Sound Ecology and Acoustic Health, Part 2
An Android Application for Recording Noise Nuisances

Published in Circuit Cellar, Kick Media Corporation, U. S. A,
Issue 301, pp 16 – 33, 2015 -- http://circuitcellar.com/

Adrien Gaspard and Mike Smith

Contact: Mike Smith:

Department of Electrical and Computer Engineering,
University of Calgary,
ICT536, 2500 University Drive, N.W.
Calgary, Alberta, Canada T2N 1N4

Email: Mike.Smith @ ucalgary.ca

Phone: +1-403-220-6142

Sound Ecology and Acoustic Health, Part 2
An Android Application for Recording Noise Nuisances

Being efficient

Extending our basic Android WAT_AN_APP built

Without Any Teenager Assistance Necessary

with

"Just Enough Additional Code"

to Record and Playback and Neighbourhood Acoustic Nuisances

In our last month's article, CC Issue #, we light-heartily discussed a supposed back yard BBQ discussion between neighbours about urban noise nuisances. Unfortunately noise nuisances are real in some of our local Calgary Communities, and we are looking for some simple, inexpensive approaches to help people investigate and reduce the problem.

We demonstrated the first steps of our solution - the development of an Android project with basic code to generate a main screen with a button that generated a welcome screen when pressed. We called this a WAT_AN_APP, meaning we were able to develop it *Without Any Teenager Assistance being Necessary*. In this article we want to extend our basic WAT_AN_APP project to recording and playing-back audio .3GPP files as shown in Fig. 1A. This will allow us to record any physical noises present that are less easily heard by others in your house or need more study as they are less noticeable during the day as they are hidden under traffic noise.

In this article, we want to take a more adult approach – use a JEAC process that uses *Just Enough Additional Code* to make the new recording activity work.

```
1.    package com.wat_an_app;  // MainActivity.java
2.    ...   // SAME AS Article 1, Listing 1, Lines 2 to 5

      // Cause display of MainActivity screen layout
10.   public class MainActivity extends Activity{
11.       ...   // SAME AS Article 1, Listing 1, Lines 11 to 15

          //  Display AudioRecordPlayback screen layout
20.       public void AudioRecordPlayback(View v){
21.         ...   // SAME AS Article 1, Listing 1, Lines 21 to 24
```

Listing 1A) Article 1 key code from *MainActivity.java in* the *WAT_AN_APP\src\* folder

```
      <!--Used by AudioRecordPlayback -->
400. <RelativeLayout
401. xmlns:android=http://schemas.android.com/apk/res/android
402. ... <!--  SAME AS Article 1, Listing 5, Lines 402 to 405 -->

410.     <TextView
411.       android:id="@+id/audio_record_playback_text"
412.       ...<-- SAME AS Article 1 Listing 5, Lines 410 to 417-->

499. </RelativeLayout>
```

Listing 1B) Article 1 *AudioRecordPlayback* activity layout from *activity_audio_record.playback.xml* file in the *WAT_AN_APP\res\layout* folder

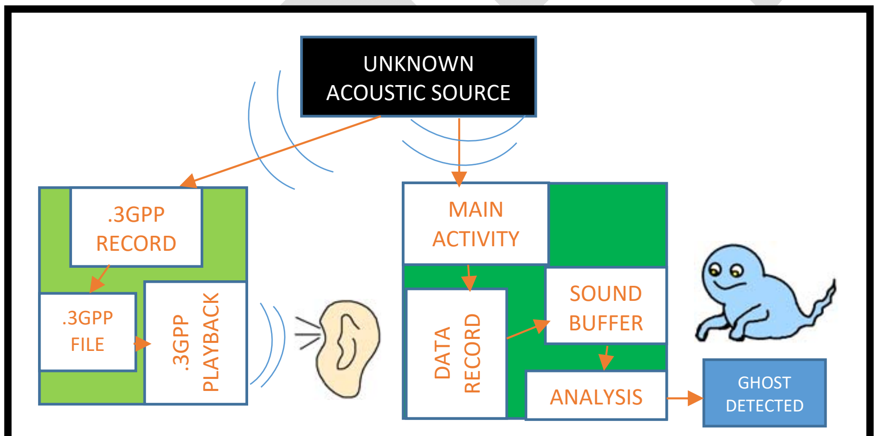

Fig 1) Our Android WAT_AN_APP for A) Audio .3GPP Record/Playback B) Data Record/Ghost Analysis

QUICK RECAP

Listing 1A provides the key elements of the main activity java file. Applying the JEAC philosophy, we added enough code to pop up a screen with a welcome message and a button. Pressing the button activated the audio record and play-back "*AudioRecordPlayback*" activity (Line 20). This activity used the layout described in the *activity_audio_record_playback.xml* file to activate a *TextView* object to print a message "DUMMY NEW ACTIVITY SCREEN" (Listing 1B). Please note that the main activity's layout file, *activity_main.xml*, is identical to the code described in Article 1 Listing 2.

In this article we are going to extent this dummy activity so that we can record and playback audio .3GPP files, the first step towards doing some real signal processing on audio signals.

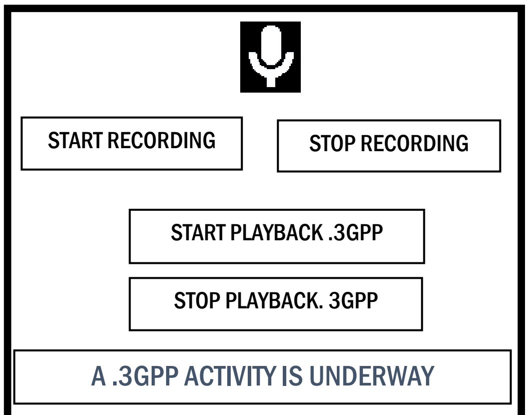

Fig 2) Our planned.3GPP file control screen

JEAC BUTTONS FOR .3GPP CONTROL

The new *activity_audio_record_playback.xml* layout, Listing 2, shows how we can use a lot of our knowledge gained from the previous article to generate an audio menu screen, Fig. 2. A few new customizing commands help to layout the four buttons controlling recording and playback. The *alignParentStart* command, Line 434, makes the leading edge of the START RECORDING button match that of the STOP RECORDING button. The attributes *alignParentLeft* and *alignParentRight* place these buttons one next to another. We have used a new *layout_below* command, Line 462, to put one button below another button. If you want to have more information concerning layout attributes visit *developer.android.com/reference/android*.

Since it just takes a few lines, we decided to add a microphone picture to the top of the screen. The new widget, *ImageView*, is used to load and display images from the "*WAT_AN_APP\res\drawable-xxx"* folders that store images with different resolutions. The "*Quick Help Guide*" section at the end of the article shows how to add an "*ic_action_mic.png*" microphone picture in the "*WAT_AN_APP\res\drawable-mdpi*" folder.

As for the text and buttons, the *ImageView* widget has to be given an id, "*microphone*", Line 421. We set the placement and source of the picture to display using Lines 422 and 423.

```
<!—Used by AudioRecordPlayback -->
400. <RelativeLayout
401. xmlns:android=http://schemas.android.com/apk/res/android
402. .... <!--SAME AS Article 1 Listing 4, Lines 402 to 405-->

410.    <-- Delete Article 1 Listing 5 Lines 410 – 417 -->

        <!-- Small microphone image -->
420.    <ImageView
421.     android:id="@+id/microphone"
422.     android:layout_marginTop="150dp"
423.     android:src="@drawable/ic_action_mic"
424.     tools:ignore="ContentDescription"
425.     android:layout_width="wrap_content"
426.     android:layout_height="wrap_content"
427.     android:layout_centerHorizontal="true"
428.    />

         <!-- Start recording button -->
430.    <Button
431.     android:id="@+id/button_start_rec"
432.     android:onClick="start_recording"
433.     android:text="@string/start_recording"
434.     android:layout_alignParentStart="true"
435.     android:layout_alignParentLeft="true"
436.     android:layout_width="wrap_content"
437.     android:layout_height="wrap_content"
438.     android:layout_centerHorizontal="true"
439.     android:layout_centerVertical="true"
440.    />

        <!-- Stop recording button -->
450.    <Button
451.     android:id="@+id/button_stop_rec"
452.     android:onClick="stop_recording"
453.     android:text="@string/stop_recording"
454.     android:layout_alignParentRight="true"
455.     android:layout_alignParentEnd ="true"
456.     <!-- COPY Lines 436 to 439 -->
457.    />

        <!-- Start playback button -->
460.    <Button
461.     android:id="@+id/button_start_playback"
462.     android:layout_below="@+id/button_stop_rec"
463.     android:onClick="start_playback"
464.     android:text="@string/start_playback"
465.     <!-- COPY Lines 436 to 439 -->
466.    />

        <!-- Stop playback button -->
470.    <Button
471.     android:id="@+id/button_stop_playback"
472.     android:layout_below="@+id/button_start_playback"
473.     android:onClick="stop_playback"
474.     android:text="@string/stop_playback"
475.     <!-- COPY Lines 436 to 439 -->
476.    />

499. </RelativeLayout>
```

Listing 2) This updated *activity_audio_record_playback.xml* layout file (*WAT_AN_APP\res\layout* folder) generates the five .wav control buttons shown in Fig. 2

ADDING THE JEAC AUDIO ACTIVITY

Android offers a simple *MediaRecorder* class which offers a “blackbox” designed to capture, save and playback all types of media including pictures, videos and audio. We have followed two online examples to build an audio recorder/player: *developer.android.com/guide/topics/media/ audio-capture.html* and *tutorialspoint.com/android/android_audio_capture.htm*.

Listings 3A and 3B show the *AudioRecordPlayback* activity. We start by defining our MediaRecorder *myRecorder*, the MediaPlayer *myPlayer*, the file name *outputFileName* for our recording and the four buttons, Lines 316 to 319. We then code the *onCreate()* method to initialize our activity. Line 322 sets the user interface (UI) from the layout resource, Listing 2, using *setContentView()*. Finally we detail the four buttons we need to interact with the application and set the *start_recording* button as active, Line 327.

Pressing the *start_recording* button will activate the *start_recording* public method, Listing 3B Line 340. This is a busy method which sets up the recording file path as well as the name of the recording *myrecording.3gpp* in Line 341. The *MediaRecorder* is initialized, Lines 342 to 346, and starts a recording, Lines 349. The method finishes with a flourish by disabling and ghosting the *start_recording* button, activating the *stop_recording* button, and issues a toast, Android message, on the screen to show the user that a recording has started, Lines 352 to 354.

Listing 3B also shows the similar format of the other audio control methods: *stop_recording*, Lines 360 to 366, start*_playback*, Lines 370 to 377 and *stop_playback* Lines, 380 to 386. They each manipulate the *MediaPlayer*, enable the next method in the audio control stream before turning themselves off.

The fact that these all methods turn themselves reminded Mike of an early electronic toy he used to have in a much more simple time. When the switch on the top of the toy was turned on, the toy’s box lid opened and a hand came out and pushed the switch to turn the toy off.

Once the recording has been stopped, Listing 3B Line 361 it is important to issue a release *MediaRecorder* command, Line 362. This frees-up the audio hardware and other system resources which are all shared across the different applications running on the phone.

```
// Replace existing code from Article 1 Listing 4 Lines 300 to 302
300. package com.wat_an_app;
301. import android.widget.Toast;
302. import android.os.Bundle;
303. import android.os.Environment;
304. import android.widget.Button;
305. import android.view.View;
306. import android.support.v7.app.ActionBarActivity;
307. import android.media.MediaPlayer;
308. import android.media.MediaRecorder;
309. import java.io.IOException;
310. import com.wat_an_app.R;

315. public class AudioRecordPlayback
                              extends ActionBarActivity {
316.   private MediaRecorder myRecorder;
317.   public MediaPlayer myPlayer = null;
318.   private String outputFileName = null;
319.   private Button button_start_recording,
       button_stop_recording,   button_start_playback,
       button_stop_playback;

320.   @Override
       protected void onCreate(Bundle savedInstanceState) {
321.     super.onCreate(savedInstanceState);
322.     setContentView(R.layout.activity_audio_record_playback);
323.     button_start_recording
            = (Button)findViewById (R.id.button_start_rec);
324.     button_stop_recording
            = (Button)findViewById (R.id.button_stop_rec);
325.     button_start_playback
           = (Button)findViewById (R.id.button_start_playback);
326.     button_stop_playback
            = (Button)findViewById(R.id.button_stop_playback);

327.     button_start_recording.setEnabled(true);
328.     button_stop_recording.setEnabled(false);
329.     button_start_playback.setEnabled(false);
330.     button_stop_playback.setEnabled(false);
331.   }

    //  public void start_recording(View view)
        //Listing 3B Lines 340 to 355

    // public void stop_recording(View view)
        //Listing 3B Lines 360 to 366

    // public void start_playback(View view) throws
          IllegalArgumentException, SecurityException,
            IllegalStateException, IOException
              //Listing 3B Lines 370 to 377

    //  public void stop_playback(View view)
         //Listing 3B Lines 380 to 386

399. } // End class AudioRecordPlayBack
```

Listing 3A) The prologue of the *AudioRecordPlayback.java* file (*WAT_AN_APP\src\* folder) sets up the User Interface. The *OnCreate()* method enables the *Start Recording* button (Line 327). The other methods in this class detailed in Listing 3B.

```
340.   public void start_recording(View view) {
341.   outputFileName =
           Environment.getExternalStorageDirectory().
               getAbsolutePath()+ "/myrecording.3gpp";
342.   myRecorder = new MediaRecorder();
343.   myRecorder.setAudioSource(MediaRecorder.
                            AudioSource.MIC);
344.   myRecorder.setOutputFormat(MediaRecorder.
                            OutputFormat.THREE_GPP);
345.   myRecorder.setAudioEncoder(MediaRecorder.
                            OutputFormat.AMR_NB);
346.   myRecorder.setOutputFile(outputFileName);
347.   try {
348.     myRecorder.prepare();
349.     myRecorder.start();
350.   }catch (IllegalStateException e) {e.printStackTrace();}
351.   catch (IOException e) {e.printStackTrace();}
352.   button_start_recording.setEnabled(false);
353.   button_stop_recording.setEnabled(true);
354.   Toast.makeText(getApplicationContext(),
         "Start Recording", Toast.LENGTH_SHORT).show();
355.  }

360.  public void stop_recording(View view){
361.   myRecorder.stop();
362.   myRecorder.release();
363.   myRecorder  = null;
364.   button_stop_recording.setEnabled(false);
365.   button_start_playback.setEnabled(true);
366.  }

370.  public void start_playback(View view)
       throws
         IllegalArgumentException, SecurityException,
          IllegalStateException, IOException{
371.   myPlayer = new MediaPlayer();
372.   myPlayer.setDataSource(outputFileName);
373.   myPlayer.prepare();
374.   myPlayer.start();
375.   button_start_playback.setEnabled(false);
376.   button_stop_playback.setEnabled(true);
377.  }

380.  public void stop_playback(View view){
381.   button_stop_playback.setEnabled(false);
382.   myPlayer.release();
383.   myPlayer = null;
384.   Toast.makeText(getApplicationContext(),
       "Stop Playing Back", Toast.LENGTH_SHORT)
           .show();
385.   button_start_recording.setEnabled(true);
386.  }

399. }
```

Listing 3B) Details of the *Recording* and *Playback* methods from the *AudioRecordPlayback.java* file (*WAT_AN_APP**src*\ folder)

```
200. <?xml version="1.0" encoding="utf-8"?>
201. <resources>
205. <!-- SAME AS Article 1 Listing 3, Lines 205 to
                        214 -->

220.  <!-- String required for the second part of the
        record and playback a sound -->
221.  <string name="start_recording">
            Start recording</string>
222.  <string name="stop_recording">
            Stop recording</string>
223.  <string name="start_playback">
            Start playback .3GPP</string>
224.  <string name="stop_playback">
            Stop playback .3GPP</string>
249. </resources>
```

Listing 4) To avoid compiler warning messages, new preset string values must be set in *strings.xml* (*WAT_AN_APP**res**values* folder)

We can start playing this recorded file by calling the *start_playback* method which re-initializes the *MediaPlayer* in a play-back mode, Line 371. We identify the recorded file, stored in *outputFileName*, then prepare the player to begin playing data. Pressing the stop playback button again releases the *MediaPlayer* resources back to the system and displays a "*Stop Playing Back*" message on the screen.

Just before you hit the compile button for the last time, remember we have been using a lot of strings. As in Article 1, avoid the compiler warning messages by adding preset strings to the *strings.xml* file in the "*WAT_AN_APP\res\values*" folder (Listing 4).

FORGIVENESS AND PERMISSIONS

In the everyday world there is a saying

"*Sometimes you get further ahead by asking for forgiveness rather than asking for permission.*"

In our JEAC world there is an equivalent saying

"Sometimes your project finishes faster if you set permissions to allow a few things rather than writing more code to allow everything."

For example, do you want to hunt ghosts or write the code needed to ensure your app can handle you switching from portrait to landscape modes? Currently an event will be trigged that will restart the audio activity if you rotate the screen while recording or playing. That risks the *MediaRecorder* or *MediaPlayer* not being properly released and re-initialized and causing the activity to crash at that point. Solve this by disabling the auto-rotate permissions in your phone's settings menu.

Other potential issues can also be prevented, rather than requiring coding, by setting permissions in the *AndroidManifest.xml* file in the "*WAT_AN_APP*" project's root directory. For example, Line 3011 in Listing 5 stops the app from going to sleep while we are recording because switching the sleeping screen

```
3000.<?xml version="1.0" encoding="utf-8"?>
3001.  <manifest xmlns:android=
          "http://schemas.android.com/apk/res/android"
3002.  package="com.wat_an_app"
3003.  android:versionCode="1"
3004.  android:versionName="1.0" >

        <!--Check lines 3005 and 3007 and update if necessary-->
3005.   <uses-sdk
3006.     android:minSdkVersion="14"
3007.     android:targetSdkVersion="21" />

        <!--Manually add lines 3008 to 3011-->
3008.   <uses-permission   android:name=
            "android.permission.RECORD_AUDIO" />
3009.   <uses-permission   android:name=
            "android.permission.WRITE_EXTERNAL_STORAGE" />
3010.   <uses-permission android:name=
          "android.permission.READ_EXTERNAL_STORAGE" />
3011.   <uses-permission android:name=
          "android.permission.WAKE_LOCK" />

        < !--Automatically added -->
3012.   <application
3013.     android:allowBackup="true"
3014.     android:icon="@drawable/ic_launcher"
3015.     android:label="@string/app_name"
3016.     android:theme="@style/AppTheme" >

3017.     <activity
3018.       android:name=".MainActivity"
3019.       android:label="@string/app_name" >

3020.       <intent-filter>
3021.         <action android:name="android.intent.action.MAIN" />
3022.         <category
3023.           android:name="android.intent.category.LAUNCHER" />
3024.       </intent-filter>

3025.     </activity>

3026.     <activity
3027.       android:name=".AudioRecordPlayback"
3028.       android:label="@string/title_activity_audio_record_playback" >
3029.     </activity>

3030.   </application>

3031.  </manifest>
```

Listing 5) *AndroidManifest.xml* from the *WAT_AN_APP* project's root directory

off can cause the current audio record / playback operation to stop.

Currently the app is like a well-trained dog sitting at your feet with ears actively waiting for your command to *GO AND PLAY*! So give our application the right to listen (record audio) and to remember what we have told it (read and write on external storage), Lines 3008 to 3010. Please note that the minimum SDK version the application can run on has been set to "14", Line 3006, compare to "9" in the first article. Make sure that it has been set to "14" in

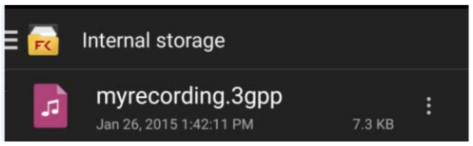

Fig 3) Recorded file physically present in the phone internal storage memory

your *AndroidManifest.xml* file as well, avoiding compatibility issue with the code we have implemented. On one of our phones, a Nexus 5 from Google, there is not physical slot for adding an external micro SD memory card. The recorded audio file is then automatically saved into the phone internal storage memory (Fig.3).

CAN A (LOG)CAT SEE GHOSTS?

In a much more far away time and civilization, cats were regarded as gods, supposedly for their unique perception of the spirit world.

When developing this app, we started to appreciate that the LogCat tool has a unique perception of your Android system. LogCat can be used to view and filter logs from applications and portions of the Android system. A crash, error or warning details for an application are outputted in the LogCat window.

A system crash message is easily interpreted, but that is not so for the some warning and error messages. As they say on the TV – *"For that there is stackoverflow.com"* with proposed solutions from the wide Android programming community. Reducing coding time with a JEAC approach means that the code is not "commercial release grade". So when debugging you should expect to have to click through (ignore) some LogCat error messages. For example, if your phone runs on the Lollipop OS then LogCat complains that we "*should have subtitle controller already set*" every time the *MediaPlayer* starts playing the recorded sound.

If you have fast enough reflexes it is possible to make LogCat issue an error message of the form "*Get Occurred on inactive InputConnection"*. From Stackoverflow I have learnt that *InputConnection* is the communication channel interface from an input method back to the application. You can get other error messages over this channel if some part of the app like a Toast is taking too long to respond.

Stackoverflow.com offers many solutions for this sort of problem. However, I would rather click through and take the JEAC amendment – "*Enough coding already – lets go ghost hunting."*

OUR APP IS READY FOR SOME 4H TIME

With this article we are now capable of recording a community noise outside in the neighbourhood and playing the sound in a quieter environment where we can examine it in more detail.

We are also ready to put in some 4H time -- *Happy Hunting Haunting Hours*. Watch out for our next article where we give up our qualitative ghost hunting approach and go in for real boasting rights. With a quantitative ghost hunting app, we will be able to prove that there are more ghosts around in our neighbourhood than anywhere else in the world!

BIOGRAPHY

Adrien Gaspard received his Master of Engineering from CPE Lyon, France in February 2015. He tackled his final practicum as an exchange student in Electrical and Computer Engineering at the University of Calgary. He undertook self-directed term projects directed towards the possible use of noise cancelling to solve the community noise problem in Calgary community of Ranchlands. His long term career goal is in embedded systems and wireless telecommunications. He can be contacted at gasp.adrien@gmail.com

Mike Smith has been contributing to Circuit Cellar magazine since the '80s. He is a professor in computer engineering at the University of Calgary, Canada. His main interests are in developing new biomedical engineering algorithms and moving them onto multi-core and multiple-processor embedded systems in a systematic and reliable fashion. Mike has recently become a convert to the application of Agile Methodologies in the embedded environment. He is Analog Devices University Ambassador (2001 – 2015). He can be contacted at mike.smith@ucalgary.ca

QUICK HELP GUIDE
If you want some additional coding help, call Android Busters This is who we looked up and called – online -- when we needed a tutorial about how to create an audio recorder/playback or for help to bust the hard, and sometimes simple, Android problems that we found "haunting" us. Tutorial at Android Developer (Audio Capture): *developer.android.com/guide/topics/media/audio-capture.html* Tutorial at Tutorials Point (Audio Record/Playback): *http://tutorialspoint.com/android/android_audio_capture.htm* Android Community help at *http://stackoverflow.com/questions/tagged/android*
Android offers several materials, including an icon pack, to facilitate our app design and implementation. Please go to *https://developer.android.com/design/downloads/index.html* and download the "Action Bar Icon Pack". In the "*…\Android_Design_Icons_20131106\Android Design - Icons 20131120\Action Bar Icons\holo_dark\08_camera_mic"* folder, you will find four drawable folders available, with the parameters *hdpi*, *mdpi*, *xhdpi* or *xxhdpi*. These parameters correspond to the density of the picture, which defines its quality. I recommend using *mdpi* as it corresponds to the medium density. Grab the *ic_action_mic.png* file from this folder and copy it in the "*WAT_AN_APP\res\drawable-mdpi\*" folder from Eclipse. This allows defining the source "*src="@drawable/ic_action_mic"* in the *ImageView* tag**.**

Table 1) Quick help guide