SOUND NRF TOOLBOX PROJECT MANUAL

Clifton Zhuo

6538460 C.k.t.zhuo@students.uu.nl

Table of Contents

1. Introduction

Sound nRF is an extended version of the nRF Toolbox which can be found at https://github.com/NordicSemiconductor/Android-nRF-Toolbox.
This manual provides the code explanation how to add sound synchronization with cadence data to the nRF Toolbox app. It also shows how to enable/re-enable sound for 2-minutes when the presented cadence deviates by 15.

2. Requirements.

- An Android IDE for Android operating system such as Android Studio (https://developer.android.com/studio)
- An Android Mobile Smartphone. Sound nRF Toolbox is currently compatible to android devices. For assurance, android 10 update is ideal.
- A Sensor device with BLE supported that provides RSC (Running speed and cadence) data. e.g. Stryd

3. Bluetooth Low Energy

BLE Communication

Sound nRF toolbox connects to the Stryd through BLE (Bluetooth Low Energy) technology. A BLE device can act as a central or peripheral role, sometimes referred to as client or server, respectively. The central (client) role initiates requests and accepts data responses, while the peripheral receives requests and return responses. In this project, the Sound nRF toolbox app will act as the central device, and the Stryd foot pod will act as the peripheral device. BLE

uses a hierarchical data structure to define the information exchange structure.

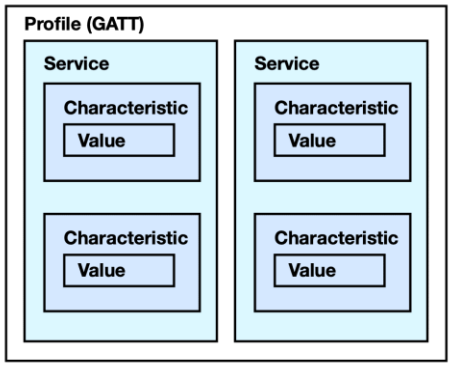

Figure 1 – BLE hierarchica data structure

GATT (Generic Attribute Profile) describes in detail how attributes (data) are transferred when once the devices have a dedicated connection. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data. A Service is a collection of characteristics. Characteristics are parts of the service that represents a piece of information/data that a

server wants to expose to a client. For example, you could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement."[1]

NRF TOOLBOX

The nRF Toolbox is an application developed by Nordic Semiconductor. It works with a wide range of the most popular Bluetooth LE accessories. It contains applications demonstrating the following profiles: Cycling Speed and Cadence, Running Speed and Cadence, Heart Rate Monitor, Blood Pressure Monitor, Health Thermometer Monitor, Glucose Monitor, Proximity Monitor and Nordic UART.[2]

To connect to Stryd with nRF toolbox:

Step 1 – Open nRF Toolbox app

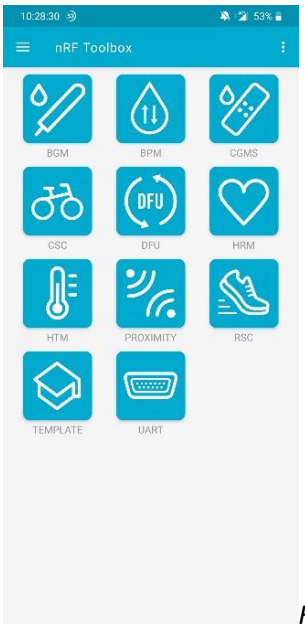

Figure 2 – Start Menu of nRF Toolbox

Step 2 – Choose designated service

[1] "Bluetooth Low Energy Overview: Android Developers," Android Developers, accessed August 16, 2020, https://developer.android.com/guide/topics/connectivity/bluetooth-le.

[2] "NRF Toolbox App," Nordic Semiconductor, accessed August 16, 2020, https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Toolbox.

Figure 3 – RSC Display

Step 3 – Tap on *connect* after turning on Bluetooth

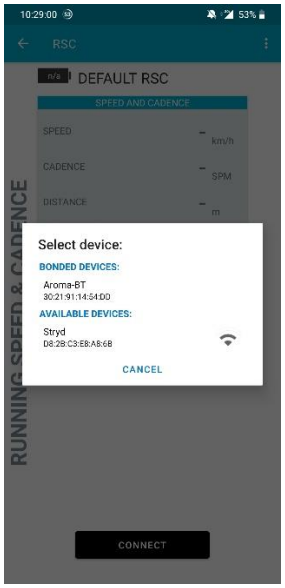

Figure 3 – Select Device display

All the devices within the proximity that advertises RSC data will be listed. Select the desired device e.g. Stryd.

Step 4 – Generate data in real time.

Figure 4 – RSC data detected

4. Sound nRF ToolBox Path

Since we are only working with RSC activities, the only java file we need to focus on is RSCActivity, which is located in:

*Users\username\...\Android-nRF-Toolbox-master\app\src\main\java\no\nordicsemi\android\nrftoolbox\rsc**RSCActivity***

The code is simple once the BLE communication is understood.
There are two main section that were implemented to add sound to the nRF toolbox.

1. Provide Auditory feedback by using ToneGenerator.

2. Enable sound for 2-minutes. And re-enable when cadence deviates by 15.

5. Adding Sound to nRF ToolBox

```
import android.media.AudioManager;

import android.media.ToneGenerator;
import android.os.Bundle;
import android.os.Handler;
import android.preference.PreferenceManager;
```

Before we can play any sound, we first need to import android.media.Audiomanager, android.media.ToneGenerator, and android.media.os.Handler.

```
private int cad=5; //variable to store cadence value
private int play_media=0; //flag to control media playing

private long per; //variable to store value of period
ToneGenerator toneG = new ToneGenerator(AudioManager.STREAM_MUSIC, 100); // tone generator object

private final Handler mHandler = new Handler(); // handler to schedule runnable task

private final Runnable mTask = new Runnable() {

       @Override
       public void run() {

                      if(play_media==1)
                      {
                             toneG.startTone(ToneGenerator.TONE_PROP_BEEP, 25);//generate beep for 25 milliseconds. It can be changed.
                      }
                      //Log.i("This is my cadence",String.valueOf(cad));
                      //Log.i("This is my period",String.valueOf(per));
              mHandler.postDelayed(mTask, per);
       }
};
```

We use *ToneGenerator.[Sound_NAME], 25* to generate the sound. The ToneGenerator sounds can be found at https://developer.android.com/reference/android/media/ToneGenerator.
Most sounds' duration are too long to sync. For this reason, we only play the first 25 millisecond of the TONE_PROP_BEEP sound. We will use the Handler to determine how frequent we have to play the sound depending the *per* variable.

Double the cadence, and broadcast receiver used for period calculation.

```
//this is broadcast receiver used for period calculation
private final BroadcastReceiver broadcastReceiver_for_media = new BroadcastReceiver() {
        @Override
        public void onReceive(final Context context, final Intent intent) {
                final String action = intent.getAction();

                if (RSCService.BROADCAST_RSC_MEASUREMENT.equals(action)) {
                        final int cadence = intent.getIntExtra(RSCService.EXTRA_CADENCE, 0);
                        cad=cadence*2;
                        if(cad>10)
                        {
                                float fraction;
                                fraction=60f/cad;
                                per= (long) (fraction*1000L);
                                play_media=1;
                        }
                        else
                        {
                                per=1000L;
                                play_media=0;
                        }
                }
        }
};
```

we multiply intake by 2 because the original nRF Toolbox calculates the RPM, not SPM despite that it displays SPM. We tested this with other apps which can be seen in the project report.

We use a condition method If the captured cadence data is higher than 10, we play the sound. To calculate the how frequent we need to play the sound, we do 60/cadence. After that we need to convert milliseconds into seconds. Thus, we multiply the fraction by 1000.

Finally, add :
LocalBroadcastManager.getInstance(this).registerReceiver(broadcastReceiver_for_media, makeIntentFilter());

mHandler.postDelayed(mTask, 1000); in the OnInitialized method, to start once the app starts.

And add :
LocalBroadcastManager.getInstance(this).unregisterReceiver(broadcastReceiver_for_media);
mHandler.removeCallbacks(mTask);
play_media=0; in the onDestroy() method to stop the callbacks when app closes.

6. Enable/Re-enable 2-minute Feedback

We enable sound for 2-minutes. As long sound is enabled, we do not need to check cadence threshold. Thus, variable *workOnCad = false*. The code measures in milliseconds, for (2*60*1000) = 120000 milliseconds which is equivalent to 2-minutes.

```
if (isFirstTimeRun) {       //checking variable named "isFirstTimeRun" is true if True then it will enter into if loop
    isFirstTimeRun = false; // Changing "isFirstTimeRun" to False when it runs for the first time
    if (workOnCad) {        //checking variable named "isFirstTimeRun" is true if True then it will enter into if loop
        workOnCad = false;  // Changing "workOnCad" to False once cadence start to increase
```

```
        Handler handler = new Handler();    // Initializing handler object
        play_media = 1;                     //enable sound
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                workOnCad = true;           // Changing "workOnCad" to True after 2 minutes beep sound

                maxCad = cad;               // Saving cadence at the end of 2 minutes in variable named "maxCad"
            }
        }, 2 * 60 * 1000);
    }
```

The variable “workOnCad” checks if the present cadence deviates by 15. At the end of the 2-minute, we save the presented “new” cadence to variable “maxCad”

```
    if (workOnCad) {
        if (maxCad - cad > 15 || maxCad - cad < -14) { //Checking both upper and lower difference of 15
            workOnCad = false;
```

```
            Handler handler = new Handler();
            play_media = 1;
            handler.postDelayed(new Runnable() {
                @Override
                public void run() {
                    workOnCad = true;
                    maxCad = cad;
                }
            }, 2 * 60 * 1000);
```

This is checked every second. When the cadence stabilizes at the end of the 2-minute mark or reaches 0, sound stays disabled.

```
            } else {
                per = 1000L;
                play_media = 0;     //Disable beep Sound
            }

        }
    }
```

Reference

"Bluetooth Low Energy Overview: Android Developers." Android Developers. Accessed August 16, 2020. https://developer.android.com/guide/topics/connectivity/bluetooth-le.

"NRF Toolbox App." Nordic Semiconductor. Accessed August 16, 2020. https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Toolbox.