. This section will highlight a few aspects of the Kinderlert device code.

The first aspect is the Kinderlert Device has the ability to adjust the output power level of the Bluetooth signal. For my current code as seen in line 131, the power level is set using the

“AT+BLEPOWERLEVEL” command with a power level of -4 DB. This is midrange power setting for the code. This allows for an indoor to outdoor distance of approximately 20 feet.

Another aspect of the program is the ability to monitor the Bluetooth connection through using “!ble.isConnected” as seen in lines 258 and 265. When the Bluetooth loses connection, the program will sub into the alert routing to sound the alert tone. When the Bluetooth is reconnected, the tone is then changed allowing the parent to know they are getting closer to the child.

V. SYSTEM TESTING

A. Functional Test

Testing for the Kinderlert was completed using both an indoor to outdoor scenario for the initial testing. During initial testing, the distance going from indoor to outdoors allowed for a distance of approximately 60 feet. This distance is far too great, so the Bluetooth power was decreased allowing for approximately 20 feet of distance between the Android device and the Kinderlert device. The reconnect distance is approximately 17 feet and the reconnect tone changed providing notification the reconnect happened between the Android device and Kinderlert device. For outside line of site distance, the device has an alert distance that averages 40 feet. The reconnect timing and distance averaged approximately 35 feet. Again, the system gave the reconnect tone notification as the system should have provided. When adjusting the Kinderlert device power, the inside to outside and line of site distance will need to be considered.

When initially testing the battery charge programming, the battery level notification happened constantly. The programming changed to allow an initial check at startup. A delay timer was added to check the voltage with no alert tone of every 3ms. If the voltage drops to 1.5 volts, the battery a notification will sound every 2 minutes until the voltage reaches 1.7 volts.

References

ada, l. (2018, Nov 3). *Adafruit Feather 32u4 Bluefruit LE*. Retrieved from Adafruit: https://learn.adafruit.com/adafruit-feather-32u4-bluefruit-le/overview

Ada, L. (2018, 05 19). *Adafruit Feather 32U4 Bluefruit LE*. Retrieved from Adafruit: https://learn.adafruit.com/adafruit-feather-32u4-bluefruit-le/installing-ble-library

Fruit, A. (2018, 05 19). *Github*. Retrieved from BluefruitLE Connect: https://github.com/adafruit/Bluefruit_LE_Connect_Android

Townsend, K. (2018, 11 12). *AT Commands*. Retrieved from Adafruit Feather 32u4 Bluefruit LE: https://learn.adafruit.com/adafruit-feather-32u4-bluefruit-le/at-commands

Appendix A

Kinderlert App Code Document KNDR001 V1

```
1  using System;
2  using System.IO;
3  using System.Text;
4  using Android.App;
5  using Android.Widget;
6  using Android.OS;
7  using Android.Bluetooth;
8  using Android.Bluetooth.LE;
9  using System.Linq;
10 using Android.Content;
11 using Android.Runtime;
12 using Android.Views;
13 using BluetoothLE.Core;
14 using BluetoothLE.Core.Events;
15 using static Android.Bluetooth.BluetoothAdapter;
16 using Java.Lang;
17 using Java.Util;
18 using System.Collections.Generic;
19 using System.Threading.Tasks;
20 using static Android.Bluetooth.BluetoothClass;
21 using Plugin.BLE.Android;
22 using BluetoothLE.Droid;
23 using Java.Util.Zip;
24 using Android.Media;
25
26 using Android.Util;
27 using Java.Nio.Charset;
28 using System.Net.Sockets;
29
30
31 namespace KinderLertApp
32 {
33     [Activity(Label = "KinderLertApp", MainLauncher = true)]
34
35
36     public class MainActivity : Activity
37     {
38
39
40
41

BluetoothGatt
gatt;
42
43
44          BluefruitLEManager myConnection = new BluefruitLEManager(); // Setup
```

```
            software link to the Bluefruit LE class
45  //BluetoothConnection myConnection2 = new BluetoothConnection();  46
47          //Bluetooth.Plugin.Android.BluetoothConnectionThread
              GetBluetoothConnection;
48
49
50          // private BleManagerListener mBleListener;

52          public static System.Guid TX_UUID = System.Guid.Parse("6E400002-B5A3-
              F393-E0A9-E50E24DCCA9E");
53          public static System.Guid RX_UUID = System.Guid.Parse("6E400003-B5A3-
              F393-E0A9-E50E24DCCA9E");
54
55          //int returnRssi = 0;    // future development for Retrieve RSSI
56
57          BluetoothSocket _socket = null;
58  private BluetoothDevice device;  59
60  //private static BleManager mInstance = null;  61
62  protected override void OnCreate(Bundle savedInstanceState)  63{
64
65  base.OnCreate(savedInstanceState);

66
67              // Set our view from the "main" layout resource
68              // Creates link to all buttons located in Main.AXML and allow
                  programming
69              // to the buttons.
70  SetContentView(Resource.Layout.Main);
71
72  ///     //Get  buttons from the layout resource,
73
74    Button connectButton = FindViewById<Button> (Resource.Id.connectBTButton1);
75  Button diconnectButton = FindViewById<Button>  (Resource.Id.DiscBTButton);
76  Button exitBTButton = FindViewById<Button>(Resource.Id.exitBTButton);
77  Button resetAlmButton = FindViewById <Button> (Resource.Id.resetAlmButton);
78  TextView connectedTextView = FindViewById<TextView>  Resource.Id.connectedTextView);
79  TextView mRssiEditText =     FindViewById<TextView> (Resource.Id.rssiTextView);
80  //     diconnectButton.Click += diconnectButton_Clicked;
81  int uid = 10489; //  Bluefruit 32U4 LE board UID
82
83
84
 85             connectButton.Click += async delegate
87                  /* Call BluetoothConnection class methods and start the discovery
                       of
88      Bluetooth devices. This will make the connection to the Adafruit
89      Feather  automatically.  */
90      Try
91       {
92       myConnection = new BluefruitLEManager();
93       // in .25 seconds
94       //await  Task.Delay(250);
```

```
95          await  myConnection.BeginScanningForDevices();  //
            connectedTextView.Visibility  = ViewStates.Visible;
            connectedTextView.Text = "Connected!"; // Let user know
            connection Made
96          diconnectButton.Enabled  = true;
97          connectButton.Enabled  = false;     // Disable Connect  Button
98
99          }
100         catch  (Java.Lang.Exception  deviceEX)
101         {
102         // can add message later
103         }
104         // found device, can stop discovery

105         myConnection.StopScanningForDevices();

106         diconnectButton.Click  += async  delegate
107         {
108         Try
109         {
110         myConnection  = new  BluefruitLEManager();
111         //        await        myConnection.BeginScanningForDevices();        //
            connectButton.Enabled = true; // if Connect is true  Continue
112         diconnectButton.Enabled  = false;
113
114         //  myConnection.CloseContextMenu();
115        //     BluetoothDevice  connectedDevice  =  gatt.Device;
116         //StartActivity(typeof(MainActivity));
117         //  connectedDevice.Dispose();
118         //StartActivity(typeof(MainActivity));
119         //Finish();
120         //Android.OS.Process.KillProcess(Android.OS.Process.MyPid  ());
121         //  BluefruitLEManager.Current.Dispose();  var
            device =
122         Plugin.BLE.CrossBluetoothLE.Current.Adapter.GetSystemConnecte
            dOrPairedDevices().FirstOrDefault(x => uid == uid); //  TX_UUID);
123         if (device != null)
124         {
             125        //  gatt.Disconnect();
126         StartActivity(typeof(MainActivity));
127         // Finish();
128         //await
129         Plugin.BLE.CrossBluetoothLE.Current.Adapter.DisconnectDeviceA
```

```
130         sync(device);
131         // connectedDevice.Dispose();
132         //  device= gatt.Device;
133         //   Wait(1000);
134         //    myConnection.DisconnectDevice(device);
135         //     _socket = null; connectedTextView.Text =
            "Disconnected!";
136
137         //this.ConnectedDevices[device].Disconnect();
138         // Set Socket to null
139         // TextView.Text = "Disconnected!";        // Let user know BT is
            Disconnected
140         BT is Disconnected
141         }
142         catch
143         {
144         ;
145         }
146         };
147         resetAlmButton.Click += delegate
148         {
149         // durationLength = 0;
150         // AlertTone();
            StartActivity(typeof(MainActivity));
151
152         };
153
154         exitBTButton.Click += delegate
155         {
156       // StartActivity(typeof(MainActivity));
            Finish();
157         // FinishAffinity();
            Android.OS.Process.KillProcess(Android.OS.Process.MyPid());
158
159
160         };
161
162
163         }
164         // Bluefruit LE Class to setup bluetooth LE connection
165         // Code pulled and modified from the Adafruit LE connect software
166
167         public class BluefruitLEManager : Activity,
```

```
168         BluetoothAdapter.ILeScanCallback
169         {
170         // event Handlers for the Event Arguments used in the Discovery,
            connection and Deisconnect calls
171             public event EventHandler ScanTimeoutElapsed = delegate { };
172             public event EventHandler<DeviceDiscoveredEventArgs>
                DeviceDiscovered
173             = delegate { };
174             public event EventHandler<DeviceConnectionEventArgs> DeviceConnected
175             = delegate { };
176             public event EventHandler<DeviceConnectionEventArgs>
                DeviceDisconnected = delegate { };
177             public event EventHandler<ServiceDiscoveredEventArgs>
                ServiceDiscovered = delegate { };
178             // event Handlers for the Event Arguments used in the Discovery,
179             connection and Deisconnect calls
180             public event EventHandler<DeviceDiscoveredEventArgs>
                DeviceDiscovered
181             = delegate { };
182             public event EventHandler<DeviceConnectionEventArgs> DeviceConnected
183             = delegate { };
184             public event EventHandler<DeviceConnectionEventArgs>
                DeviceDisconnected = delegate { };
185             public event EventHandler<ServiceDiscoveredEventArgs>
                ServiceDiscovered = delegate { };
186             // Setup declarations used in Bluefruit class
187             protected BluetoothManager _manager;
188             protected BluetoothAdapter _adapter;
189             protected GattCallback _gattCallback;
190             public string myfeatherDevice;
/// <summary>
/// Whether or not we're currently scanning for peripheral devices
/// </summary>
/// <value><c>true</c> if this instance is scanning; otherwise,
191                     <c>false</c>.</value>
192             public bool IsScanning
193             {
194             get { return this._isScanning; }
195             }
196             protected bool _isScanning = false;
197             protected const int _scanTimeout = 10000;
/// <summary>
/// Gets the discovered peripherals.
```

```
/// </summary>
198                 /// <value>The discovered peripherals.</value>
199          public List<BluetoothDevice> DiscoveredDevices
200                 {
 201                 get { return this._discoveredDevices; }
202                 }
203          public List<BluetoothDevice> _discoveredDevices = new
204          List<BluetoothDevice>();
205                 /// <summary>
206          /// Gets the connected peripherals.
207          /// TODO: in the xplat API, make sure to combine the GATT into a
             single
208          /// IDevice object so it isn't necessary to create a dictionary
             to track them.
209          /// </summary>
210          /// <value>The discovered peripherals.</value>
211          {
212          get { return this._connectedDevices; }
213          }
214          protected Dictionary<BluetoothDevice, BluetoothGatt>
215          _connectedDevices = new Dictionary<BluetoothDevice,
             BluetoothGatt> ();
216
217          /// <summary>
218
219
220          /// Gets the services.
221          /// </summary>
222          /// <value>The services.</value>
223          public Dictionary<BluetoothDevice, IList<BluetoothGattService>>
             Services
224          {
225          get { return this._services; }
226          }
227          protected Dictionary<BluetoothDevice, IList<BluetoothGattService>>
228          _services = new Dictionary<BluetoothDevice,
             IList<BluetoothGattService>>();
229          //  /// <summary>
230          //        /// Need to have this because the google BLE API is
             terrible.  in it, we cache the device's
231          //  /// GATT when we call Connect, so that later, we can add it
             to _connectedDevices, because
232          //  /// we're not given a reference to the device when it
```

```
            connects
233         //  /// </summary>
234         /   protected Dictionary<BluetoothDevice, BluetoothGatt>
235         _connectingDevices = new Dictionary<BluetoothDevice,
            BluetoothGatt> ();
236         public static BluefruitLEManager Current
237         {
238         get { return current; }
239         }
240
241         public BluetoothDevice device { get; private set; }
242        public Func<string, BluetoothDevice> connectedDevice { get;
243        private set; }
244
245                        BluefruitLEManager current;
246                static BluefruitLEManager()
247        {
248        current = new BluefruitLEManager();
249        }
250
251        public BluefruitLEManager()
252        {
253        var appContext = Android.App.Application.Context;
254        // get a reference to the bluetooth system service
255        this._manager = (BluetoothManager)appContext.GetSystemService
256        ("bluetooth");
257         this ._adapter = this._manager.Adapter;
  258                        this._gattCallback = new GattCallback(this);
259        }
260        // Scan for any bluetooth devices
261        public async Task BeginScanningForDevices()
262        {
263        Console.WriteLine("BluefruitLEManager: Starting a scan for
           devices.");
264
265        // clear out the list
                         this                  ._discoveredDevices
           = new List<BluetoothDevice>();
266
267        // start scanning
                          this ._isScanning = true;
268        _adapter.StartLeScan(this);
269
```

```
270
271         // in 10 seconds, stop the scan
            await Task.Delay(10000);
272
273         // if we're still scanning
            if (this._isScanning)
274         {
275         Console.WriteLine("BluefruitLEManager: Scan timeout has
            elapsed.");
276                     _adapter.StopLeScan(this);
277                             this.ScanTimeoutElapsed(this, new EventArgs());
278                 }
279                 }
280                 /// <summary>
281          /// Stops the Central Bluetooth Manager from scanning for more
            devices. Automatically
282          /// called after 10 seconds to prevent battery drain.
283          /// </summary>
284          public void StopScanningForDevices()
285          {
286        Console.WriteLine("BluefruitLEManager: Stopping the scan for devices.");
287             this._isScanning = false;
288             this._adapter.StopLeScan(this);
289
290                 }
291
292        public void OnLeScan(BluetoothDevice device, int rssi, byte[] scanRecord)
293         {
294         Console.WriteLine("LeScanCallback: " + device.Name);
295         // TODO: for some reason, this doesn't work, even though they
            have the same pointer,
296         // it thinks that the item doesn't exist. so i had to write my
            own implementation
297          //      if(!this._discoveredDevices.Contains(device) ) {
298              //  this._discoveredDevices.Add (device );
299                 // }
300           if (!DeviceExistsInDiscoveredList(device))
301           ._  .discoveredDevices.Add(device);
302           // TODO: in the cross platform API, cache the RSSI
              this .DeviceDiscovered(this, new
              DeviceDiscoveredEventArgs
303           { Device = device, Rssi = rssi, ScanRecord = scanRecord });
              if (DeviceExistsInDiscoveredList(device))
```

```
304          {
305                  //  BluefruitLEManager.Current.ConnectToDevice(device);
306       //         Current.ConnectToDevice(device);
307          if (device.Name == "Adafruit Bluefruit LE")
308          {
309          myfeatherDevice = device.Name;
310
311                      BluefruitLEManager.Current.ConnectToDevice(device);
312          }
313          else
314          {
315          ;
316          }
317          }
318          else
319          {
320          ;
321          }
322          }
323          protected bool DeviceExistsInDiscoveredList(BluetoothDevice device)
324          {
325          foreach (var d in this._discoveredDevices)
326      {// TODO: verify that address is unique if (device.Address ==d.Address)
327                  return true;
328      }
329      return false;
330          }
331          //TODO: this really should be async. in the xplat API, make sure to
             asyncify
332          // Q: how to return in same context (requires a
             callback) public void ConnectToDevice(BluetoothDevice
             device)
333          {
334          // returns the BluetoothGatt, which is the API for BLE stuff
335          // TERRIBLE API design on the part of google here.
             device.ConnectGatt( Android.App.Application.Context, true,
336          this._gattCallback);
337       }
338          public void DisconnectDevice(BluetoothDevice device)
339          {
340          //device.Name = "Adafruit Bluefruit LE";
341                          this.ConnectedDevices[device].Disconnect();
342                          this .ConnectedDevices[device].Close();
```

```
343         }
344     public BluetoothDevice GetConnectedDeviceByName(string deviceName)
345       {
346       foreach (var item in this._connectedDevices)
347       {
348
349       }
350       // if we got here we didn't find it. return null;
351       }
352       public class DeviceDiscoveredEventArgs : EventArgs
353       {
354       public BluetoothDevice Device;
355       public int Rssi;
356       public byte[] ScanRecord;
357
358       public DeviceDiscoveredEventArgs() : base()
359       { }
360       }
361
362       public class DeviceConnectionEventArgs : EventArgs
363       {
364       public BluetoothDevice Device;
365
366       public DeviceConnectionEventArgs() : base()
367       { }
368       public class ServiceDiscoveredEventArgs : EventArgs
369       {
370       public BluetoothGatt Gatt;
371
372       public ServiceDiscoveredEventArgs() : base()
373       { }
374       }
375
376       protected class GattCallback : BluetoothGattCallback
377       {
378       protected BluefruitLEManager _parent; int durationLength = 0;
379
380
381
382       public GattCallback(BluefruitLEManager parent)
383       {
384                           this._parent = parent;
385       }
```

```
386         public override void OnConnectionStateChange(BluetoothGatt gatt,
            GattStatus status, ProfileState newState)
387         {
388         Console.WriteLine("OnConnectionStateChange:  ");
389         base.OnConnectionStateChange(gatt, status, newState);
390
391
392
393         switch (newState)
394         {
395         // disconnected
396         case ProfileState.Disconnected: Console.WriteLine("disconnected");
397         //TODO/BUG: Need to remove this, but can't remove the key
            (uncomment and see bug on disconnect)
398         //   if
399         (this._parent._connectedDevices.ContainsKey  (gatt.Device))
400         //
401         this._parent._connectedDevices.Remove  (gatt.Device); this
            ._parent.DeviceDisconnected(this, new
402         DeviceConnectionEventArgs() { Device = gatt.Device });
            durationLength = 100;
403         AlertTone(); break;
404         // connecting
405          case ProfileState.Connecting:
            Console.WriteLine("Connecting");
406          break;
407          // connected
408          case ProfileState.Connected:
409          // connectedTextView.Text = "Connected!"; // Let user know
            connection Made
410          //TODO/BUGBUG: need to remove this when disconnected
411       // _parent._connectedDevices.Add(gatt.Device, gatt);     //
            removed this.
412          _parent.DeviceConnected(this, new DeviceConnectionEventArgs()
            { Device = gatt.Device });
413
414
415          break;
416          // disconnecting
417          case ProfileState.Disconnecting:
            Console.WriteLine("Disconnecting"); break;
418          }
419
```

```
420         }
421         public override void OnServicesDiscovered(BluetoothGatt gatt,
            GattStatus status)
422         {
423         base.OnServicesDiscovered(gatt, status);
424
425         Console.WriteLine("OnServicesDiscovered: " + status.ToString ());
426
427         //TODO: somehow, we need to tie this directly to the device, rather
            than for all
428         // google's API deisgners are children.
429
430         //TODO: not sure if this gets called after all services have been
            enumerated or not
431         if (!this._parent._services.ContainsKey(gatt.Device))
432          this ._parent.Services.Add(gatt.Device,
433         this._parent._connectedDevices[gatt.Device].Services);
434         Else
435         ._parent.this.vices[gatt.Device] =
436         this._parent._connectedDevices[gatt.Device].Services;
437         this._parent.ServiceDiscovered(this,
438         new ServiceDiscoveredEventArgs()
439         {
440         Gatt = gatt
441         });
442                 }
443         //  Setup audio alert for phone to be called when device looses
            bluetooth connection.
444         private void AlertTone()
445         {
446         // for (int c = durationLength; c > 0; c--)
447         {
448         var duration = durationLength; var
            sampleRate = 8000;
449         var numSamples = duration * sampleRate; var
            sample = new double[numSamples]; var freqOfTone
            = 1900;
450         byte[] generatedSnd = new byte[2 * numSamples];
451
452         for (int i = 0; i < numSamples; ++i)
453         {
454         {
```

```
455         sample[i] = System.Math.Sin(2 * System.Math.PI * i / (sampleRate /
            freqOfTone));
456         }
457
458         int idx = 0;
459         foreach (double dVal in sample)
460         {
461         short val = (short)(dVal * 32767);
462         generatedSnd[idx++] = (byte)(val & 0x00ff);
463         generatedSnd[idx++] = (byte)((val & 0xff00) >> 8);
464         }
465       var track = new AudioTrack
466                 (global::Android.Media.Stream.Music,  sampleRate,
467                 ChannelConfiguration.Default,
468                 Android.Media.Encoding.Default, numSamples,
469                 AudioTrackMode.Static);
470          track.Write(generatedSnd, 0, numSamples);
471          track.Play();
472          }
473          }
474          }
475          }
476          }
477
```

Appendix B

```
/*************************************************************************
This is an example for our nRF51822 based Bluefruit LE modules

Pick one up today in the adafruit shop!

Adafruit invests time and resources providing this open source code, please support
Adafruit and open-source hardware by purchasing products from Adafruit!

MIT license, check LICENSE for more information
All text above, and the splash screen below must be included in
any redistribution
*************************************************************************/
//.Kinderlert Device Code
// Reference KNDRDEV ver 4
// By John P Biehle
//11/27/2018
#include <Arduino.h>
#include <SPI.h>
#include "Adafruit_BLE.h"
#include "Adafruit_BluefruitLE_SPI.h"
#include "Adafruit_BluefruitLE_UART.h"

#include "BluefruitConfig.h"

#if SOFTWARE_SERIAL_AVAILABLE
#include <SoftwareSerial.h>
#endif

    /*=========================================================================
    APPLICATION SETTINGS
```

```
33                                      ? ? FACTORYRESET_ENABLE? ?  Perform a factory reset when
                                        running this sketch
34                                      ? ?
35                                      ? ?                 Enabling this will put your Bluefruit LE module
36                                      in a 'known good' state and clear any config
37                                      data set in previous sketches or projects, so 38 ? ?
                                        running this at least once is a good idea.
39                                      ? ?
40                                      ? ?                 When deploying your project, however, you will
41                                      want to disable factory reset by setting this
42                                      value to 0.? If you are making changes to your 43 ? ?
                                        Bluefruit LE device via AT commands, and those
44                                      changes aren't persisting across resets, this
is the reason why.? Factory reset will erase 46                 the non-volatile memory where config data is
47                      values.
? ? ? ?
? ? Some sketches that require you to bond to a
central device (HID mouse, keyboard, etc.)
won't work at all with this feature enabled
45                                      stored, setting it back to factory default 48
                                        values.
49                                      ? ? ? ?
50                                      ? ?                 Some sketches that require you to bond to a
51                                      central device (HID mouse, keyboard, etc.)
52                                      won't work at all with this feature enabled
53                                      since the factory reset will clear all of the 54
                                        bonding data stored on the chip, meaning the 55
                                        central device won't be able to reconnect.
56   ------------------------------------------------------------------------*/ 57   #define FACTORYRESET_ENABLE     0
58 /*=========================================================================*/
59


60


61   // Create the bluefruit object, either software serial...uncomment these lines
62   /*
```

```
63  SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN,
    BLUEFRUIT_SWUART_RXD_PIN);
64

    Adafruit_BluefruitLE_UART ble(bluefruitSS, BLUEFRUIT_UART_MODE_PIN,
    BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);
    */
68

69  /* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line */
70  // Adafruit_BluefruitLE_UART ble(BLUEFRUIT_HWSERIAL_NAME, BLUEFRUIT_UART_MODE_PIN);
71

72  /* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected CS/IRQ/RST */
73  Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
74

75  /* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user selected CS/IRQ/RST */
76  //Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
77  //                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
78  //                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
79


80


81      // A small helper
82      void error(const __FlashStringHelper*err) {
83      Serial.println(err);
84      while (1);
85      }
86


87        /*
88        Declarations for initial setup and pin setup
89        */
90        #define VBATPIN A7
91        #define AudioOutPIN A5
92


93    int  rssiDataReceived = 0;
94    int connectedState= 0;
```

```
 95    int initialConnection = 0;
 96    int initialBatCheck = 0;
 97    float measuredvbat = analogRead(VBATPIN);
 98    int batCount=0;
99

100             //conduct an initial battery check
101             void CheckBattery()
102             {
103             if (measuredvbat <= 1.5)  // if battery less than 1.5 vols provide a 500 hz tone for 5 counts.
104             {
105             if(initialBatCheck  == 0)
106

107                 for (int i = 0; i <= 5; i++)
108                 {
109                 tone(13, 500);;
110                 delay(100);
111                 noTone(13);
112                 delay(100);
113                 initialBatCheck = 1;
114                 }
115

116                     else if(initialBatCheck  == 1)
117                     {
118                     for (int i = 0; i <= 5; i++)
119                     {
120                     delay(1000);
121                     initialBatCheck = 0;
122                     }
123                     }
124                     }}
125

126 void setup() {
127

128       {
129       // while (!Serial)
```

```
130       {
131       'AT+BLEPOWERLEVEL=-4';  // Setup bluetooth power level
132
133       // required for Flora & Micro
134       delay(500);
135
136      // Serial.begin(115200);
137      Serial.println(F("Adafruit Bluefruit AT Command Example"));
138      Serial.println(F("-------------------------------------"));
139
140      /* Initialise the module */
141      Serial.print(F("Initialising the Bluefruit LE module: "));
142
143        if ( !ble.begin(VERBOSE_MODE) )
144        {
145        error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check wiring?"));
146        }
          147    if ( FACTORYRESET_ENABLE )
          148     {
          149       /* Perform a factory reset to make sure everything is in a known state */
          150       Serial.println(F("Performing a factory reset: ")); if ( !
          ble.factoryReset() ){
          151         error(F("Couldn't factory reset"));
          152       }
          153     }
          154     /* Disable command echo from Bluefruit */   ble.echo(false);
          155     Serial.println("Requesting Bluefruit info:");
          156     /* Print Bluefruit information */   ble.info();
```

```
157
}}

158   /**************************************************************************
//*!
159       @brief  Constantly poll for new command or response data

160   */

161   /**************************************************************************

/

162     // startup Routine to check for Device


163   float AudioOut = AudioOutPIN, pinMode(13, OUTPUT); //Setup Pin 13 for audio output
164     // Sets up so battery outputs votlage when battery check is called.

165     measuredvbat *= 2;    // we divided by 2, so multiply back

166     measuredvbat *= 3.3;  // Multiply by 3.3V, our reference voltage

167     measuredvbat /= 1024; // convert to voltage   //Print out
battery voltage to serial port.
168     Serial.println("VBat: "); Serial.println(measuredvbat);

169     Serial.println("RESTARTED");

170       //Query RSSI and output to serial port

171     'AT+BLEGETRSSI';

172       Serial.print(F("AT+BLEGETRSSI"));

173       'AT+GAPGETCONN';  // check to see if bluetooth is connected

174       Serial.print(F("AT+GAPGETCONN")); // Print if connected to serial port
```

```
175     // Output Battery Level if(
initialBatCheck = 0)
176     { int i
= 0;
177    if (measuredvbat >=3.5)    // if voltage is greater than 3.5


 178   Serial.println( F("OK!") );
179          { for (int i = 0; i = 1; i++)
180          {
181          Serial.println("VBAT");
182          tone(13, 10000);;
183          delay(1000);
184          noTone(13); ;
185          delay(1000);
  208  }

  209

186          }
187          else if (measuredvbat >= 3.2)  // if voltage is greater than 3.2
188          { for (int i = 0; i = 2; i++)
189          {
190          Serial.println("32 ");
191          AudioOut = HIGH;
192          digitalWrite(13, 10000);
193          delay(1000);
194          AudioOut = LOW;
195          digitalWrite(13, LOW);
196          delay(1000);
  221

  222  }

  223

197          }
   224          else if (measuredvbat >=2.5)   // if voltage is greater than 2.5
   225          { for (int i = 0; i = 4; i++)
   226          {
   227          Serial.println("29");
```

```
228           AudioOut = HIGH;
229           digitalWrite(13, 10000);
230           delay(1000);
231           AudioOut = LOW;
232           digitalWrite(13, LOW);
233           delay(1000);
234           }
236

237  }

238

239           else if (measuredvbat <= 1.8)   // if voltage is less than 1.8
240           { for (int i = 0; i = 5; i++)
241           {
242           Serial.println("28");
243           AudioOut = HIGH;
244           digitalWrite(13, 10000);
245           delay(1000);
246           AudioOut = LOW;
247           digitalWrite(13, LOW);
248           delay(1000);
249           }


250            initialBatCheck = 1;

251


                }

                }


252           void loop()

253           {


254           float AudioOut = AudioOutPIN, pinMode(13, OUTPUT); //Setup Pint 13 for audio output
```

```
255             Serial.println("loop");
256           // while(Serial)
257             {
258   while(!ble.isConnected())  //  If bluetooth is not connected for initial connection sub into
      routine.
259           {
260             initialConnection=1;
261             Serial.println("initial Connection");
262           }
263   int reconnect = 0;
264   while(ble.isConnected()&& (initialConnection == 2))   //if Bluetooth is reconnected after loosing
      initialconnection
265           {
266             Serial.println("initialConnection " + initialConnection);        initialBatCheck = 1;
267             CheckBattery();
      for(inti<=3;i++)
268              {
269             reconnect = 0;          'AT+BLEGETRSSI'; if ( 0< 'AT+BLEGETRSSI' >= 5)     //if RSSI is between
      0 and 5 set tone to 4KHZ
270   'AT+BLEGETRSSI';
271   if ( 0< 'AT+BLEGETRSSI' >= 5) //if RSSI is between 0 and 5 set tone to 4KHZ
272   {
273   Serial.println("BLEGETRSSI10");
274   Serial.print(("AT+BLEGETRSSI" ,'AT+BLEGETRSSI'));
275   Serial.print("\n");
276   Serial.print("VBAT ");
277   Serial.println(measuredvbat);
278   Serial.print("\n");
279   tone(13, 4000);;
280    }
```

```
281  else if ( 5< 'AT+BLEGETRSSI' >= 10) //if RSSI is between 5
282  {
283  Serial.println("BLEGETRSSI20");
284  Serial.println("BLEGETRSSI20");
285  Serial.print(("AT+BLEGETRSSI" ,'AT+BLEGETRSSI' ));
286  Serial.println("reconnected");
287  Serial.print(3,"\n");
288  tone(13, 1000);;
289  delay(200);
290  tone(13, 5000);;
291  delay(200);
292  }
293  else if ( 10< 'AT+BLEGETRSSI' >= 20) //if RSSI is between
294  10 and 20 alternate tone between 1K and 7 KHZ
295  {
296  Serial.println("BLEGETRSSI30");
297  Serial.print(("AT+BLEGETRSSI" ,'AT+BLEGETRSSI' ));
298  Serial.println("reconnected");
299  Serial.print(3,"\n");
300  tone(13, 1000);;
301  delay(200);
302  tone(13, 7000);;
303  delay(200);
304  }
305  else if ( 20< 'AT+BLEGETRSSI' >= 30) //if RSSI is between
306  20 and 30 alternate tone between 1K and 8 KHZ
307  {
308  Serial.println("BLEGETRSSI40");
309  Serial.print(("AT+BLEGETRSSI" ,'AT+BLEGETRSSI' ));
310  Serial.println("reconnected");
311  Serial.print(3,"\n");
312  tone(13, 1000);;
313  delay(200);
314  tone(13, 8000);;
315  // noTone(13); ;
316  delay(200);
317  }
318  else if( 30<'AT+BLEGETRSSI'>=40) //if RSSI is between 30 and
319  40 alternate tone between 1K and 10 KHZ
320  {
321  Serial.println("BLEGETRSSI50");
```

```
322  Serial.print(("BLEGETRSSI" ,'AT+BLEGETRSSI'));
323  Serial.print(3,"\n");
324  Serial.print("VBAT ");
325  Serial.println(measuredvbat);
326  Serial.print(3,"\n");
327  tone(13, 1000);;
328  delay(200);
329  tone(13, 10000);
330  delay(200);
331  }
332  else if( 40<'AT+BLEGETRSSI') //if RSSI is between 30 and 40
333  alternate tone between 4K and 10 KHZ
334  {
335  Serial.println("BLEGETRSSI50");
336  Serial.print(("BLEGETRSSI" ,'AT+BLEGETRSSI'));
337  Serial.print(3,"\n");
338  Serial.print("VBAT ");
339  Serial.println(measuredvbat);
340  Serial.print(3,"\n");
341  tone(13, 4000);;
342  delay(200);
343  tone(13, 10000);
344  delay(200);
345  }
346  }
347  }
348  while(!ble.isConnected()) // If Bluetooth looses connection alternate
349  tone between 1k and 4 KHZ
350  {
351  Serial.print(4,"\n");
352  Serial.println("not connected");
353  tone(13, 1000);;
354  delay(100);
355  tone(13, 4000);;
356  delay(100);
357  initialConnection = 2;
358  }
359  }
360  }
361
```

```
362  static void batteryVoltageCheck()
363  {
364  float measuredvbat = analogRead(VBATPIN);
365  // float AudioOut = AudioOutPIN, pinMode(13, OUTPUT);
366  //pinMode(13, OUTPUT);
367  measuredvbat *= 2; // we divided by 2, so multiply back
368  measuredvbat *= 3.3; // Multiply by 3.3V, our reference voltage
369  measuredvbat /= 1024; // convert to voltage
370  Serial.print("VBat: ");
371  Serial.println(measuredvbat);
372  return measuredvbat;
373  }

374

375

376  }
```