Early Language Assistance Therapy (ELAT) Robot for Children with Autism Spectrum Disorder (ASD)

Miguel Esteban, Yu-chun (Eugene) Hsiao, Jonathan Ko, Myles Lewis, Remi Shittu, Andrew Yoon, Maurie Zhang

By signing below, I (we) certify that I (we) have read and have a copy of the Final Report.

Name of Technical Advisor: DR. Andrey Vyshedskiy Signature:

Name of Technical Advisor: ________________ Signature: ______________________

Department of Biomedical Engineering
Boston University

April 19 2019

Abstract

1 in 59 children is affected by Autism Spectrum Disorder (ASD), which can cause decreased language development.[1] Current speech therapy is dependent on visits with Speech Language Pathologists (SLPs), however, SLPs are not always accessible or affordable. Thus, there is a need for a more cost and time effective therapy.

An audio analyzer was reconstructed and combined with the SDK of WowWee's CHiP robot. High pass filters were applied to the audio analyzer in order to isolate the vocalizations of children. The combined product was then optimized for usability, stability, and functionality.

An automatic speech recognition (ASR) algorithm was developed to analyze the verbalization attempts of children. A machine learning approach was used to develop this algorithm, and it was trained on three datasets commonly used for language processing. The algorithm outputs a character-based spelling of the input.

The product is able to detect vocalization and provide rewards through the robot. It can distinguish between adult and children, and will only reward child vocalizations. The ASR is capable of printing out characters based on its input. From that, a percent similarity score is calculated based on the similarity between the output and the target word. This can be used in future endeavors to further track children's progress through their speech therapy.

This product will have great implications in providing speech therapy for children affected by ASD. The product will provide a cost effective form of therapy that addresses the issues of SLP cost and availability.

Introduction & Background

Autism spectrum disorder is a neurodevelopmental disorder that causes several impairments to a child's social interactions and communication skills.[2] The Centers for Disease Control estimates that 1 in 59 children is affected by autism spectrum disorder (ASD) that disrupts early development in social communication and behavior.[3] Approximately two-thirds of children with ASD will eventually have significant cognitive social impairments.[4] Of the children affected by ASD, thirty to forty percent of the children will remain nonverbal into adulthood.[5] Because of the amount of children that remain nonverbal, the American Academy of Pediatrics (AAP) recommends that all children are screened for ASD, and that they immediate start intensive treatment if diagnosed with ASD.[6] In addition to the prevalence of this disorder in the U.S. population, the cost of caring for children with autism spectrum disorder ranges from $11.5 billion to $60.9 billion.[7] As a result of the expensive cost of care, many children do not receive the care that they need to develop proper verbal communication skills. Furthermore, there has been an increase in the number of children diagnosed with ASD over the years, while most places in the United States have seen a decrease in the number of ASD-trained professionals.[8,9] Families of newly diagnosed children face long waitlists for therapy during the most critical early period of development. Some families receive no treatment in these cases.

However, the use of technology can aid in shrinking the gap between the amount of therapy that is recommended for children with ASD, and the amount that they actually receive. Over the last five years ImagiRation has developed early-intervention applications for children with ASD. The application, Mental Imagery Therapy for Autism (MITA), trains users in verbal exercises with simple words and progresses to more complex phrases and sentences. Although these methods of treatment are useful and effective in improving receptive language skills, the current methods do not thoroughly provide the social and verbal intervention that autistic children need to develop satisfactory verbal communication and social interaction. The proposed Early Language Assistance Therapy (ELAT) Robot for autistic children is a form of treatment that is able to work with children in order to improve their verbal, expressive communication skills.

Research has shown that expressive speech therapy can be accomplished using a form of operant conditioning, where a positive stimulus is administered as a child progresses in their speech development. This motivates the child to vocalize and continue in their progression to coherent speech.[10,11,12,13,14,15] ELAT seeks to provide expressive speech therapy using a similar model. ELAT uses an affordable robot to interact with children. Children seem to have an affinity for anthropomorphic robots, and it has been shown that these robots can inspire children to vocalize.[16,17,18] The ELAT robot provides positive stimulus, in the form of lights, sounds, and dances, to the child as he/she develops in their vocalizations. ELAT also the capability to detect children's vocalizations in real time using a smartphone application. There are existing programs that can accomplish this with adults; however, this same software cannot be applied to children who cannot form clear words. Combining these two approaches provides a powerful tool that can motivate non-verbal children to begin vocalizing, and could present a novel approach and supplement to language therapy. In addition, ELAT also has an adaptive language detection aspect. Using a machine learning approach, a word recognition software has been developed. The Automated Speech Recognition (ASR) algorithm is able to receive an audio input and output character spellings of what it receives. The characters are compared to a target and a grade is applied to the trial. As the ELAT product goes through future iterations, this analytical tool can be used to gauge a child's progress throughout the treatment. ELAT will be able to recognize how well a child is performing and will start to require closer approximations to a target before administering the reward.

The U.S faces significant numbers in ASD diagnoses and suffers from a lack of Speech Therapists to resolve this issue. There is a need to fill this void in speech language therapy for children diagnosed with ASD. The use of technology has been available to aid in this respect;

however, little has been done to treat children in their expressive language skills. ELAT is significant in the fact that it is a novel treatment, applying operant condition to expressive language therapy. It is an affordable alternative or supplement to ASD affected children in order to maximize their therapy in critical years of development, in hopes of reducing the rate of children remaining non-verbal through adulthood.

Aims

The goal of this project is to help children affected by Autism Spectrum Disorder (ASD) improve their language and social skills, so that they reach a level where they feel confident practicing and interacting more with other children and adults.

To combat the effects of ASD we need to address the language deficiency as early as possible[1] and build up the child's language repertoire. This project proposes a solution: the product is an interactive language therapy robot which will work alongside current methods of Speech Language Therapy in order to strengthen the vocalization abilities of children with ASD.

The functional requirements of the robot are that it must be able to record and evaluate child vocalizations, and to help the child overcome his/her own specific language vocalization difficulties.

The WowWee dog robot model, CHiP, was chosen because, of its affordability and life-like characteristics. The robot will be interactive so that the child will be able to trigger certain actions, such as dancing or singing, with his/her voice. These actions are meant to be positive reinforcement for the child performing a desired action. This will motivate the child to progress in their speech development.

Aim 1: Develop a software to detect vocalizations, calculate a percent similarity value, and adapt.

The purpose of the software is to allow for sound funneling, to help the child refine incoherent vocalizations into more intelligible sounds, which are understandable to English speakers. The child's iteration of the model word will be compared to accepted pronunciations of the model world; the software will return a similarity index as a percentage of closeness to the accurate pronunciation.

Aim 2: Integrate the developed software into an affordable robot.

This robot will serve as a user interface for the software described in Aim 1. Although this project aims to make use of the CHIP robot provided by WowWee, the framework allows for a variety of robotic interfaces that allow the treatment to be personalized for each child.

Methods

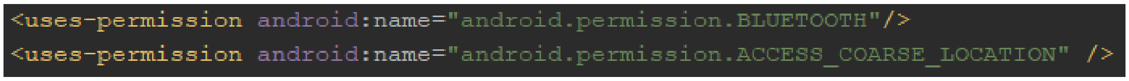
Audio Analyzer/CHiP SDK

A combined solution for capturing vocalization attempts and interaction with the WowWee CHiP robot was designed by adapting and merging an Audio Analyzer software provided by Dr. Vyshedskiy, and a Software Development Kit (SDK) created for CHiP. All modifications were made through Android Studios, the officially integrated development environment (IDE) for Google's Android operating system. The ease of compatibility of Android Studios with a phone connected to a WowWee robot provided a platform where specific tasks can be accomplished by writing function code. The Audio Analyzer was designed to take blocks of vocal data, each 64 ms long, and perform the Fourier Transform of that block. From the Fourier Transform, the power spectral density of the sample was calculated in order to capture the surrounding sound and registered whether or not a vocalization was detected. The program then identifies three peaks in different regions of the power spectral density in order to compare those peaks to background levels in those regions. The intensity threshold for each peak was chosen to be 20 dB above the mean in order to qualify as a significant difference. From there, a weighted ratio was calculated, with a ratio of 10 or greater representing a vocalization. A filter was also implemented to block out adult vocalizations so that only correct children's' vocalizations are detected

This functionality from the Audio Analyzer was integrated with the Bluetooth connection and base functionality fragments of the CHiP SDK in order to allow the the app and the robot to connect. A new fragment of the app was created under the name toDogActivity to push the output of the Audio Analyzer vocalization detector and to command the robot to react when a correct vocalization is detected.

Android Studio

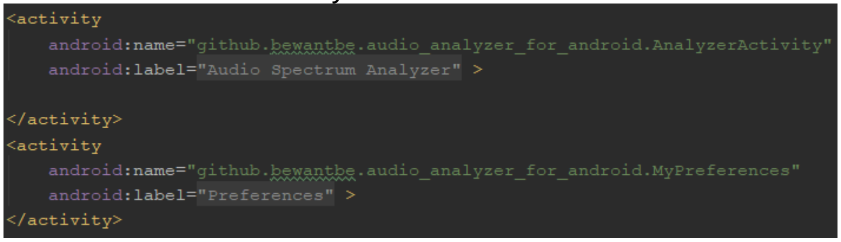
Given that two separate functioning programs were being edited and merged into one cohesive whole, there were clashes in software versions which had to be resolved. To start, the AndroidManifest.xml file had to be updated to reflect the inclusion of the newly written activities, as well as to allow for Android permission to access bluetooth and location services, to be able to detect and connect to the CHiP. To add permissions, we added the two lines of code within the Manifest file, and during runtime had the program request permission from each user.



```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Figure 1. Requesting Permission from User.

Updating the manifest file to include the new activities (figure below) involved making a new <activity> label within the .xml file, which both links the added activity to other activities, and also labels it as necessary.



```
<activity
    android:name="github.bewantbe.audio_analyzer_for_android.AnalyzerActivity"
    android:label="Audio Spectrum Analyzer" >

</activity>
<activity
    android:name="github.bewantbe.audio_analyzer_for_android.MyPreferences"
    android:label="Preferences" >
</activity>
```

Figure 2. Updating Manifest File to Include and Link Activities

The second step of merging the two programs, involved updating the Android Studio '.gradle' files. Gradle is the system's way of keeping track of software versions and managing how the project manages all of the separate java, xml, and resource files. Given that the original

AudioAnalyzer was written for a much earlier implementation of Android Studio. First, the target and minimum SDK versions were changed to match the Wowwee robot levels. Next, the bluetooth library for the robot used was imported along with the FFTLibrary used by the audio analyzer. Then build type differences between the CHIP-SDK and the AudioAnalyzer were resolved by using the sdk versions as they were less restrictive. Finally, the depreciated library dependency, “compile”, was replaced with the appropriate “api”, and a large number of depreciated import statements were updated.

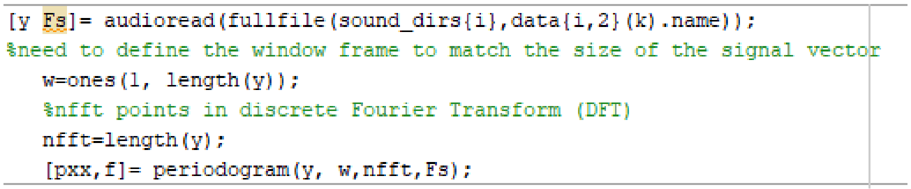
Automated Speech Recognition (ASR)

The automatic speech recognition algorithm was trained on three language corpus released data sets: VCTK, LibriSpeech, and TEDlium. These three data sets are common widespread training datasets in the natural language processing community, and have been proven to exhibit powerful, yet generalized predictive power. The final output of our algorithm is a character-based spelling of the input word .wav file.

DSP to Assess Child Vowel Space (MATLAB)

FmapsChild.m looks at 4 different American English word vocalizations made by young children aged 2- 5 yrs old; the words considered were *Ball, Daddy, Jeep, and No*. The original directories, included a large number unintelligible .wavs, these files were deleted to avoid distortion of any meaningful outputs. Additionally, child .wav files, where the difference between male or female speaker was distinguishable, were discounted from being treated in the *FmapsChild.m*, because these were more in the adult range rather than young child vocalizations.

These deductions from the child directory .wav files, greatly limited the data points. The adjusted *Ball directory* includes 13 .wav files from 35% to 60% intelligibility. Adjusted *Daddy directory* includes 5 .wav files, from 20 - 70% language intelligibility. Most of *Jeep*’s original .wavs had mostly 0% intelligibility, so the adjusted set includes only 5 .wav files from 5-15% intelligibility. Adjusted *No* includes 3 .wav files, from 30-70% language intelligibility.



```
[y Fs]= audioread(fullfile(sound_dirs{i},data{i,2}(k).name));
%need to define the window frame to match the size of the signal vector
   w=ones(1, length(y));
   %nfft points in discrete Fourier Transform (DFT)
   nfft=length(y);
   [pxx,f]= periodogram(y, w,nfft,Fs);
```

Figure 3. Periodogram processing to obtain 1:1 vectors ***pxx*** *and* ***f.***

FmapsChild.m opens each sound directory independently, processes the raw speech data using *audioread( ),* and uses *periodogram( ),* to calculate the Power Spectral Density (PSD) of the speech signal. periodogram () differs from a fourier transform process, in that, the PSD describes the intensity of the signal distributed over frequency, while a Fourier Transform describes the spectral content of the signal, the amplitude and phase of harmonics in your signal[21].

[pxx, f] = periodogram (y, w, nfft, Fs) , returns a vector *pxx* which corresponds to the PSD - y values , and vector *f* corresponds to the frequency values paired with the y- PSD values. *pxx* and *f* match one to one, so finding the local maxima *pxx* values, provides the indices to acquire the formant frequencies, F1, F2, and F3.[22]

FmapsChild.m considers each word folder individually, reads all the .wav files in a folder, finds one *[pxx f]* and F1:F2:F3 for each .wav file, and stores all the data into a multi-dimensional cell called *data.* Looping through data provides the program outputs: 4 plots, one 3-D plot comparing F1:F2:F3 for all words, and three 2-D plots comparing only 2 different formants at a time.

The *.wav* files considered to be in the adult category are processed through *FmapsAdult.m*. Changing only the path variables, allowed for use of the same code to provide two different plots. The same could have been accomplished using just one comprehensive program, but two sets of variables and would have to be created, having the same variable names makes the programs more easily readable.

Unaltered Audio Analyzer Testing

In order to test the viability of the unaltered Audio Analyzer, the program was run and vocalizations were presented, ie. a person made an audible *"aaah"* sound at about 60dB. The program's response was recorded over a period of time where there was silence, the vocalization was presented, and silence following the vocalization.

Audio Analyzer/CHiP SDK Implementation (Merged Product Testing)

The merged Android Analyzer—CHiP SDK was tested in order to evaluate how well the product was able to recognize meaningful vocalizations. The product was tested against 3 negative controls (silence, whispers, background noise) and tested against 2 experimental tests, both considering normal conversational vocalizations, about 60 dB, with regard to range, closeness versus a distance of 5 feet. The participant set included 5 participants, 3 males, 2 females, and with each participant repeating each test three times.

Each test consisted of a seated participant holding the Android Device, where the microphone was located, in their hand, resting it on the table in front of them. The first negative control test consisted of each participant sitting silently, and the robot's response was recorded. The second negative control trial consisted of each participant whispering at a level less than 20 dB above the background. Each whisper was monitored using the unaltered Audio Analyzer to ensure these conditions were met. The robot's response was recorded. The third negative control test considered effects of vibrational noise, and consisted of each participant hitting the table next to the Android device with their hand, the robot's response was recorded. The fourth test, consisted of each participant speaking at a regular volume level, and the robot's response was recorded. The fifth test consisted of the participant speaking regularly at a distance of about five feet from the Android device, and the robot's response was recorded.

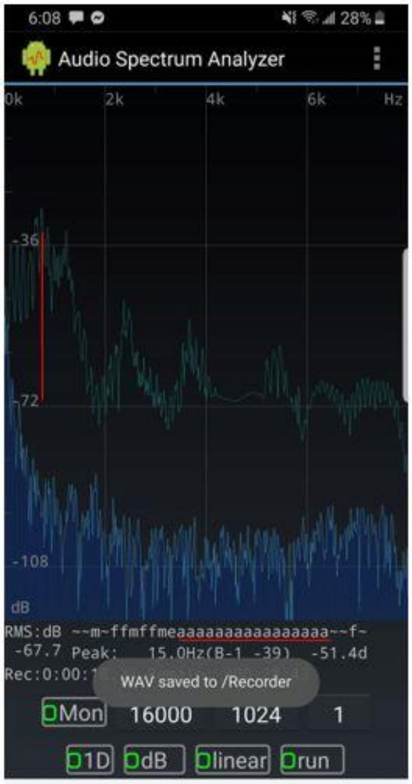
Automated Speech Recognition (ASR) Testing

In order to determine the effectiveness of our ASR system, we tested the program on a dataset of five words (Ball, Daddy, Jeep, No, Teapot) provided to us by our Technical Adviser, Dr. Vyshedskiy. This five-word dataset, includes several recordings of vocalization attempts of each word by both autistic children and adults, and vocalization quality ranges from 20% correct pronunciation, to perfect pronunciation.

We ran the ASR program using each recording, individually. The automatic speech recognition program then examined each recording and attempted to detect recognizable vowel sounds in order to construct a character output based on its own recognition of sounds. After all of the outputs were collected, we analyzed the alphabetic spellings of each of the character outputs and scored them on a scale of 1-100. Lastly, the score that was attributed to the automatic speech recognition program was plotted against the subjective quality scores of each recording and we determined the correlation coefficient based on the trend line in Excel.

Results

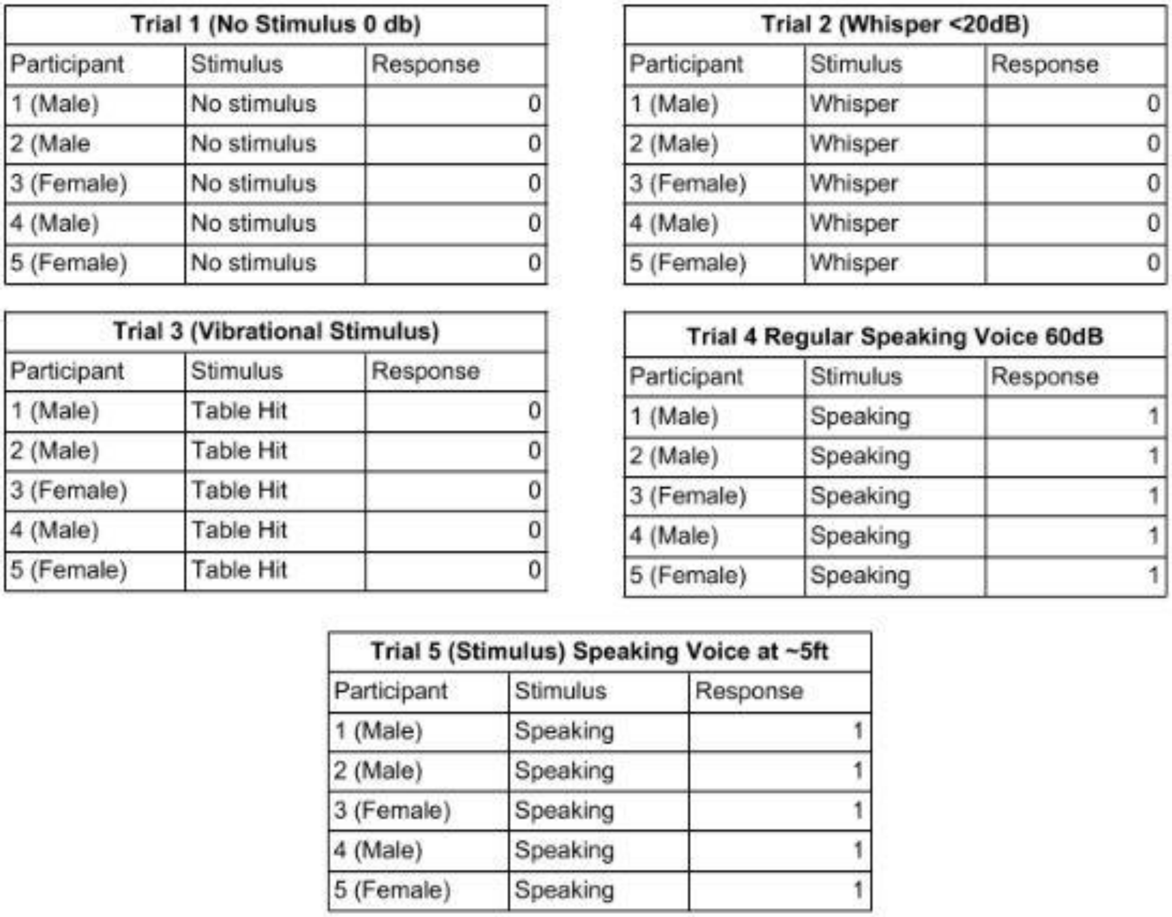
The Audio Analyzer is an integral part of the project, since it captures vocalizations and determines if the robot reacts to the child's vocalizations and words. The results described are the results from the accuracy tests that were run to test the viability of the Audio Analyzer. The results of the unaltered Audio Analyzer Testing showed that the program performed as it was intended to. The program monitors the surrounding sounds in 64 ms intervals and assigns a "classification character" to each interval. The "a" classifier demonstrates that a vocalization was detected.



```
RMS:dB ~~m~ffmffmeaaaaaaaaaaaaaaaa~~f~
 -67.7 Peak:   15.0Hz(B-1 -39)  -51.4d
Rec:0:00:1
```

Figure 4: The figure above shows the display of the Audio Analyzer Function. The classification characters can be seen directly below the graph. The string of "a" characters (underlined) shows the occurrence of the vocalization. The vertical displacement (marked in red) shows the primary peak standing 20 dB above the background in order to qualify as a vocalization

The results of the merged Audio Analyzer, CHiP SDK testing show that the robot responded appropriately 100% of the time. During Trial 1, no stimulus was presented and the robot did not present any reward. During Trial 2, a stimulus was introduced. However, the stimulus did not break the threshold to qualify as a vocalization, and the robot did not present any reward. During Trial 3, a hand hit the table with no vocalization accompanying it. Again, the robot did not present any reward. Trial 4 consisted of a regular speaking level vocalization, and the robot presented the reward appropriately. Trial 5 consisted of a regular speaking level vocalization performed at a distance of about 5 feet from the microphone. The robot still detected the vocalization and presented the reward appropriately. These experiments showed that the basic functionality of the robot performs as it should. However, these tests did not include the filters to eliminate adult speech and isolate children's vocalization. More testing will be required to see how the robot performs with the applied filters.



Trial 1 (No Stimulus 0 db)		
Participant	Stimulus	Response
1 (Male)	No stimulus	0
2 (Male	No stimulus	0
3 (Female)	No stimulus	0
4 (Male)	No stimulus	0
5 (Female)	No stimulus	0

Trial 2 (Whisper <20dB)		
Participant	Stimulus	Response
1 (Male)	Whisper	0
2 (Male)	Whisper	0
3 (Female)	Whisper	0
4 (Male)	Whisper	0
5 (Female)	Whisper	0

Trial 3 (Vibrational Stimulus)		
Participant	Stimulus	Response
1 (Male)	Table Hit	0
2 (Male)	Table Hit	0
3 (Female)	Table Hit	0
4 (Male)	Table Hit	0
5 (Female)	Table Hit	0

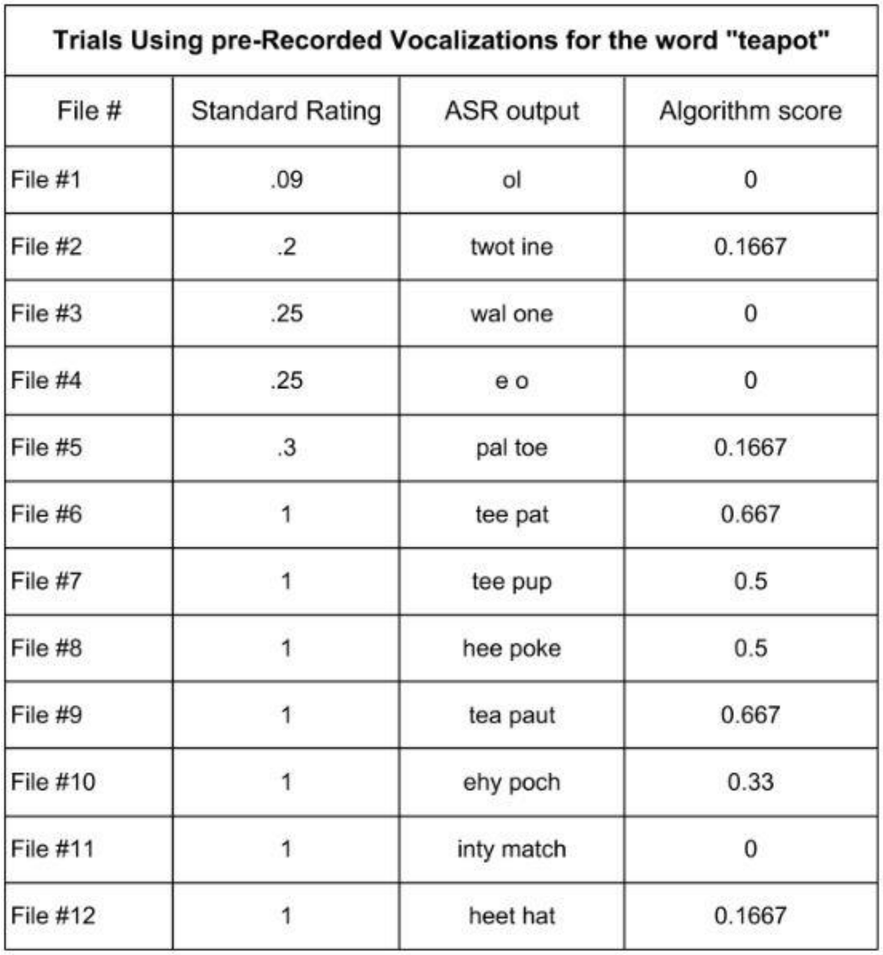
Trial 4 Regular Speaking Voice 60dB		
Participant	Stimulus	Response
1 (Male)	Speaking	1
2 (Male)	Speaking	1
3 (Female)	Speaking	1
4 (Male)	Speaking	1
5 (Female)	Speaking	1

Trial 5 (Stimulus) Speaking Voice at ~5ft		
Participant	Stimulus	Response
1 (Male)	Speaking	1
2 (Male)	Speaking	1
3 (Female)	Speaking	1
4 (Male)	Speaking	1
5 (Female)	Speaking	1

Figure 5: The figure above displays the results from the accuracy tests for the Audio Analyzer. These results demonstrate the accuracy of the Audio Analyzer in addition to its ability to detect vocalizations and respond appropriately. For each trial a different stimulus was presented, during Trial 1 there was no stimulus. For Trial 2 there was a stimulus presented, whispering, speech at less than 20 decibels. During Trial 3 there was a vibrational stimulus introduced. During Trial 4 the stimulus was regular speech at 60 decibels and during Trial 5 the stimulus was regular speech from a distance of approximately 5 feet away.

The Automatic Speech Recognition (ASR) algorithm is a critical component of the software for the robot, since the ASR algorithm will determine the what the autistic child is attempting to say by deciphering the vocalizations phonetically. The output of the algorithm is a character- spelling of what the algorithm is able to identify. In addition to the the identification of the word, another component of the code scores the accuracy of the output by comparing the spelling of the output word to the word that is prompted, in this case a test word. Since little is known about how machine learning algorithms exactly assign letters and spellings to words that it detects, it is important that the algorithm detects the correct phonemes most of the time. Phonemes are units of the phonetic system that correspond to a set of speech sounds that make distinct sounds in a given language. The following results illustrate the ASR algorithm's ability to detect and interpret juvenile vocalizations.

The rating that our algorithm assigned to the output of the ASR is similar to the standardized score that was provided to us by our advisor. For this test, the algorithm used a pre-recorded audio file of an autistic child trying to pronounce the word "teapot". Each audio file is a recording from a different child. The score that was provided to us by our advisor was a number between one and one hundred. The scoring algorithm that was developed rated each output based on the similarity of the spelling of the output to the spelling of the test word. For example, File #1, was rated 9 out of 100 resulting in a score of 0.09. Our algorithm provided outputs that were consistently scored lower than the standardized rating that our advisor provided.



Trials Using pre-Recorded Vocalizations for the word "teapot"			
File #	Standard Rating	ASR output	Algorithm score
File #1	.09	ol	0
File #2	.2	twot ine	0.1667
File #3	.25	wal one	0
File #4	.25	e o	0
File #5	.3	pal toe	0.1667
File #6	1	tee pat	0.667
File #7	1	tee pup	0.5
File #8	1	hee poke	0.5
File #9	1	tea paut	0.667
File #10	1	ehy poch	0.33
File #11	1	inty match	0
File #12	1	heet hat	0.1667

Figure 6: The figure above displays the results from the test for the Automatic Speech Recognition algorithm and score algorithm (ASR). These results demonstrate the ability of the ASR and scoring algorithms to identify juvenile vocalizations in individuals with ASD. For the entirety of the test, twelve different recordings of juvenile autistic children pronouncing the word “teapot” were used to assess the ASR’s ability to decipher these vocalizations. The standard score was determined by our advisor prior to our testing and the “Algorithm score” was the determined during our testing and it was based on the similarity of the spelling of the output and the test word.

Linguistics studying adult phonemic formants, were able to map a clear relationship between the first and second formant frequencies for all English Vowel phonemes and linked these relative values to distinct physical orientations for the tongue and mouth.

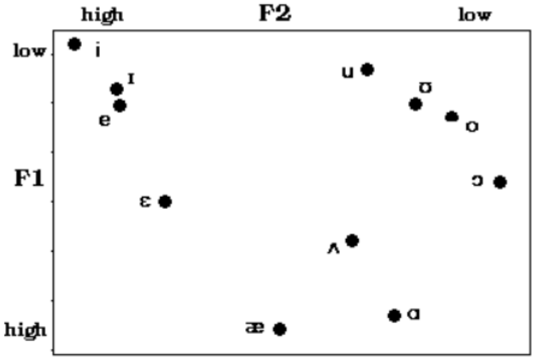


Figure 7. Linguistic Formant Analysis of English Vowel Sounds. International Phonetic Alphabet (IPA) symbols for phonemic English Vowel sounds shown in the plot above. (From University of Manitoba's Phonetics Course Site)[23]

Using the DSP to assess Child Vowel Space in MATLAB, plots were created to elucidate certain regions. F1, corresponds to the position of the tongue in the mouth. High F1 means the tongue is not raised high, or a low vowel, like an "ahhh" sound. A low F1 corresponds to a tongue raised closer to the roof of the mouth, or a linguistic high vowel, "eeeee". For adults the second formant describes vowel frontness/backness; frontness means the sound is projected more outwards, like the "aaah" sound, backness means the sound projection outwards is limited, like the "eeee" sound. ("eeee" corresponds to IPA symbol /i/, and "aaah" to /a/).[23]

The question being addressed in *FmapsChild.m* was, whether or not, there was any similar sort of meaningful relationship that could be deduced from some combination of a F1:F2:F3 plot. A clear relationship between child-formants, would mean correct intelligible utterances correspond to valid formant ranges, ie. analyzing the power spectral density plot of a sound file, would allow for a description of actual physical mechanisms required to create any vowel dominant sound.

The plots outputted from the MATLAB program *FmapsChild.m* show that the approach did not provide the relationships existent for adult phoneme formant frequencies.

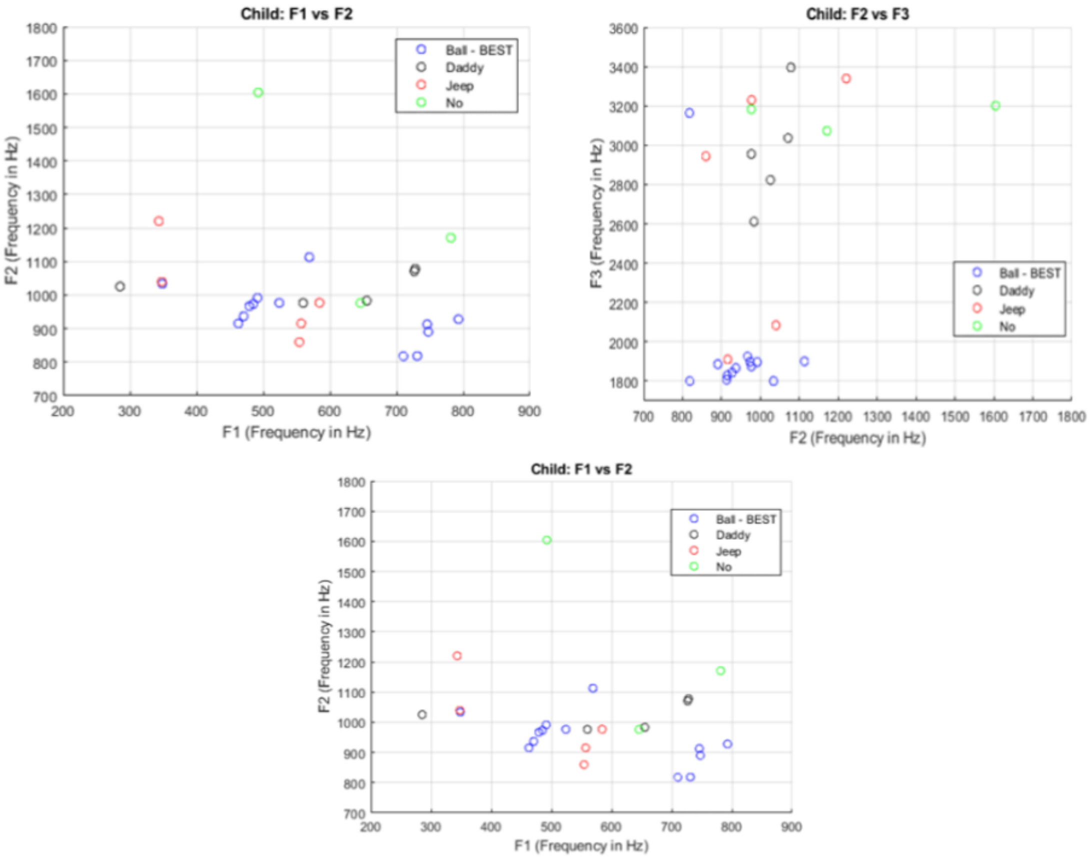


Figure 8. The figure above displays Child 2D - Formant plots created with MATLAB scatter(). No distinct regions exist for any sound category.

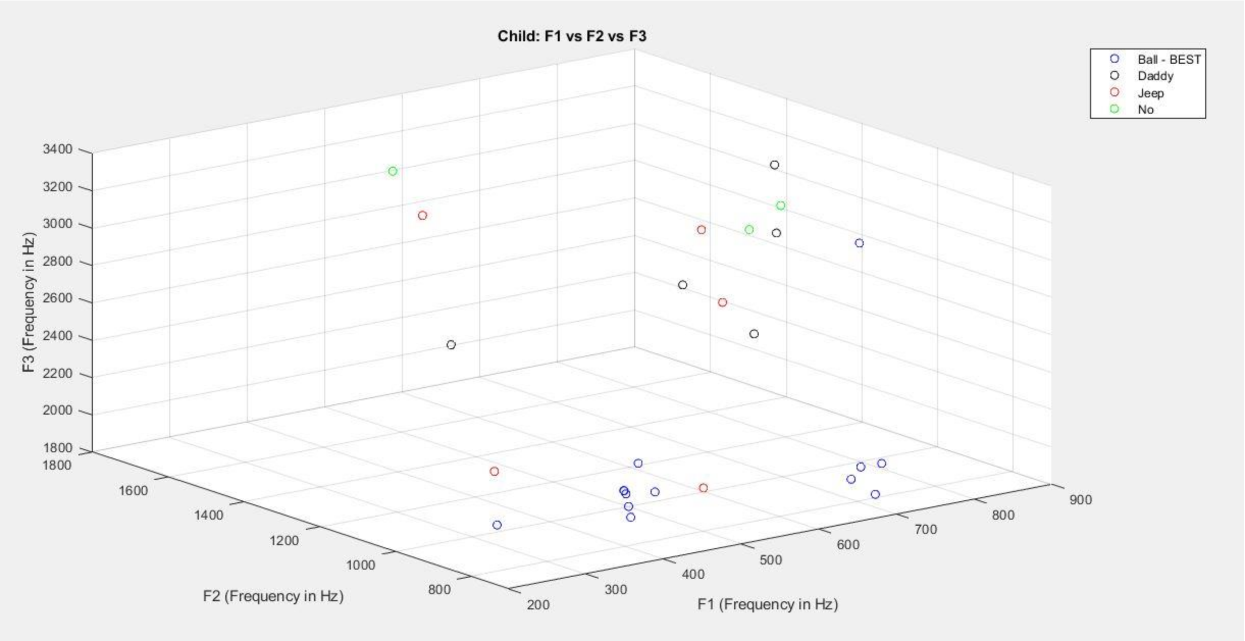


Figure 9. The figure above shows a MATLAB scatter3() used to compare 3 formant sets at once, on an xyz axis.

A successful output of the *FmapsChild.m* would have distinct regions of blue (*Ball*), black (*Daddy*), red(*Jeep)*, green(*No).* The actual program outputs of *FmapsChild.m* have no discernable domains for each word type, implying that the approach's key assumptions need to be reassessed.

The portion of the *.wav* processed, considered the entire sound file; this was done, because it is not possible to retroactively remove consonant sounds to obtain phonemic sounds. The assumption in the program logic, was based on the fact that consonants are fricative and do not have corresponding resonance qualities. While vowels are the core of a word, make up a core of the intensity, they are also shaped by the consonants that surround them. What bare minimum set of linguistic features would be required to more systematically map vowel space is unclear, as the corpus of linguistic research has seriously considered this topic, with little conclusive results either.[24]

In general, there are fewer data points for the Child Formant Plots than data points used in the Adult Formant Plots, because a majority of the original .wav files had too low of an intelligibility score, the utterances of the words sounded more like random noises.

The *FmapsAdult.m* program used *.wav* files from "adults", older children and adults, voices with clear gender differentiation, however, the majority of the sound files were indeed sound files from adults, who sounded to at least 18 years old. The approach used was exactly the same as the approach used for the *FmapsChild.m* program, however, the outputted adult formant estimation plots are more normalized, in the sense that each word has more of a well-defined domain in frequency space. This suggests that there are factors that impact the perception of child vocalizations, which are not factors that impact in adult vocalizations. One obvious factor, which was considered, was maturation of the vocal folds (larynx), this was the reason for the .wav separations.

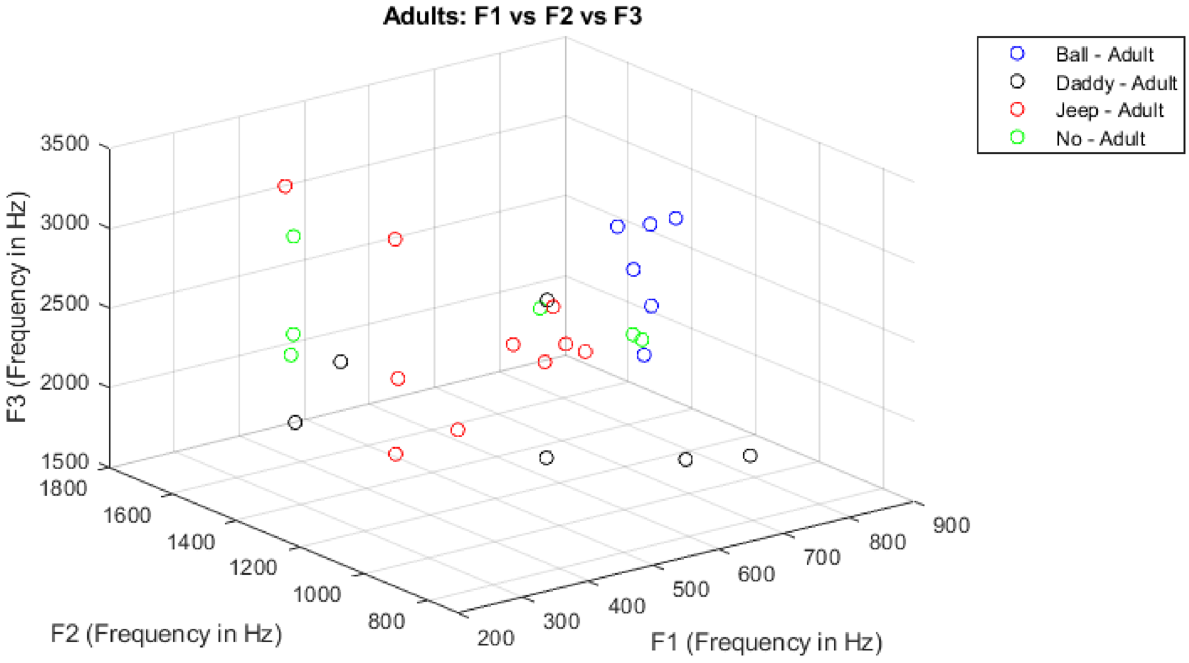


Figure 10. The figure above shows an Adult Formant Analysis plots created with MATLAB scatter(). A clearer relationship between points of the same type, but no well-defined distinct domains.

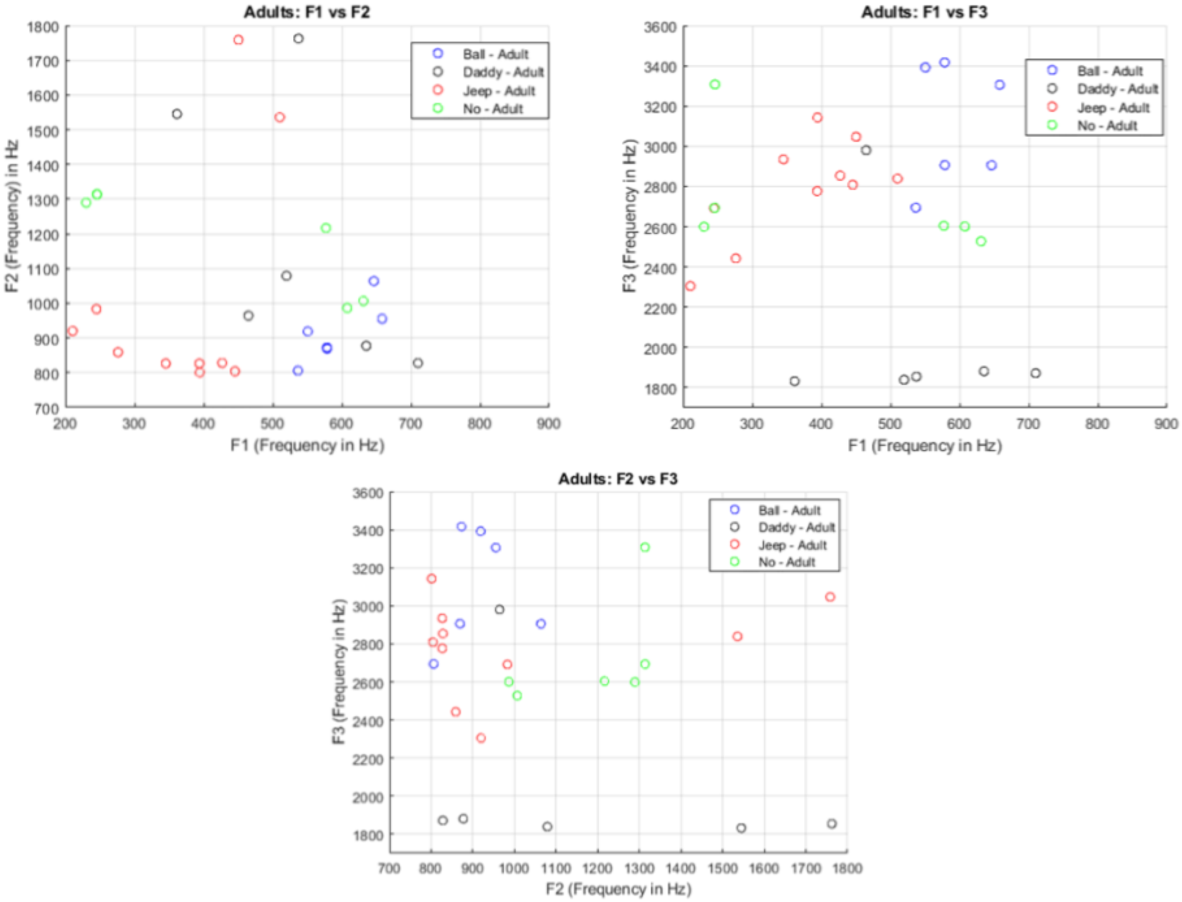
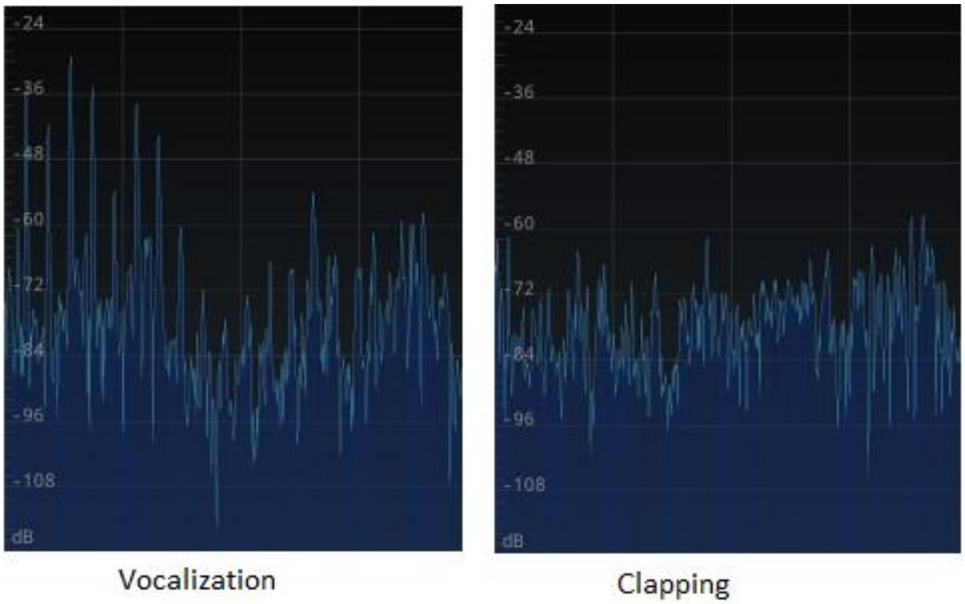


Figure 11. The figure above displays Adult 2D - Formant plots created with MATLAB scatter(). No distinct regions exist for any sound category.

Discussion

<u>AudioAnalyzer and SDK</u>

The first major requirement for success was to have the audio analyzer correctly identify child vocalizations. The five trials performed show that the robot is at least capable of correctly identifying pitched vocalizations. The robot did not react to general noise, such as table slaps or clapping, indicating that the harmonics were some large criteria for identifying vocalization and that high amplitude noise with low spectral density would not qualify as a vocalization. Initially, there was a problem of the robot's reward method featuring a song portion which would be identified as a vocalization. This caused any vocalization noticed by the robot to cause a reward response, which would then cause subsequent reward responses. The initial work around for this problem was simple to make it so that the rewardDog method did not feature any auditory output, instead favoring a 45-degree head turn to indicate a vocalization had been detected. While this workaround was suitable for bug testing purposes on both the reward dog method and vocalization detection algorithm, filtering out these robot reward responses from the audio analyzer would have to happen in order to satisfy future goals of this project to feature a song and dance reward response.



Vocalization

Clapping

Figure 12: This image displays visually the difference between a vocalization, which features resonances and a more detailed transform, versus a high amplitude input which has no distinguishing features outside of an increase in amplitude

The trials performed before the filter implementation mirror the qualitative observations about robot reactions after filter application. Simply, the robot correctly identifies higher pitched voices while ignoring those of the 7 adult voiced with which we tested the filter. Additionally, because of the implementation of the filter is a few lines of code in android studio, it is easily adjustable to fit the user through the programming interface. That being said, no testing was done to further assess what those adjustments would look like. Should the filter be adjusted for children of different sex, for every integer age is there a typical level the filter should be set to? For future iterations of the project, it would be suggested that on app startup the child make some sort of pitched vocalization for the robot to detect what level the filter should be at. Unfortunately, this might present a further problem as the target audience is children with ASD who are on the lower level of vocal.

One important feature for our project supervisor was to develop a Bluetooth auto connect routine that would detect the presence of nearby CHIP robots, and then connect to them and begin the learning curriculum. While these auto connect routines can be implemented easily through standard android studio methods, the WowWee robots have their own set of connection protocols that have to be followed. When testing the auto connect functionality, the robot could be physically turned off or out of range, and be able to auto connect and function properly. This is a pivotal functionality primarily targeted at the users’ end, where they can resume session activity without having to manually reconnect at every disconnection.

One recommendation to future groups would be to user a different robot, or come to an agreement with WowWee about making hardware modifications to the CHIP robot. While the android SDK is a suitable platform for some application development, there are some routines hard programmed into the robot which make it a poor choice for this particular project. For instance, after a brief time of inactivity, we measured it to be between 30 to 90 seconds, the WowWee CHIP robot would bark, or move in a small set of ways. It was assumed that this movement was implemented to have the robot function like an actual dog. For this project specifically, however, those random movements can completely ruin the intended application. For many with ASD, unexpected interruptions to routine or thought process can induce intense anxiety.[19] If during a lesson the CHIP robot were to bark or move around unexpectedly, that could cause the user undue stress. Additionally, unexpected interruptions in thought processes can cause those with ASD to become overloaded or unable to perform to ability in an academic setting.[20] These inhibitions could cause any curriculum or grading system to inaccurately measure the user's ability. Furthermore, being required to use WowWee's Bluetooth protocols means that any further group wanting to develop this project further would have to spend time learning that software very specifically, rather than just being proficient in either Java, Kotlin, or C++.

ASR

The ASR analysis output of our program does not give phonemes, rather it outputs data as a character array of what the programs thinks has been said. There are times that the program will read "Tea Pot" as "tee pat" or "tea paut", etc. These deviations in spelling cause the scoring algorithm to grade differently depending on the character array, rather than the phonemes. This method of grading means that the algorithm will have a lower standard score than a listener might give a grade because each possible variation on a words spelling must be accounted for. As can be seen in **Figure 12**, the algorithm scores do, in fact, have a lower average score, and sometimes are even poorly indicative of a success or failure, in the case of file 12, where the qualitative scoring method gave a perfect score and the algorithm effectively failed the run. In order to circumvent this scoring method, there are various solutions. First, simply make the scoring algorithm more robust. If the target word is a character array of "Tea Pot", then allow for some deviation from that particular spelling. Conditionals, or regex, can be used to give more flexibility in the scoring system. Furthermore, the accepted academic English spelling of a word can vary greatly from its phonetic spelling. Rather than compare the output of the file to the accepted academic English spelling of the target word, each word being graded should be check to see if there is a different, phonetic spelling which should be used. For instance, "Phishing" should be tried against "Fishing", and "Adore" against "Adoor".

DSP to Assess Child Vowel Space (MATLAB)

The intent of trying this approach was to try and assess if there could be a systematic way to use vowel characteristics to organize word sounds into distinct formant frequency regions. The results for the child formant estimations were unclear and inconclusive. A larger data set of child vocalizations, including a wider range of words would be useful in future iterations this research, however, the analysis approach must as well be altered. At the core, the rationale behind the approach was to contrast different word-sounds and determine valid ranges. Adult vocalizations are more normalized, and the adult formant estimation plots for word-sounds were more normalized as well. This suggests that a more informative approach may be to study the variation among what is perceived to be linguistically similar, compare single child word-vocalizations to other child vocalizations of the same exact word, and try to determine what factors actually shift language intelligibility.

References

[1] Autism Speaks. (n.d.). Autism Facts and Figures. Retrieved April 18, 2019, from https://www.autismspeaks.org/autism-facts-and-figures

[2] Boat, Thomas F. “Clinical Characteristics of Autism Spectrum Disorder.” *Mental Disorders and Disabilities Among Low-Income Children.*, U.S. National Library of Medicine, 28 Oct. 2015, www.ncbi.nlm.nih.gov/books/NBK332891

[3] M. Wingate et al., “Prevalence of autism spectrum disorder among children aged 8 years-autism and developmental disabilities monitoring network, 11 sites, United States, 2010,” MMWR Surveill. Summ., vol. 63, no. 2, 2014.

[4] M. Yeargin-Allsopp, C. Rice, T. Karapurkar, N. Doernberg, C. Boyle, and C. Murphy, “Prevalence of autism in a US metropolitan area,” Jama, vol. 289, no. 1, pp. 49–55, 2003.

[5] H. Tager-Flusberg and C. Kasari, “Minimally verbal school-aged children with autism spectrum disorder: the neglected end of the spectrum,” *Autism Res.*, vol. 6, no. 6, pp. 468–478, 2013.

[6] M. A. Maglione, D. Gans, L. Das, J. Timbie, C. Kasari, and others, “Nonmedical interventions for children with ASD: Recommended guidelines and further research needs,” *Pediatrics*, vol. 130, no. Supplement 2, pp. S169–S178, 2012.

[7] Boat, Thomas F. “Clinical Characteristics of Autism Spectrum Disorder.” *Mental Disorders and Disabilities Among Low-Income Children.*, U.S. National Library of Medicine, 28 Oct. 2015, www.ncbi.nlm.nih.gov/books/NBK332891

[8] M. D. Wise, A. A. Little, J. B. Holliman, P. H. Wise, and C. J. Wang, “Can state early intervention programs meet the increased demand of children suspected of having autism spectrum disorders?,” *J. Dev. Behav. Pediatr.*, vol. 31, no. 6, pp. 469–476, 2010

[9] K. Squires, “Addressing the Shortage of Speech-Language Pathologists in School Settings.,” *J. Am. Acad. Spec. Educ. Prof.*, vol. 131, p. 137, 2013.

[10] F. M. Hewett, “Teaching speech to an autistic child through operant conditioning.,” *Am. J. Orthopsychiatry*, vol. 35, no. 5, p. 927, 1965.

[11] O. I. Lovaas, J. P. Berberich, B. F. Perloff, and B. Schaeffer, “Acquisition of imitative speech by schizophrenic children,” *Science*, vol. 151, no. 3711, pp. 705–707, 1966.

[12] O. I. Lovaas, R. Koegel, J. Q. Simmons, and J. S. Long, “Some generalization and follow-up measures on autistic children in behavior therapy,” *J. Appl. Behav. Anal.*, vol. 6, no. 1, pp. 131–165, 1973.

[13] N. R. Marshall and J. R. Hegrenes, “Programmed communication therapy for autistic mentally retarded children.,” *J. Speech Hear. Disord.*, 1970.

[14] G. Picaizen, A. A. Berger, D. Baronofsky, A. C. Nichols, and R. Karen, “Applications of operant techniques to speech therapy with non-verbal children,” *J. Commun. Disord.*, vol. 2, no. 3, pp. 203–211, 1969.

[15] E. J. Sweeney-Kerwin, “Improving the Vocal Production of Echoic Responses for Cildren with Autism Using Echoic Training with Phonological Breakdowns,” Simmons College, 2011.

[16] E. Hyun, S. Kim, S. Jang, and S. Park, “Comparative study of effects of language instruction program using intelligence robot and multimedia on linguistic ability of young children,” in *Robot and Human Interactive Communication, 2008. RO-MAN 2008. The 17th IEEE International Symposium on*, 2008, pp. 187–192.

[17] L. Boccanfuso, S. Scarborough, R. K. Abramson, A. V. Hall, H. H. Wright, and J. M. O’Kane, “A low-cost socially assistive robot and robot-assisted intervention for children with autism spectrum disorder: field trials and lessons learned,” *Auton. Robots*, vol. 41, no. 3, pp. 637–655, 2017.

[18] R. E. Simut, J. Vanderfaeillie, A. Peca, G. Van de Perre, and B. Vanderborght, “Children with autism spectrum disorders make a fruit salad with Probo, the social robot: an interaction study,” *J. Autism Dev. Disord.*, vol. 46, no. 1, pp. 113–126, 2016.

[19] Blakeley-Smith, Audrey, Judy Reaven, Katherine Ridge, and Susan Hepburn. "Parent–child Agreement of Anxiety Symptoms in Youth with Autism Spectrum Disorders." *Research in Autism Spectrum Disorders* 6, no. 2 (2012): 707-16. doi:10.1016/j.rasd.2011.07.020.

[20] Cassella, Megan Duffy, Tina M. Sidener, David W. Sidener, and Patrick R. Progar. "Response Interruption And Redirection For Vocal Stereotypy In Children With Autism: A Systematic Replication." *Journal of Applied Behavior Analysis* 44, no. 1 (2011): 169-73. doi:10.1901/jaba.2011.44-169.

[21] Mavavilj, Mavaviljmavavilj 5471626, GillesGilles 2, & CMDoolittleCMDoolittle 50029. (n.d.). Power spectral density vs. FFT bin magnitude. Retrieved from https://dsp.stackexchange.com/questions/24780/power-spectral-density-vs-fft-bin-magnitude

[22]MathWorks. (n.d.). Periodogram Documentation. Retrieved from https://www.mathworks.com/help/signal/ref/periodogram.html

[23] https://home.cc.umanitoba.ca/~krussll/phonetics/acoustic/formants.html.

[24] https://sail.usc.edu/~lgoldste/General_Phonetics/Consonants/Cb.html

APPENDIX

- **I.** **Android Studio Code**
- **II.** **ASR Code**
- **III.** **DSP to Asses Child Vocalization MatLab Code**

I. CODE FOR ANDROID STUDIOS

Gradle Files:

build.gradle

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.
buildscript {
    repositories {
        jcenter()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
        google()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.3.1'
    }
}

allprojects {
    repositories {
        jcenter()
        maven {
            url 'https://maven.google.com/'
            name 'Google'
        }
    }
}
```

settings.gradle

```
include ':FFTLibrary'
include ':audioSpectrumAnalyzer'
```

gradle-wrapper.properties

```
#Mon Feb 25 16:20:11 EST 2019
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-4.10.1-all.zip
```

FFTLibrary

RealDoubleFFT

```
/* Copyright 2011 Google Inc.
```

```
 *
 *Licensed under the Apache License, Version 2.0 (the "License");
 *you may not use this file except in compliance with the License.
 *You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 *Unless required by applicable law or agreed to in writing, software
 *distributed under the License is distributed on an "AS IS" BASIS,
 *WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *See the License for the specific language governing permissions and
 *limitations under the License.
 *
 * Derived from jffpack, by suhler@google.com.
 *
 * jfftpack is a Java version of fftpack. jfftpack is based
 * on Paul N. Swarztraubre's Fortran code and Pekka Janhuen's
 * C code. It is developed as part of my official duties as
 * lead software engineer for SCUBA-2 FTS projects
 * (www.roe.ac.uk/ukatc/projects/scubatwo/)
 *
 * The original fftpack was public domain, so jfftpack is public domain too.
 * @author Baoshe Zhang
 * @author Astronomical Instrument Group of University of Lethbridge.
 */

package com.google.corp.productivity.specialprojects.android.fft;

public class RealDoubleFFT extends RealDoubleFFT_Mixed {
  /**
   * <em>norm_factor</em> can be used to normalize this FFT transform. This is because
   * a call of forward transform (<em>ft</em>) followed by a call of backward transform
   * (<em>bt</em>) will multiply the input sequence by <em>norm_factor</em>.
   */
  public double norm_factor;
  private double wavetable[];
  private double[] ch;   // reusable work array
  private int ndim;

  /**
   * Construct a wavenumber table with size <em>n</em>.
   * The sequences with the same size can share a wavenumber table. The prime
   * factorization of <em>n</em> together with a tabulation of the trigonometric functions
   * are computed and stored.
   *
   * @param  n  the size of a real data sequence. When <em>n</em> is a multiplication of small
   * numbers (4, 2, 3, 5), this FFT transform is very efficient.
   */
  public RealDoubleFFT(int n)
  {
    ndim = n;
    norm_factor = n;
    if(wavetable == null || wavetable.length !=(2*ndim+15)) {
      wavetable = new double[2*ndim + 15];
    }
    rffti(ndim, wavetable);
    ch = new double[n];
  }

  /**
   * Forward real FFT transform. It computes the discrete transform of a real data sequence.
   *
   * @param x an array which contains the sequence to be transformed. After FFT,
   * <em>x</em> contains the transform coeffients used to construct <em>n</em> complex FFT
coeffients.
   * <br>
   * The real part of the first complex FFT coeffients is <em>x</em>[0]; its imaginary part
   * is 0. If <em>n</em> is even set <em>m</em> = <em>n</em>/2, if <em>n</em> is odd set
   * <em>m</em> = <em>n</em>/2, then for
   * <br>
   * <em>k</em> = 1, ..., <em>m</em>-1 <br>
   * the real part of <em>k</em>-th complex FFT coeffients is <em>x</em>[2*<em>k</em>-1];
   * <br>
   * the imaginary part of <em>k</em>-th complex FFT coeffients is
<em>x</em>[2*<em>k</em>-2].
   * <br>
   * If <em>n</em> is even,
   * the real of part of (<em>n</em>/2)-th complex FFT coeffients is
<em>x</em>[<em>n</em>]; its imaginary part is 0.
   * The remaining complex FFT coeffients can be obtained by the symmetry relation:
   * the (<em>n</em>-<em>k</em>)-th complex FFT coeffient is the conjugate of <em>n</em>-
th complex FFT coeffient.
   *
   */
  public void ft(double x[]) {
    if(x.length != ndim)
      throw new IllegalArgumentException("The length of data can not match that of the
wavetable");
```

```
    rfftf(ndim, x, wavetable, ch);
  }
}
```

RealDoubleFFT_Mixed

```
/* Copyright 2011 Google Inc.
 *
 *Licensed under the Apache License, Version 2.0 (the "License");
 *you may not use this file except in compliance with the License.
 *You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 *Unless required by applicable law or agreed to in writing, software
 *distributed under the License is distributed on an "AS IS" BASIS,
 *WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *See the License for the specific language governing permissions and
 *limitations under the License.
 *
 * Derived from jffpack, by suhler@google.com.
 *
 * jfftpack is a Java version of fftpack. jfftpack is based
 * on Paul N. Swarztraubre's Fortran code and Pekka Janhuen's
 * C code. It is developed as part of my official duties as
 * lead software engineer for SCUBA-2 FTS projects
 * (www.roe.ac.uk/ukatc/projects/scubatwo/)
 *
 * The original fftpack was public domain, so jfftpack is public domain too.
 * @author Baoshe Zhang
 * @author Astronomical Instrument Group of University of Lethbridge.
 */
package com.google.corp.productivity.specialprojects.android.fft;

class RealDoubleFFT_Mixed {
  static final int[] ntryh= new int[] {4, 2, 3, 5};

  /*-------------------------------------------------
   radf2: Real FFT's forward processing of factor 2
  -------------------------------------------------*/
  void radf2(int ido, int l1, final double cc[], double ch[],
           final double wtable[], int offset) {
    int     i, k, ic;
    double  ti2, tr2;
    int iw1;
    iw1 = offset;

    for(k=0; k<l1; k++) {
      ch[2*k*ido]=cc[k*ido]+cc[(k+l1)*ido];
      ch[(2*k+1)*ido+ido-1]=cc[k*ido]-cc[(k+l1)*ido];
    }
    if(ido<2) return;
    if(ido !=2) {
      for(k=0; k<l1; k++) {
        for(i=2; i<ido; i+=2) {
          ic=ido-i;
          tr2 = wtable[i-2+iw1]*cc[i-1+(k+l1)*ido]
                                   +wtable[i-1+iw1]*cc[i+(k+l1)*ido];
          ti2 = wtable[i-2+iw1]*cc[i+(k+l1)*ido]
                                   -wtable[i-1+iw1]*cc[i-1+(k+l1)*ido];
          ch[i+2*k*ido]=cc[i+k*ido]+ti2;
          ch[ic+(2*k+1)*ido]=ti2-cc[i+k*ido];
          ch[i-1+2*k*ido]=cc[i-1+k*ido]+tr2;
          ch[ic-1+(2*k+1)*ido]=cc[i-1+k*ido]-tr2;
        }
      }
      if(ido%2==1)return;
    }
    for(k=0; k<l1; k++) {
      ch[(2*k+1)*ido]=-cc[ido-1+(k+l1)*ido];
      ch[ido-1+2*k*ido]=cc[ido-1+k*ido];
    }
  }

  /*-------------------------------------------------
   radf3: Real FFT's forward processing of factor 3
  -------------------------------------------------*/
  void radf3(int ido, int l1, final double cc[], double ch[],
           final double wtable[], int offset) {
    final double taur=-0.5D;
    final double taui=0.866025403784439D;
    int     i, k, ic;
    double  ci2, di2, di3, cr2, dr2, dr3, ti2, ti3, tr2, tr3;
    int iw1, iw2;
    iw1 = offset;
    iw2 = iw1 + ido;

    for(k=0; k<l1; k++) {
      cr2=cc[(k+l1)*ido]+cc[(k+2*l1)*ido];
      ch[3*k*ido]=cc[k*ido]+cr2;
      ch[(3*k+2)*ido]=taui*(cc[(k+l1*2)*ido]-cc[(k+l1)*ido]);
      ch[ido-1+(3*k+1)*ido]=cc[k*ido]+taur*cr2;
    }
    if(ido==1) return;
    for(k=0; k<l1; k++) {
      for(i=2; i<ido; i+=2) {
        ic=ido-i;
        dr2 = wtable[i-2+iw1]*cc[i-1+(k+l1)*ido]
                             +wtable[i-1+iw1]*cc[i+(k+l1)*ido];
        di2 = wtable[i-2+iw1]*cc[i+(k+l1)*ido]
                             -wtable[i-1+iw1]*cc[i-1+(k+l1)*ido];
        dr3 = wtable[i-2+iw2]*cc[i-1+(k+l1*2)*ido]
                             +wtable[i-1+iw2]*cc[i+(k+l1*2)*ido];
        di3 = wtable[i-2+iw2]*cc[i+(k+l1*2)*ido]
                             -wtable[i-1+iw2]*cc[i-1+(k+l1*2)*ido];
        cr2 = dr2+dr3;
        ci2 = di2+di3;
        ch[i-1+3*k*ido]=cc[i-1+k*ido]+cr2;
        ch[i+3*k*ido]=cc[i+k*ido]+ci2;
        tr2=cc[i-1+k*ido]+taur*cr2;
        ti2=cc[i+k*ido]+taur*ci2;
        tr3=taui*(di2-di3);
        ti3=taui*(dr3-dr2);
        ch[i-1+(3*k+2)*ido]=tr2+tr3;
        ch[ic-1+(3*k+1)*ido]=tr2-tr3;
        ch[i+(3*k+2)*ido]=ti2+ti3;
        ch[ic+(3*k+1)*ido]=ti3-ti2;
      }
    }
  }

  /*-------------------------------------------------
   radf4: Real FFT's forward processing of factor 4
  -------------------------------------------------*/
  void radf4(int ido, int l1, final double cc[], double ch[],
           final double wtable[], int offset) {
    final double hsqt2=0.7071067811865475D;
    int i, k, ic;
    double  ci2, ci3, ci4, cr2, cr3, cr4, ti1, ti2, ti3, ti4, tr1, tr2, tr3, tr4;
    int iw1, iw2, iw3;
    iw1 = offset;
    iw2 = offset + ido;
```

```
iw3 = iw2 + ido;
for(k=0; k<l1; k++) {
  tr1=cc[(k+l1)*ido]+cc[(k+3*l1)*ido];
  tr2=cc[k*ido]+cc[(k+2*l1)*ido];
  ch[4*k*ido]=tr1+tr2;
  ch[ido-1+(4*k+3)*ido]=tr2-tr1;
  ch[ido-1+(4*k+1)*ido]=cc[k*ido]-cc[(k+2*l1)*ido];
  ch[(4*k+2)*ido]=cc[(k+3*l1)*ido]-cc[(k+l1)*ido];
}
if(ido<2) return;
if(ido !=2) {
  for(k=0; k<l1; k++) {
    for(i=2; i<ido; i+=2) {
      ic=ido-i;
      cr2 = wtable[i-2+iw1]*cc[i-1+(k+l1)*ido]
                              +wtable[i-1+iw1]*cc[i+(k+l1)*ido];
      ci2 = wtable[i-2+iw1]*cc[i+(k+l1)*ido]
                              -wtable[i-1+iw1]*cc[i-1+(k+l1)*ido];
      cr3 = wtable[i-2+iw2]*cc[i-1+(k+2*l1)*ido]
                              +wtable[i-1+iw2]*cc[i+(k+2*l1)*ido];
      ci3 = wtable[i-2+iw2]*cc[i+(k+2*l1)*ido]
                              -wtable[i-1+iw2]*cc[i-1+(k+2*l1)*ido];
      cr4 = wtable[i-2+iw3]*cc[i-1+(k+3*l1)*ido]
                              +wtable[i-1+iw3]*cc[i+(k+3*l1)*ido];
      ci4 = wtable[i-2+iw3]*cc[i+(k+3*l1)*ido]
                              -wtable[i-1+iw3]*cc[i-1+(k+3*l1)*ido];
      tr1=cr2+cr4;
      tr4=cr4-cr2;
      ti1=ci2+ci4;
      ti4=ci2-ci4;
      ti2=cc[i+k*ido]+ci3;
      ti3=cc[i+k*ido]-ci3;
      tr2=cc[i-1+k*ido]+cr3;
      tr3=cc[i-1+k*ido]-cr3;
      ch[i-1+4*k*ido]=tr1+tr2;
      ch[ic-1+(4*k+3)*ido]=tr2-tr1;
      ch[i+4*k*ido]=ti1+ti2;
      ch[ic+(4*k+3)*ido]=ti1-ti2;
      ch[i-1+(4*k+2)*ido]=ti4+tr3;
      ch[ic-1+(4*k+1)*ido]=tr3-ti4;
      ch[i+(4*k+2)*ido]=tr4+ti3;
      ch[ic+(4*k+1)*ido]=tr4-ti3;
    }
  }
```

```
    if(ido%2==1) return;
  }
  for(k=0; k<l1; k++) {
    ti1=-hsqt2*(cc[ido-1+(k+l1)*ido]+cc[ido-1+(k+3*l1)*ido]);
    tr1=hsqt2*(cc[ido-1+(k+l1)*ido]-cc[ido-1+(k+3*l1)*ido]);
    ch[ido-1+4*k*ido]=tr1+cc[ido-1+k*ido];
    ch[ido-1+(4*k+2)*ido]=cc[ido-1+k*ido]-tr1;
    ch[(4*k+1)*ido]=ti1-cc[ido-1+(k+2*l1)*ido];
    ch[(4*k+3)*ido]=ti1+cc[ido-1+(k+2*l1)*ido];
  }
}

/*-------------------------------------------------
 radf5: Real FFT's forward processing of factor 5
-------------------------------------------------*/
void radf5(int ido, int l1, final double cc[], double ch[],
           final double wtable[], int offset) {
  final double tr11=0.309016994374947D;
  final double ti11=0.951056516295154D;
  final double tr12=-0.809016994374947D;
  final double ti12=0.587785252292473D;
  int     i, k, ic;
  double  ci2, di2, ci4, ci5, di3, di4, di5, ci3, cr2, cr3, dr2, dr3,
  dr4, dr5, cr5, cr4, ti2, ti3, ti5, ti4, tr2, tr3, tr4, tr5;
  int iw1, iw2, iw3, iw4;
  iw1 = offset;
  iw2 = iw1 + ido;
  iw3 = iw2 + ido;
  iw4 = iw3 + ido;

  for(k=0; k<l1; k++) {
    cr2=cc[(k+4*l1)*ido]+cc[(k+l1)*ido];
    ci5=cc[(k+4*l1)*ido]-cc[(k+l1)*ido];
    cr3=cc[(k+3*l1)*ido]+cc[(k+2*l1)*ido];
    ci4=cc[(k+3*l1)*ido]-cc[(k+2*l1)*ido];
    ch[5*k*ido]=cc[k*ido]+cr2+cr3;
    ch[ido-1+(5*k+1)*ido]=cc[k*ido]+tr11*cr2+tr12*cr3;
    ch[(5*k+2)*ido]=ti11*ci5+ti12*ci4;
    ch[ido-1+(5*k+3)*ido]=cc[k*ido]+tr12*cr2+tr11*cr3;
    ch[(5*k+4)*ido]=ti12*ci5-ti11*ci4;
  }
  if(ido==1) return;
  for(k=0; k<l1;++k) {
    for(i=2; i<ido; i+=2) {
```

```
  ic=ido-i;
  dr2 = wtable[i-2+iw1]*cc[i-1+(k+l1)*ido]
                      +wtable[i-1+iw1]*cc[i+(k+l1)*ido];
  di2 = wtable[i-2+iw1]*cc[i+(k+l1)*ido]
                      -wtable[i-1+iw1]*cc[i-1+(k+l1)*ido];
  dr3 = wtable[i-2+iw2]*cc[i-1+(k+2*l1)*ido]
                      +wtable[i-1+iw2]*cc[i+(k+2*l1)*ido];
  di3 = wtable[i-2+iw2]*cc[i+(k+2*l1)*ido]
                      -wtable[i-1+iw2]*cc[i-1+(k+2*l1)*ido];
  dr4 = wtable[i-2+iw3]*cc[i-1+(k+3*l1)*ido]
                      +wtable[i-1+iw3]*cc[i+(k+3*l1)*ido];
  di4 = wtable[i-2+iw3]*cc[i+(k+3*l1)*ido]
                      -wtable[i-1+iw3]*cc[i-1+(k+3*l1)*ido];
  dr5 = wtable[i-2+iw4]*cc[i-1+(k+4*l1)*ido]
                      +wtable[i-1+iw4]*cc[i+(k+4*l1)*ido];
  di5 = wtable[i-2+iw4]*cc[i+(k+4*l1)*ido]
                      -wtable[i-1+iw4]*cc[i-1+(k+4*l1)*ido];
  cr2=dr2+dr5;
  ci5=dr5-dr2;
  cr5=di2-di5;
  ci2=di2+di5;
  cr3=dr3+dr4;
  ci4=dr4-dr3;
  cr4=di3-di4;
  ci3=di3+di4;
  ch[i-1+5*k*ido]=cc[i-1+k*ido]+cr2+cr3;
  ch[i+5*k*ido]=cc[i+k*ido]+ci2+ci3;
  tr2=cc[i-1+k*ido]+tr11*cr2+tr12*cr3;
  ti2=cc[i+k*ido]+tr11*ci2+tr12*ci3;
  tr3=cc[i-1+k*ido]+tr12*cr2+tr11*cr3;
  ti3=cc[i+k*ido]+tr12*ci2+tr11*ci3;
  tr5=ti11*cr5+ti12*cr4;
  ti5=ti11*ci5+ti12*ci4;
  tr4=ti12*cr5-ti11*cr4;
  ti4=ti12*ci5-ti11*ci4;
  ch[i-1+(5*k+2)*ido]=tr2+tr5;
  ch[ic-1+(5*k+1)*ido]=tr2-tr5;
  ch[i+(5*k+2)*ido]=ti2+ti5;
  ch[ic+(5*k+1)*ido]=ti5-ti2;
  ch[i-1+(5*k+4)*ido]=tr3+tr4;
  ch[ic-1+(5*k+3)*ido]=tr3-tr4;
  ch[i+(5*k+4)*ido]=ti3+ti4;
  ch[ic+(5*k+3)*ido]=ti4-ti3;
}
```

```
  }
}

/*-------------------------------------------------------
 radfg: Real FFT's forward processing of general factor
-------------------------------------------------------*/
void radfg(int ido, int ip, int l1, int idl1, double cc[],
           double c1[], double c2[], double ch[], double ch2[],
           final double wtable[], int offset) {
  final double twopi=2.0D*Math.PI; //6.28318530717959;
  int     idij, ipph, i, j, k, l, j2, ic, jc, lc, ik, is, nbd;
  double  dc2, ai1, ai2, ar1, ar2, ds2, dcp, arg, dsp, ar1h, ar2h;
  int iw1 = offset;

  arg=twopi / ip;
  dcp=Math.cos(arg);
  dsp=Math.sin(arg);
  ipph=(ip+1)/ 2;
  nbd=(ido-1)/ 2;
  if(ido !=1) {
    for(ik=0; ik<idl1; ik++) ch2[ik]=c2[ik];
    for(j=1; j<ip; j++)
      for(k=0; k<l1; k++)
        ch[(k+j*l1)*ido]=c1[(k+j*l1)*ido];
    if(nbd<=l1) {
      is=-ido;
      for(j=1; j<ip; j++) {
        is+=ido;
        idij=is-1;
        for(i=2; i<ido; i+=2) {
          idij+=2;
          for(k=0; k<l1; k++) {
            ch[i-1+(k+j*l1)*ido]=
              wtable[idij-1+iw1]*c1[i-1+(k+j*l1)*ido]
                                    +wtable[idij+iw1]*c1[i+(k+j*l1)*ido];
            ch[i+(k+j*l1)*ido]=
              wtable[idij-1+iw1]*c1[i+(k+j*l1)*ido]
                                    -wtable[idij+iw1]*c1[i-1+(k+j*l1)*ido];
          }
        }
      }
    } else {
      is=-ido;
      for(j=1; j<ip; j++) {
```

```
      is+=ido;
      for(k=0; k<l1; k++) {
        idij=is-1;
        for(i=2; i<ido; i+=2) {
          idij+=2;
          ch[i-1+(k+j*l1)*ido]=
            wtable[idij-1+iw1]*c1[i-1+(k+j*l1)*ido]
                                  +wtable[idij+iw1]*c1[i+(k+j*l1)*ido];
          ch[i+(k+j*l1)*ido]=
            wtable[idij-1+iw1]*c1[i+(k+j*l1)*ido]
                                  -wtable[idij+iw1]*c1[i-1+(k+j*l1)*ido];
        }
      }
    }
  }
  if(nbd>=l1) {
    for(j=1; j<ipph; j++) {
      jc=ip-j;
      for(k=0; k<l1; k++) {
        for(i=2; i<ido; i+=2) {
          c1[i-1+(k+j*l1)*ido]=ch[i-1+(k+j*l1)*ido]+ch[i-1+(k+jc*l1)*ido];
          c1[i-1+(k+jc*l1)*ido]=ch[i+(k+j*l1)*ido]-ch[i+(k+jc*l1)*ido];
          c1[i+(k+j*l1)*ido]=ch[i+(k+j*l1)*ido]+ch[i+(k+jc*l1)*ido];
          c1[i+(k+jc*l1)*ido]=ch[i-1+(k+jc*l1)*ido]-ch[i-1+(k+j*l1)*ido];
        }
      }
    }
  } else {
    for(j=1; j<ipph; j++) {
      jc=ip-j;
      for(i=2; i<ido; i+=2) {
        for(k=0; k<l1; k++) {
          c1[i-1+(k+j*l1)*ido]=
            ch[i-1+(k+j*l1)*ido]+ch[i-1+(k+jc*l1)*ido];
          c1[i-1+(k+jc*l1)*ido]=ch[i+(k+j*l1)*ido]-ch[i+(k+jc*l1)*ido];
          c1[i+(k+j*l1)*ido]=ch[i+(k+j*l1)*ido]+ch[i+(k+jc*l1)*ido];
          c1[i+(k+jc*l1)*ido]=ch[i-1+(k+jc*l1)*ido]-ch[i-1+(k+j*l1)*ido];
        }
      }
    }
  }
} else {
  for(ik=0; ik<idl1; ik++) c2[ik]=ch2[ik];
}
```

```
for(j=1; j<ipph; j++) {
  jc=ip-j;
  for(k=0; k<l1; k++) {
    c1[(k+j*l1)*ido]=ch[(k+j*l1)*ido]+ch[(k+jc*l1)*ido];
    c1[(k+jc*l1)*ido]=ch[(k+jc*l1)*ido]-ch[(k+j*l1)*ido];
  }
}

ar1=1;
ai1=0;
for(l=1; l<ipph; l++) {
  lc=ip-l;
  ar1h=dcp*ar1-dsp*ai1;
  ai1=dcp*ai1+dsp*ar1;
  ar1=ar1h;
  for(ik=0; ik<idl1; ik++) {
    ch2[ik+l*idl1]=c2[ik]+ar1*c2[ik+idl1];
    ch2[ik+lc*idl1]=ai1*c2[ik+(ip-1)*idl1];
  }
  dc2=ar1;
  ds2=ai1;
  ar2=ar1;
  ai2=ai1;
  for(j=2; j<ipph; j++) {
    jc=ip-j;
    ar2h=dc2*ar2-ds2*ai2;
    ai2=dc2*ai2+ds2*ar2;
    ar2=ar2h;
    for(ik=0; ik<idl1; ik++) {
      ch2[ik+l*idl1]+=ar2*c2[ik+j*idl1];
      ch2[ik+lc*idl1]+=ai2*c2[ik+jc*idl1];
    }
  }
}
for(j=1; j<ipph; j++)
  for(ik=0; ik<idl1; ik++)
    ch2[ik]+=c2[ik+j*idl1];

if(ido>=l1) {
  for(k=0; k<l1; k++) {
    for(i=0; i<ido; i++) {
      cc[i+k*ip*ido]=ch[i+k*ido];
    }
  }
```

```
} else {
  for(i=0; i<ido; i++) {
    for(k=0; k<l1; k++) {
      cc[i+k*ip*ido]=ch[i+k*ido];
    }
  }
}
for(j=1; j<ipph; j++) {
  jc=ip-j;
  j2=2*j;
  for(k=0; k<l1; k++) {
    cc[ido-1+(j2-1+k*ip)*ido]=ch[(k+j*l1)*ido];
    cc[(j2+k*ip)*ido]=ch[(k+jc*l1)*ido];
  }
}
if(ido==1) return;
if(nbd>=l1) {
  for(j=1; j<ipph; j++) {
    jc=ip-j;
    j2=2*j;
    for(k=0; k<l1; k++) {
      for(i=2; i<ido; i+=2) {
        ic=ido-i;
        cc[i-1+(j2+k*ip)*ido]=ch[i-1+(k+j*l1)*ido]+ch[i-1+(k+jc*l1)*ido];
        cc[ic-1+(j2-1+k*ip)*ido]=ch[i-1+(k+j*l1)*ido]-ch[i-1+(k+jc*l1)*ido];
        cc[i+(j2+k*ip)*ido]=ch[i+(k+j*l1)*ido]+ch[i+(k+jc*l1)*ido];
        cc[ic+(j2-1+k*ip)*ido]=ch[i+(k+jc*l1)*ido]-ch[i+(k+j*l1)*ido];
      }
    }
  }
} else {
  for(j=1; j<ipph; j++) {
    jc=ip-j;
    j2=2*j;
    for(i=2; i<ido; i+=2) {
      ic=ido-i;
      for(k=0; k<l1; k++) {
        cc[i-1+(j2+k*ip)*ido]=ch[i-1+(k+j*l1)*ido]+ch[i-1+(k+jc*l1)*ido];
        cc[ic-1+(j2-1+k*ip)*ido]=ch[i-1+(k+j*l1)*ido]-ch[i-1+(k+jc*l1)*ido];
        cc[i+(j2+k*ip)*ido]=ch[i+(k+j*l1)*ido]+ch[i+(k+jc*l1)*ido];
        cc[ic+(j2-1+k*ip)*ido]=ch[i+(k+jc*l1)*ido]-ch[i+(k+j*l1)*ido];
      }
    }
  }
  }
}

/*---------------------------------------------------------
 rfftf1: further processing of Real forward FFT
--------------------------------------------------------*/
// NOTE: ch must be preallocated to size n
void rfftf1(int n, double c[], final double wtable[], int offset, double[] ch) {
  int     i;
  int     k1, l1, l2, na, kh, nf, ip, iw, ido, idl1;

  System.arraycopy(wtable, offset, ch, 0, n);

  nf=(int)wtable[1+2*n+offset];
  na=1;
  l2=n;
  iw=n-1+n+offset;
  for(k1=1; k1<=nf;++k1) {
    kh=nf-k1;
    ip=(int)wtable[kh+2+2*n+offset];
    l1=l2 / ip;
    ido=n / l2;
    idl1=ido*l1;
    iw-=(ip-1)*ido;
    na=1-na;
    if(ip==4) {
      if(na==0) {
        radf4(ido, l1, c, ch, wtable, iw);
      } else {
        radf4(ido, l1, ch, c, wtable, iw);
      }
    } else if(ip==2) {
      if(na==0) {
        radf2(ido, l1, c, ch, wtable, iw);
      } else {
        radf2(ido, l1, ch, c, wtable, iw);
      }
    } else if(ip==3) {
      if(na==0) {
        radf3(ido, l1, c, ch, wtable, iw);
      } else {
        radf3(ido, l1, ch, c, wtable, iw);
      }
    } else if(ip==5) {
      if(na==0) {
        radf5(ido, l1, c, ch, wtable, iw);
      } else {
        radf5(ido, l1, ch, c, wtable, iw);
      }
    } else {
      if(ido==1) na=1-na;
      if(na==0) {
        radfg(ido, ip, l1, idl1, c, c, c, ch, ch, wtable, iw);
        na=1;
      } else {
        radfg(ido, ip, l1, idl1, ch, ch, ch, c, c, wtable, iw);
        na=0;
      }
    }
    l2=l1;
  }
  if(na==1) return;
  for(i=0; i<n; i++) c[i]=ch[i];
}

/*---------------------------------------------------------
 rfftf: Real forward FFT
---------------------------------------------------------*/
void rfftf(int n, double r[], double wtable[], double[] ch) {
  if(n==1) return;
  rfftf1(n, r, wtable, 0, ch);
}        /*rfftf*/

/*---------------------------------------------------------
 rffti1: further initialization of Real FFT
---------------------------------------------------------*/
void rffti1(int n, double wtable[], int offset) {

  final double twopi=2.0D*Math.PI;
  double  argh;
  int     ntry=0, i, j;
  double  argld;
  int     k1, l1, l2, ib;
  double  fi;
  int     ld, ii, nf, ip, nl, is, nq, nr;
  double  arg;
  int     ido, ipm;
  int     nfm1;
```

```
nl=n;
nf=0;
j=0;

factorize_loop:
  while(true) {
    ++j;
    if(j<=4)
      ntry=ntryh[j-1];
    else
      ntry+=2;
    do {
      nq=nl / ntry;
      nr=nl-ntry*nq;
      if(nr !=0) continue factorize_loop;
      ++nf;
      wtable[nf+1+2*n+offset]=ntry;

      nl=nq;
      if(ntry==2 && nf !=1) {
        for(i=2; i<=nf; i++) {
          ib=nf-i+2;
          wtable[ib+1+2*n+offset]=wtable[ib+2*n+offset];
        }
        wtable[2+2*n+offset]=2;
      }
    } while(nl !=1);
    break factorize_loop;
  }
wtable[0+2*n+offset] = n;
wtable[1+2*n+offset] = nf;
argh=twopi /(n);
is=0;
nfm1=nf-1;
l1=1;
if(nfm1==0) return;
for(k1=1; k1<=nfm1; k1++) {
  ip=(int)wtable[k1+1+2*n+offset];
  ld=0;
  l2=l1*ip;
  ido=n / l2;
  ipm=ip-1;
  for(j=1; j<=ipm;++j) {
        ld+=l1;
        i=is;
        argld=ld*argh;

        fi=0;
        for(ii=3; ii<=ido; ii+=2) {
          i+=2;
          fi+=1;
          arg=fi*argld;
          wtable[i-2+n+offset] = Math.cos(arg);
          wtable[i-1+n+offset] = Math.sin(arg);
        }
        is+=ido;
      }
      l1=l2;
    }
  } /*rffti1*/

  /*---------------------------------------------------------
   rffti: Initialization of Real FFT
  --------------------------------------------------------*/
  void rffti(int n, double wtable[])  /* length of wtable = 2*n + 15 */ {
    if(n==1) return;
    rffti1(n, wtable, 0);
  } /*rffti*/
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.google.corp.productivity.specialprojects.android.fft"
      android:versionCode="1"
      android:versionName="1.0">
    <!--<uses-sdk android:minSdkVersion="7" />-->
</manifest>
```

audioSpectrumAnalyzer
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="github.bewantbe.audio_analyzer_for_android"
    android:versionCode="20180111"
```

```
android:versionName="1.7.0alpha"
android:installLocation="auto">

<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />


<application
    tools:replace="android:icon"
    android:allowBackup="true"
    android:fullBackupContent="@xml/backup_descriptor"
    android:icon="@drawable/icon"
    android:label="@string/app_name"
    android:theme="@style/DarkThemeSelector" >
    <activity
        android:name="github.bewantbe.audio_analyzer_for_android.AnalyzerActivity"
        android:label="@string/app_name" >
    </activity>
    <activity
        android:name="github.bewantbe.audio_analyzer_for_android.MyPreferences"
        android:label="@string/preferences" >
    </activity>
    <activity
        android:name=".toDogActivity">
            <intent-filter>
             <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
        <intent-filter>
        <!-- Deeplink support -->
        <!-- https://developer.android.com/training/app-indexing/deep-linking.html -->
        <!-- Can start app by -->
        <!-- adb shell am start -W -a android.intent.action.VIEW -d "bewantbe://asa" -->
              <action android:name="android.intent.action.VIEW" />
              <category android:name="android.intent.category.DEFAULT" />
              <category android:name="android.intent.category.BROWSABLE" />
              <data android:scheme="bewantbe" android:host="asa" />
         </intent-filter>
    </activity>
    <activity
```

```
            android:name="github.bewantbe.audio_analyzer_for_android.InfoRecActivity"
            android:label="@string/title_activity_info_rec"

android:parentActivityName="github.bewantbe.audio_analyzer_for_android.AnalyzerActivity" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="github.bewantbe.audio_analyzer_for_android.AnalyzerActivity" />
        </activity>
    </application>

</manifest>
```

Java

Alert Dialog

```
package github.bewantbe.audio_analyzer_for_android;

import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v7.app.AppCompatDialogFragment;

public class AlertDialog extends AppCompatDialogFragment {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState){
        android.support.v7.app.AlertDialog.Builder builder = new
android.support.v7.app.AlertDialog.Builder(getActivity());
        builder.setTitle("Alert!")
                .setMessage("Disconnected from Robot")
                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialogInterface, int i) {

                    }
                });
        return builder.create();
    }


}
```

AnalyzerActivity

```
/* Copyright 2011 Google Inc.
 *
 *Licensed under the Apache License, Version 2.0 (the "License");
 *you may not use this file except in compliance with the License.
 *You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 *Unless required by applicable law or agreed to in writing, software
 *distributed under the License is distributed on an "AS IS" BASIS,
 *WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *See the License for the specific language governing permissions and
 *limitations under the License.
 *
 * @author Stephen Uhler
 *
 * 2014 Eddy Xiao <bewantbe@gmail.com>
 * GUI extensively modified.
 * Add spectrogram plot, smooth gesture view control and various settings.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.Manifest;
import android.app.Activity;
import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.content.res.Resources;
import android.net.Uri;
import android.os.Bundle;
import android.os.Handler;
import android.os.SystemClock;
import android.preference.PreferenceManager;
import android.renderscript.Sampler;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.app.FragmentManager;
import android.support.v4.content.ContextCompat;
import android.support.v4.view.GestureDetectorCompat;
import android.text.InputType;
import android.util.Log;
import android.view.GestureDetector;
```

```java
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.View.OnLongClickListener;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.webkit.MimeTypeMap;
import android.widget.Button;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.util.Random;
import java.lang.String;

import com.wowwee.bluetoothrobotcontrollib.BluetoothRobot;
import com.wowwee.bluetoothrobotcontrollib.chip.ChipCommandValues;
import com.wowwee.bluetoothrobotcontrollib.chip.ChipRobot;
import com.wowwee.bluetoothrobotcontrollib.chip.ChipRobotFinder;

/**
 * Audio "FFT" analyzer.
 * @author suhler@google.com (Stephen Uhler)
 */


public class AnalyzerActivity extends Activity          // AnalyzerActivity is directly linked to
AnalyzerGraphic.Readyghbnnn
        implements OnLongClickListener, OnClickListener,
        OnItemClickListener, AnalyzerGraphic.Ready
{
    private static final String TAG="AnalyzerActivity:";


    AnalyzerViews analyzerViews;                        // Also direct depencency on AnalyzerViews
    SamplingLoop samplingThread = null;
    private RangeViewDialogC rangeViewDialogC;
    private GestureDetectorCompat mDetector;
    private String m_Text = "";
```

```
toDogActivity analyzerToDog = new toDogActivity();

private AnalyzerParameters analyzerParam = null;    // Same with params
String currentIndicator;

double dtRMS = 0;
double dtRMSFromFT = 0;
double maxAmpDB;
double maxAmpFreq;
double[] viewRangeArray = null;

private boolean isMeasure = false;
private boolean isLockViewRange = false;
volatile boolean bSaveWav = true;//false;AV

CalibrationLoad calibLoad = new CalibrationLoad();  // data for calibration of spectrum



@Override
public void onCreate(Bundle savedInstanceState) {
    //  Debug.startMethodTracing("calc");
    final int maxMemory = (int) (Runtime.getRuntime().maxMemory() / 1024);
    Log.i(TAG, " max runtime mem = " + maxMemory + "k");

    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Resources res = getResources();
    analyzerParam = new AnalyzerParameters(res);

    // Initialized preferences by default values
    PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
    // Read preferences and set corresponding variables
    loadPreferenceForView();


    analyzerViews = new AnalyzerViews(this);

    // currentIndicator = analyzerViews attempting to get stuff TODO: as before

    // travel Views, and attach ClickListener to the views that contain android:tag="select"
    visit((ViewGroup) analyzerViews.graphView.getRootView(), new Visit() {
        @Override
        public void exec(View view) {
            view.setOnLongClickListener(AnalyzerActivity.this);
            view.setOnClickListener(AnalyzerActivity.this);
            ((TextView) view).setFreezesText(true);
        }
    }, "select");
    // P sure this is what actually turns the objects into buttons? Which is, like super cool?


    rangeViewDialogC = new RangeViewDialogC(this, analyzerViews.graphView);

    mDetector = new GestureDetectorCompat(this, new AnalyzerGestureListener());
}

/**
 * Run processClick() for views, transferring the state in the textView to our
 * internal state, then begin sampling and processing audio data
 */

@Override
protected void onResume() {
    Log.d(TAG, "onResume()");
    super.onResume();

    LoadPreferences();
    analyzerViews.graphView.setReady(this);  // TODO: move this earlier?
    //analyzerViews.enableSaveWavView(bSaveWav);

    // Used to prevent extra calling to restartSampling() (e.g. in LoadPreferences())
    bSamplingPreparation = true;

    // Start sampling
    restartSampling(analyzerParam);
}
@Override
public void onBackPressed(){
    toDogActivity.dogEdit.putString("dogID", " ");
    super.onBackPressed();
    finish();
}

@Override
protected void onPause() {
    Log.d(TAG, "onPause()");
```

```
        bSamplingPreparation = false;
        if (samplingThread != null) {
            samplingThread.finish();
        }
        getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        super.onPause();
    }

    @Override
    protected void onDestroy() {
        Log.d(TAG, "onDestroy()");
//      Debug.stopMethodTracing();
        super.onDestroy();
    }

    @Override
    public void onSaveInstanceState(Bundle savedInstanceState) {
        Log.d(TAG, "onSaveInstanceState()");
        savedInstanceState.putDouble("dtRMS",       dtRMS);
        savedInstanceState.putDouble("dtRMSFromFT", dtRMSFromFT);
        savedInstanceState.putDouble("maxAmpDB",    maxAmpDB);
        savedInstanceState.putDouble("maxAmpFreq",  maxAmpFreq);

        super.onSaveInstanceState(savedInstanceState);
    }

    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        Log.d(TAG, "onRestoreInstanceState()");
        // will be called after the onStart()
        super.onRestoreInstanceState(savedInstanceState);

        dtRMS       = savedInstanceState.getDouble("dtRMS");
        dtRMSFromFT = savedInstanceState.getDouble("dtRMSFromFT");
        maxAmpDB    = savedInstanceState.getDouble("maxAmpDB");
        maxAmpFreq  = savedInstanceState.getDouble("maxAmpFreq");
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.info, menu);
        return true;
    }
    static final int REQUEST_AUDIO_GET = 1;
    static final int REQUEST_CALIB_LOAD = 2;

    public void selectFile(int requestType) {
        // https://developer.android.com/guide/components/intents-common.html#Storage
        Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
        if (requestType == REQUEST_AUDIO_GET) {
            intent.setType("audio/*");
        } else {
            intent.setType("*/*");
        }
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        if (intent.resolveActivity(getPackageManager()) != null) {
            startActivityForResult(intent, requestType);
        } else {
            Log.e(TAG, "No file chooser found!.");

            // Potentially direct the user to the Market with a Dialog
            Toast.makeText(this, "Please install a File Manager.",
                    Toast.LENGTH_SHORT).show();
        }
    }

    public static String getMimeType(String url) {
        String type = null;
        String extension = MimeTypeMap.getFileExtensionFromUrl(url);
        if (extension != null) {
            type = MimeTypeMap.getSingleton().getMimeTypeFromExtension(extension);
        }
        return type;
    }

    void fillFftCalibration(AnalyzerParameters _analyzerParam, CalibrationLoad _calibLoad) {
        if (_calibLoad.freq == null || _calibLoad.freq.length == 0 || _analyzerParam == null) {
            return;
        }
        double[] freqTick = new double[_analyzerParam.fftLen/2 + 1];
        for (int i = 0; i < freqTick.length; i++) {
            freqTick[i] = (double)i / _analyzerParam.fftLen * _analyzerParam.sampleRate;
        }
        _analyzerParam.micGainDB = AnalyzerUtil.interpLinear(_calibLoad.freq, _calibLoad.gain,
freqTick);
        _analyzerParam.calibName = _calibLoad.name;
```

```
//        for (int i = 0; i < _analyzerParam.micGainDB.length; i++) {
//            Log.i(TAG, "calib: " + freqTick[i] + "Hz : " + _analyzerParam.micGainDB[i]);
//        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == REQUEST_CALIB_LOAD && resultCode == RESULT_OK) {
            final Uri uri = data.getData();
            calibLoad.loadFile(uri, this);
            Log.w(TAG, "mime:" + getContentResolver().getType(uri));
            fillFftCalibration(analyzerParam, calibLoad);
        } else if (requestCode == REQUEST_AUDIO_GET) {
            Log.w(TAG, "requestCode == REQUEST_AUDIO_GET");
        }
    }

    // for pass audioSourceIDs and audioSourceNames to MyPreferences
    public final static String MYPREFERENCES_MSG_SOURCE_ID =
"AnalyzerActivity.SOURCE_ID";
    public final static String MYPREFERENCES_MSG_SOURCE_NAME =
"AnalyzerActivity.SOURCE_NAME";

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Log.i(TAG, "onOptionsItemSelected(): " + item.toString());
        switch (item.getItemId()) {
            case R.id.menu_manual:
                analyzerViews.showInstructions();
                return true;
            case R.id.menu_settings:
                Intent settings = new Intent(getBaseContext(), MyPreferences.class);
                settings.putExtra(MYPREFERENCES_MSG_SOURCE_ID,
analyzerParam.audioSourceIDs);
                settings.putExtra(MYPREFERENCES_MSG_SOURCE_NAME,
analyzerParam.audioSourceNames);
                startActivity(settings);
                return true;
            case R.id.menu_test_recorder:
                Intent int_info_rec = new Intent(this, InfoRecActivity.class);
                startActivity(int_info_rec);
                return true;
            case R.id.menu_view_range:
                rangeViewDialogC.ShowRangeViewDialog();
                return true;
            case R.id.menu_calibration:
                selectFile(REQUEST_CALIB_LOAD);
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    // Popup menu click listener
    // Read chosen preference, save the preference, set the state.
    @Override
    public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
        // get the tag, which is the value we are going to use
        String selectedItemTag = v.getTag().toString();
        // if tag() is "0" then do not update anything (it is a title)
        if (selectedItemTag.equals("0")) {
            return ;
        }

        // get the text and set it as the button text
        String selectedItemText = ((TextView) v).getText().toString();

        int buttonId = Integer.parseInt((parent.getTag().toString()));
        Button buttonView = (Button) findViewById(buttonId);
        buttonView.setText(selectedItemText);

        boolean b_need_restart_audio;

        // Save the choosen preference
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
        SharedPreferences.Editor editor = sharedPref.edit();

        // so change of sample rate do not change view range
        if (! isLockViewRange) {
            viewRangeArray = analyzerViews.graphView.getViewPhysicalRange();
            // if range is align at boundary, extend the range.
            Log.i(TAG, "set sampling rate:a " + viewRangeArray[0] + " ==? " + viewRangeArray[6]);
            if (viewRangeArray[0] == viewRangeArray[6]) {
                viewRangeArray[0] = 0;
            }
        }

        // dismiss the pop up
        switch (buttonId) {
            case R.id.button_sample_rate:
                analyzerViews.popupMenuSampleRate.dismiss();
                if (! isLockViewRange) {
                  Log.i(TAG, "set sampling rate:b " + viewRangeArray[1] + " ==? " +
        viewRangeArray[6 + 1]);
                  if (viewRangeArray[1] == viewRangeArray[6 + 1]) {
                    viewRangeArray[1] = Integer.parseInt(selectedItemTag) / 2;
                  }
                  Log.i(TAG, "onItemClick(): viewRangeArray saved. " + viewRangeArray[0] + " ~ " +
        viewRangeArray[1]);
                }
                analyzerParam.sampleRate = Integer.parseInt(selectedItemTag);
                b_need_restart_audio = true;
                editor.putInt("button_sample_rate", analyzerParam.sampleRate);
                break;
            case R.id.button_fftlen:
                analyzerViews.popupMenuFFTLen.dismiss();
                analyzerParam.fftLen = Integer.parseInt(selectedItemTag);
                analyzerParam.hopLen = (int)(analyzerParam.fftLen*(1 -
        analyzerParam.overlapPercent/100) + 0.5);
                b_need_restart_audio = true;
                editor.putInt("button_fftlen", analyzerParam.fftLen);
                fillFftCalibration(analyzerParam, calibLoad);
                break;
            case R.id.button_average:
                analyzerViews.popupMenuAverage.dismiss();
                analyzerParam.nFFTAverage = Integer.parseInt(selectedItemTag);
                if (analyzerViews.graphView != null) {
                  analyzerViews.graphView.setTimeMultiplier(analyzerParam.nFFTAverage);
                }
                b_need_restart_audio = false;
                editor.putInt("button_average", analyzerParam.nFFTAverage);
                break;
            default:
                Log.w(TAG, "onItemClick(): no this button");
                b_need_restart_audio = false;
        }

        editor.commit();

        if (b_need_restart_audio) {
            restartSampling(analyzerParam);
        }
    }
    // Load preferences for Views
    // When this function is called, the SamplingLoop must not running in the meanwhile.
    private void loadPreferenceForView() {
        // load preferences for buttons
        // list-buttons
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
        analyzerParam.sampleRate   = sharedPref.getInt("button_sample_rate", 16000);
        analyzerParam.fftLen       = sharedPref.getInt("button_fftlen",      1024);
        analyzerParam.nFFTAverage  = sharedPref.getInt("button_average",        1);
        // toggle-buttons
        analyzerParam.isAWeighting = sharedPref.getBoolean("dbA", false);
        if (analyzerParam.isAWeighting) {
            ((SelectorText) findViewById(R.id.dbA)).nextValue();
        }
        boolean isSpam = sharedPref.getBoolean("spectrum_spectrogram_mode", true);
        if (!isSpam) {
            ((SelectorText) findViewById(R.id.spectrum_spectrogram_mode)).nextValue();
        }
        String axisMode = sharedPref.getString("freq_scaling_mode", "linear");
        SelectorText st = (SelectorText) findViewById(R.id.freq_scaling_mode);
        st.setValue(axisMode);

        Log.i(TAG, "loadPreferenceForView():"+
                "\n  sampleRate  = " + analyzerParam.sampleRate +
                "\n  fftLen      = " + analyzerParam.fftLen +
                "\n  nFFTAverage = " + analyzerParam.nFFTAverage);
        ((Button)
        findViewById(R.id.button_sample_rate)).setText(Integer.toString(analyzerParam.sampleRate));
        ((Button)
        findViewById(R.id.button_fftlen     )).setText(Integer.toString(analyzerParam.fftLen));
        ((Button)
        findViewById(R.id.button_average    )).setText(Integer.toString(analyzerParam.nFFTAverage));
    }

    private void LoadPreferences() {
        // Load preferences for recorder and views, beside loadPreferenceForView()
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);

        boolean keepScreenOn = sharedPref.getBoolean("keepScreenOn", true);
        if (keepScreenOn) {
            getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        } else {
            getWindow().clearFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
```

```
        }

        analyzerParam.audioSourceId = Integer.parseInt(sharedPref.getString("audioSource",
Integer.toString(analyzerParam.RECORDER_AGC_OFF)));
        analyzerParam.wndFuncName = sharedPref.getString("windowFunction", "Hanning");
        analyzerParam.spectrogramDuration =
Double.parseDouble(sharedPref.getString("spectrogramDuration",
                Double.toString(6.0)));
        analyzerParam.overlapPercent =
Double.parseDouble(sharedPref.getString("fft_overlap_percent", "0.0"));//AV changed to zero
        analyzerParam.hopLen = (int)(analyzerParam.fftLen*(1 -
analyzerParam.overlapPercent/100) + 0.5);

        // Settings of graph view
        // spectrum
        analyzerViews.graphView.setShowLines( sharedPref.getBoolean("showLines", false) );
        // set spectrum show range
        analyzerViews.graphView.setSpectrumDBLowerBound(
                Float.parseFloat(sharedPref.getString("spectrumRange",
Double.toString(AnalyzerGraphic.minDB)))
        );

        // spectrogram

analyzerViews.graphView.setSpectrogramModeShifting(sharedPref.getBoolean("spectrogramS
hifting", false));
        analyzerViews.graphView.setShowTimeAxis
(sharedPref.getBoolean("spectrogramTimeAxis", true));
        analyzerViews.graphView.setShowFreqAlongX
(sharedPref.getBoolean("spectrogramShowFreqAlongX", true));
        analyzerViews.graphView.setSmoothRender
(sharedPref.getBoolean("spectrogramSmoothRender", false));
        analyzerViews.graphView.setColorMap             (sharedPref.getString
("spectrogramColorMap", "Hot"));
        // set spectrogram show range
        analyzerViews.graphView.setSpectrogramDBLowerBound(Float.parseFloat(
                sharedPref.getString("spectrogramRange",
Double.toString(analyzerViews.graphView.spectrogramPlot.spectrogramBMP.dBLowerBound)))
);
        analyzerViews.graphView.setLogAxisMode(
                sharedPref.getBoolean("spectrogramLogPlotMethod", true));

        analyzerViews.bWarnOverrun = sharedPref.getBoolean("warnOverrun", false);
        analyzerViews.setFpsLimit(Double.parseDouble(
                sharedPref.getString("spectrogramFPS",
getString(R.string.spectrogram_fps_default))));

        // Apply settings by travel the views with android:tag="select".
        visit((ViewGroup) analyzerViews.graphView.getRootView(), new Visit() {
            @Override
            public void exec(View view) {
                processClick(view);
            }
        }, "select");

        // Get view range setting
        boolean isLock = sharedPref.getBoolean("view_range_lock", false);
        if (isLock) {
            Log.i(TAG, "LoadPreferences(): isLocked");
            // Set view range and stick to measure mode
            double[] rr = new double[AnalyzerGraphic.VIEW_RANGE_DATA_LENGTH];
            for (int i = 0; i < rr.length; i++) {
                rr[i] = AnalyzerUtil.getDouble(sharedPref, "view_range_rr_" + i, 0.0/0.0);
                if (Double.isNaN(rr[i])) {  // not properly initialized
                    Log.w(TAG, "LoadPreferences(): rr is not properly initialized");
                    rr = null;
                    break;
                }
            }
            if (rr != null) {
                viewRangeArray = rr;
            }
            stickToMeasureMode();
        } else {
            stickToMeasureModeCancel();
        }
    }

    void stickToMeasureMode() {
        isLockViewRange = true;
        switchMeasureAndScaleMode();  // Force set to Measure mode
    }

    void stickToMeasureModeCancel() {
        isLockViewRange = false;
        if (isMeasure) {
            switchMeasureAndScaleMode();  // Force set to ScaleMode
        }
```

```
}

private boolean isInGraphView(float x, float y) {
    analyzerViews.graphView.getLocationInWindow(windowLocation);
    return x >= windowLocation[0] && y >= windowLocation[1] &&
            x < windowLocation[0] + analyzerViews.graphView.getWidth() &&
            y < windowLocation[1] + analyzerViews.graphView.getHeight();
}

// Button processing
public void showPopupMenu(View view) {
    analyzerViews.showPopupMenu(view);
}

/**
 * Gesture Listener for graphView (and possibly other views)
 * How to attach these events to the graphView?
 * @author xyy
 */
private class AnalyzerGestureListener extends GestureDetector.SimpleOnGestureListener {
    @Override
    public boolean onDown(MotionEvent event) {  // enter here when down action happen
        flyingMoveHandler.removeCallbacks(flyingMoveRunnable);
        return true;
    }

    @Override
    public void onLongPress(MotionEvent event) {
        if (isInGraphView(event.getX(0), event.getY(0))) {
            if (!isMeasure) {  // go from "scale" mode to "cursor" mode
                switchMeasureAndScaleMode();
            }
        }
        measureEvent(event);  // force insert this event
    }

    @Override
    public boolean onDoubleTap(MotionEvent event) {
        if (!isMeasure) {
            scaleEvent(event);            // ends scale mode
            analyzerViews.graphView.resetViewScale();
        }
        return true;
    }
```

```
        @Override
        public boolean onFling(MotionEvent event1, MotionEvent event2,
                               float velocityX, float velocityY) {
            if (isMeasure) {
                // seems never reach here...
                return true;
            }
            // Log.d(TAG, "  AnalyzerGestureListener::onFling: " +
event1.toString()+event2.toString());
            // Fly the canvas in graphView when in scale mode
            shiftingVelocity = Math.sqrt(velocityX*velocityX + velocityY*velocityY);
            shiftingComponentX = velocityX / shiftingVelocity;
            shiftingComponentY = velocityY / shiftingVelocity;
            float DPRatio = getResources().getDisplayMetrics().density;
            flyAcceleration = 1200 * DPRatio;
            timeFlingStart = SystemClock.uptimeMillis();
            flyingMoveHandler.postDelayed(flyingMoveRunnable, 0);
            return true;
        }

        Handler flyingMoveHandler = new Handler();
        long timeFlingStart;                     // Prevent from running forever
        double flyDt = 1/20.;                     // delta t of refresh
        double shiftingVelocity;                   // fling velocity
        double shiftingComponentX;                // fling direction x
        double shiftingComponentY;                // fling direction y
        double flyAcceleration = 1200.;           // damping acceleration of fling, pixels/second^2

        Runnable flyingMoveRunnable = new Runnable() {
            @Override
            public void run() {
                double shiftingVelocityNew = shiftingVelocity - flyAcceleration*flyDt;
                if (shiftingVelocityNew < 0) shiftingVelocityNew = 0;
                // Number of pixels that should move in this time step
                double shiftingPixel = (shiftingVelocityNew + shiftingVelocity)/2 * flyDt;
                shiftingVelocity = shiftingVelocityNew;
                if (shiftingVelocity > 0f
                        && SystemClock.uptimeMillis() - timeFlingStart < 10000) {
                    // Log.i(TAG, "  fly pixels x=" + shiftingPixelX + " y=" + shiftingPixelY);
                    AnalyzerGraphic graphView = analyzerViews.graphView;
                    graphView.setXShift(graphView.getXShift() - shiftingComponentX*shiftingPixel /
graphView.getCanvasWidth() / graphView.getXZoom());
```

```
                    graphView.setYShift(graphView.getYShift() - shiftingComponentY*shiftingPixel /
graphView.getCanvasHeight() / graphView.getYZoom());
                    // Am I need to use runOnUiThread() ?
                    analyzerViews.invalidateGraphView();
                    flyingMoveHandler.postDelayed(flyingMoveRunnable, (int)(1000*flyDt));
                }
            }
        };
    }

    private void switchMeasureAndScaleMode() {
        if (isLockViewRange) {
            isMeasure = true;
            return;
        }
        isMeasure = !isMeasure;
        //SelectorText st = (SelectorText) findViewById(R.id.graph_view_mode);
        //st.performClick();
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        if (isInGraphView(event.getX(0), event.getY(0))) {
            this.mDetector.onTouchEvent(event);
            if (isMeasure) {
                measureEvent(event);
            } else {
                scaleEvent(event);
            }
            analyzerViews.invalidateGraphView();
            // Go to scaling mode when user release finger in measure mode.
            if (event.getActionMasked() == MotionEvent.ACTION_UP) {
                if (isMeasure) {
                    switchMeasureAndScaleMode();
                }
            }
        } else {
            // When finger is outside the plot, hide the cursor and go to scaling mode.
            if (isMeasure) {
                analyzerViews.graphView.hideCursor();
                switchMeasureAndScaleMode();
            }
        }
        return super.onTouchEvent(event);
    }

    /**
     *  Manage cursor for measurement
     */
    private void measureEvent(MotionEvent event) {
        switch (event.getPointerCount()) {
            case 1:
                analyzerViews.graphView.setCursor(event.getX(), event.getY());
                // TODO: if touch point is very close to boundary for a long time, move the view
                break;
            case 2:
                if (isInGraphView(event.getX(1), event.getY(1))) {
                    switchMeasureAndScaleMode();
                }
        }
    }

    /**
     *  Manage scroll and zoom
     */
    final private static double INIT = Double.MIN_VALUE;
    private boolean isPinching = false;
    private double xShift0 = INIT, yShift0 = INIT;
    private double x0, y0;
    private int[] windowLocation = new int[2];

    private void scaleEvent(MotionEvent event) {
        if (event.getAction() != MotionEvent.ACTION_MOVE) {
            xShift0 = INIT;
            yShift0 = INIT;
            isPinching = false;
            // Log.i(TAG, "scaleEvent(): Skip event " + event.getAction());
            return;
        }
        // Log.i(TAG, "scaleEvent(): switch " + event.getAction());
        AnalyzerGraphic graphView = analyzerViews.graphView;
        switch (event.getPointerCount()) {
            case 2 :
                if (isPinching)  {
                    graphView.setShiftScale(event.getX(0), event.getY(0), event.getX(1),
event.getY(1));
                } else {
                    graphView.setShiftScaleBegin(event.getX(0), event.getY(0), event.getX(1),
event.getY(1));
                }
                isPinching = true;
                break;
            case 1:
                float x = event.getX(0);
                float y = event.getY(0);
                graphView.getLocationInWindow(windowLocation);
                // Log.i(TAG, "scaleEvent(): xy=" + x + " " + y + "  wc = " + wc[0] + " " + wc[1]);
                if (isPinching || xShift0 == INIT) {
                    xShift0 = graphView.getXShift();
                    x0 = x;
                    yShift0 = graphView.getYShift();
                    y0 = y;
                } else {
                    // when close to the axis, scroll that axis only
                    if (x0 < windowLocation[0] + 50) {
                        graphView.setYShift(yShift0 + (y0 - y) / graphView.getCanvasHeight() /
graphView.getYZoom());
                    } else if (y0 < windowLocation[1] + 50) {
                        graphView.setXShift(xShift0 + (x0 - x) / graphView.getCanvasWidth() /
graphView.getXZoom());
                    } else {
                        graphView.setXShift(xShift0 + (x0 - x) / graphView.getCanvasWidth() /
graphView.getXZoom());
                        graphView.setYShift(yShift0 + (y0 - y) / graphView.getCanvasHeight() /
graphView.getYZoom());
                    }
                }
                isPinching = false;
                break;
            default:
                Log.i(TAG, "Invalid touch count");
                break;
        }
    }

    @Override
    public boolean onLongClick(View view) {
        vibrate(300);
        Log.i(TAG, "long click: " + view.toString());
        return true;
    }
    // Responds to layout with android:tag="select"
    // Called from SelectorText.super.performClick()
    @Override
    public void onClick(View v) {
        if (processClick(v)) {
            restartSampling(analyzerParam);
        }
        analyzerViews.invalidateGraphView();
    }

    private final int MY_PERMISSIONS_REQUEST_RECORD_AUDIO = 1;  // just a number
    private final int MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE = 2;
    Thread graphInit;
    private boolean bSamplingPreparation = false;

    private void restartSampling(final AnalyzerParameters _analyzerParam) {
        // Stop previous sampler if any.
        if (samplingThread != null) {
            samplingThread.finish();
            try {
                samplingThread.join();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            samplingThread = null;
        }

        if (viewRangeArray != null) {
            analyzerViews.graphView.setupAxes(analyzerParam);
            double[] rangeDefault = analyzerViews.graphView.getViewPhysicalRange();
            Log.i(TAG, "restartSampling(): setViewRange: " + viewRangeArray[0] + " ~ " +
viewRangeArray[1]);
            analyzerViews.graphView.setViewRange(viewRangeArray, rangeDefault);
            if (! isLockViewRange) viewRangeArray = null;  // do not conserve
        }

        // Set the view for incoming data
        graphInit = new Thread(new Runnable() {
            public void run() {
                analyzerViews.setupView(_analyzerParam);
            }
        });
        graphInit.start();
        // Check and request permissions
        if (! checkAndRequestPermissions())
            return;

        if (! bSamplingPreparation)
            return;

        // Start sampling
        samplingThread = new SamplingLoop(this, _analyzerParam);
        samplingThread.start();
        handler.postDelayed(runnableCode,1000);

        //put reward dog logic here

        //trial 2
       /* int counter = 0;
        if(AnalyzerViews.test.contains("a")){
            counter=counter+1;}
            if(counter>=1){
                rewardDog();
                counter = 0;
            }*/



        //trial 1
        /*
        int counter = 0;
        while(true) {
            if (AnalyzerViews.textCurChar[counter] == 'a') {
                rewardDog();
            }
            else{
            counter=counter+1;}
        }*/
    }

    // For call requestPermissions() after each showPermissionExplanation()
    private int count_permission_explanation = 0;

    // For preventing infinity loop: onResume() -> requestPermissions() ->
onRequestPermissionsResult() -> onResume()
    private int count_permission_request = 0;
    // Test and try to gain permissions.
    // Return true if it is OK to proceed.
    // Ref.
    //   https://developer.android.com/training/permissions/requesting.html
    //   https://developer.android.com/guide/topics/permissions/requesting.html
    private boolean checkAndRequestPermissions() {
        if (ContextCompat.checkSelfPermission(AnalyzerActivity.this,
Manifest.permission.RECORD_AUDIO) != PackageManager.PERMISSION_GRANTED) {
            Log.w(TAG, "Permission RECORD_AUDIO denied. Trying  to request...");
            if (ActivityCompat.shouldShowRequestPermissionRationale(AnalyzerActivity.this,
Manifest.permission.RECORD_AUDIO) &&
                    count_permission_explanation < 1) {
                Log.w(TAG, "  Show explanation here....");

analyzerViews.showPermissionExplanation(R.string.permission_explanation_recorder);
                count_permission_explanation++;
            } else {
                Log.w(TAG, "  Requesting...");
                if (count_permission_request < 3) {
                    ActivityCompat.requestPermissions(AnalyzerActivity.this,
                            new String[]{Manifest.permission.RECORD_AUDIO},
                            MY_PERMISSIONS_REQUEST_RECORD_AUDIO);
                    count_permission_explanation = 0;
                    count_permission_request++;
                } else {
                    this.runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            Context context = getApplicationContext();
                            String text = "Permission denied.";
                            Toast toast = Toast.makeText(context, text, Toast.LENGTH_LONG);
                            toast.show();
                        }
                    });
                }
            }
            return false;
        }
        if (bSaveWav && ContextCompat.checkSelfPermission(AnalyzerActivity.this,
Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
PackageManager.PERMISSION_GRANTED) {
            Log.w(TAG, "Permission WRITE_EXTERNAL_STORAGE denied. Trying  to request...");
            //((SelectorText) findViewById(R.id.button_recording)).nextValue();
            //bSaveWav = false;
            //analyzerViews.enableSaveWavView(bSaveWav);
            ActivityCompat.requestPermissions(AnalyzerActivity.this,
                    new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                    MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE);
        }

        return true;
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           @NonNull String permissions[], @NonNull int[] grantResults) {
        switch (requestCode) {
            case MY_PERMISSIONS_REQUEST_RECORD_AUDIO: {
                if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                    Log.w(TAG, "RECORD_AUDIO Permission granted by user.");
                } else {
                    Log.w(TAG, "RECORD_AUDIO Permission denied by user.");
                }
                break;
            }
            case MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE: {
                if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                    Log.w(TAG, "WRITE_EXTERNAL_STORAGE Permission granted by user.");
                    if (! bSaveWav) {
                        Log.w(TAG, "... bSaveWav == true");
                        runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                ((SelectorText) findViewById(R.id.button_recording)).nextValue();
                                bSaveWav = true;
                                analyzerViews.enableSaveWavView(bSaveWav);
                            }
                        });
                    } else {
                        Log.w(TAG, "... bSaveWav == false");
                    }
                } else {
                    Log.w(TAG, "WRITE_EXTERNAL_STORAGE Permission denied by user.");
                }
                break;
            }
        }
        // Then onResume() will be called.
    }

    /**
     * Process a click on one of our selectors.
     * @param v   The view that was clicked
     * @return    true if we need to update the graph
     */

    public boolean processClick(View v) {
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(this);
        SharedPreferences.Editor editor = sharedPref.edit();
        String value;
        if (v instanceof SelectorText) {
            value = ((SelectorText) v).getValue();
        } else {
            value = ((TextView) v).getText().toString();
        }
        switch (v.getId()) {
            case R.id.button_recording:
                bSaveWav = true;//value.equals("Rec");//AV
                //  SelectorText st = (SelectorText) findViewById(R.id.run);
                //  if (bSaveWav && ! st.getText().toString().equals("stop")) {
                //    st.nextValue();
                //    if (samplingThread != null) {
                //      samplingThread.setPause(true);
                //    }
                //  }
                analyzerViews.enableSaveWavView(bSaveWav);
                return true;
            case R.id.run:
                boolean pause = value.equals("stop");
                if (samplingThread != null && samplingThread.getPause() != pause) {
                    samplingThread.setPause(pause);
                }
                analyzerViews.graphView.spectrogramPlot.setPause(pause);
                return false;
            //case R.id.graph_view_mode:
            //  isMeasure = !value.equals("scale");
            //  return false;
            case R.id.freq_scaling_mode: {
                Log.d(TAG, "processClick(): freq_scaling_mode = " + value);
                analyzerViews.graphView.setAxisModeLinear(value);
                editor.putString("freq_scaling_mode", value);
                editor.commit();
                return false;
            }
            case R.id.dbA:
                analyzerParam.isAWeighting = !value.equals("dB");
                if (samplingThread != null) {
                    samplingThread.setAWeighting(analyzerParam.isAWeighting);
                }
                editor.putBoolean("dbA", analyzerParam.isAWeighting);
                editor.commit();
                return false;
            case R.id.spectrum_spectrogram_mode:
                if (value.equals("spum")) {
                    analyzerViews.graphView.switch2Spectrum();
                } else {
                    analyzerViews.graphView.switch2Spectrogram();
                }
                editor.putBoolean("spectrum_spectrogram_mode", value.equals("spum"));
                editor.commit();
                return false;
            default:
                return true;
        }
    }

    private void vibrate(int ms) {
        //((Vibrator) getSystemService(Context.VIBRATOR_SERVICE)).vibrate(ms);
    }

    /**
     * Visit all subviews of this view group and run command
     * @param group   The parent view group
     * @param cmd     The command to run for each view
     * @param select  The tag value that must match. Null implies all views
     */

    private void visit(ViewGroup group, Visit cmd, String select) {
        exec(group, cmd, select);
        for (int i = 0; i < group.getChildCount(); i++) {
            View c = group.getChildAt(i);
            if (c instanceof ViewGroup) {
                visit((ViewGroup) c, cmd, select);
            } else {
                exec(c, cmd, select);
            }
        }
    }

    private void exec(View v, Visit cmd, String select) {
        if (select == null || select.equals(v.getTag())) {
            cmd.exec(v);
        }
    }

    /**
     * Interface for view hierarchy visitor
     */
    interface Visit {
        void exec(View view);
    }

    /**
     * The graph view size has been determined - update the labels accordingly.
     */
    @Override
    public void ready() {
        // put code here for the moment that graph size just changed
        Log.v(TAG, "ready()");
        analyzerViews.invalidateGraphView();
    }


    public void rewardDog() {
        if (ChipRobotFinder.getInstance().getChipRobotConnectedList().size() > 0) {
            ChipRobot robot = (ChipRobot)
ChipRobotFinder.getInstance().getChipRobotConnectedList().get(0);

            robot.chipPlayBodycon((byte) (1));


        }}

    private Handler handler = new Handler();
    boolean stop = false;
    //repetitive task that should check to see if dog is connected. if not display alert
    private Runnable runnableCode = new Runnable() {
        @Override
        public void run() {

            if (ChipRobotFinder.getInstance().getChipRobotConnectedList().size() == 0)
            {
                opentoDogActivity();
                finish();
                stop=true;
                //handler.removeCallbacks(runnableCode);
            }
            else{}

            if(!stop){handler.postDelayed(this, 1000);}



        }
    };

    public void opentoDogActivity(){
        Intent intent = new Intent(this,toDogActivity.class);
        startActivity(intent);
    }


    //might not use this
    public void openDialog() {
        AlertDialog alertDialog = new AlertDialog();
        alertDialog.setShowsDialog(true);

    }


}
```

AnalyzerGraphic

```
/* Copyright 2011 Google Inc.
 *
 *Licensed under the Apache License, Version 2.0 (the "License");
 *you may not use this file except in compliance with the License.
 *You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 *Unless required by applicable law or agreed to in writing, software
 *distributed under the License is distributed on an "AS IS" BASIS,
 *WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *See the License for the specific language governing permissions and
 *limitations under the License.
 *
 * @author Stephen Uhler
 *
 * 2014 Eddy Xiao <bewantbe@gmail.com>
 * GUI extensively modified.
 * Add some naive auto refresh rate control logic.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.content.Context;
import android.content.res.Resources;
import android.graphics.Canvas;
import android.os.Parcel;
import android.os.Parcelable;
import android.util.AttributeSet;
import android.util.Log;
import android.view.View;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

/**
 * Custom view to draw the FFT graph
 */

public class AnalyzerGraphic extends View {
    private final String TAG = "AnalyzerGraphic:";
    private Context context;
    private double xZoom, yZoom;     // horizontal and vertical scaling
    private double xShift, yShift;   // horizontal and vertical translation, in unit 1 unit
    static final double minDB = -144f;    // hard lower bound for dB
    static final double maxDB = 12f;      // hard upper bound for dB

    private int canvasWidth, canvasHeight;   // size of my canvas
    private int[] myLocation = {0, 0}; // window location on screen
    private volatile static boolean isBusy = false;
    private double freq_lower_bound_for_log = 0f;
    private double[] savedDBSpectrum = new double[0];
    static final int VIEW_RANGE_DATA_LENGTH = 6;

    SpectrumPlot    spectrumPlot;
    SpectrogramPlot spectrogramPlot;

    private PlotMode showMode = PlotMode.SPECTRUM;

    enum PlotMode {  // java's enum type is inconvenient
        SPECTRUM(0), SPECTROGRAM(1);

        private final int value;
        PlotMode(int value) { this.value = value; }
        public int getValue() { return value; }
    }

    public AnalyzerGraphic(Context context, AttributeSet attrs, int defStyle) {
        // https://developer.android.com/training/custom-views/create-view.html
        super(context, attrs, defStyle);
        setup(context);
    }

    public AnalyzerGraphic(Context context, AttributeSet attrs) {
        super(context, attrs);
        setup(context);
    }

    public AnalyzerGraphic(Context context) {
        super(context);
        setup(context);
    }

    private void setup(Context _context) {
        context = _context;
        Log.i(TAG, "in setup()");

        xZoom  = 1f;
        xShift = 0f;
        yZoom  = 1f;
        yShift = 0f;
        canvasWidth = canvasHeight = 0;
        // Demo of full initialization
        spectrumPlot    = new SpectrumPlot(context);
        spectrogramPlot = new SpectrogramPlot(context);

        spectrumPlot   .setCanvas(canvasWidth, canvasHeight, null);
        spectrogramPlot.setCanvas(canvasWidth, canvasHeight, null);

        spectrumPlot   .setZooms(xZoom, xShift, yZoom, yShift);
        spectrogramPlot.setZooms(xZoom, xShift, yZoom, yShift);

        Resources res = _context.getResources();
        spectrumPlot.axisY.vLowerBound =
    Float.parseFloat(res.getString(R.string.max_DB_range));
    }

    AnalyzerParameters analyzerParamCache;

    void setupAxes(AnalyzerParameters analyzerParam) {
        int sampleRate       = analyzerParam.sampleRate;
        int fftLen           = analyzerParam.fftLen;
        int nAve             = analyzerParam.nFFTAverage;
        double timeDurationE = analyzerParam.spectrogramDuration;

        freq_lower_bound_for_log = (double)sampleRate/fftLen;
        double freq_lower_bound_local = 0;
        if (spectrumPlot.axisX.mapType == ScreenPhysicalMapping.Type.LOG) {
          freq_lower_bound_local = freq_lower_bound_for_log;
        }
        // Spectrum
        double axisBounds[] = new double[]{freq_lower_bound_local, 0.0, sampleRate/2.0,
    spectrumPlot.axisY.vUpperBound};
        Log.i(TAG, "setupAxes(): W=" + canvasWidth + "  H=" + canvasHeight + "  dB=" +
    spectrumPlot.axisY.vUpperBound);
        spectrumPlot.setCanvas(canvasWidth, canvasHeight, axisBounds);

        // Spectrogram
        freq_lower_bound_local = 0;
        if (spectrogramPlot.axisFreq.mapType == ScreenPhysicalMapping.Type.LOG) {
          freq_lower_bound_local = freq_lower_bound_for_log;
        }
        if (spectrogramPlot.showFreqAlongX) {
          axisBounds = new double[]{freq_lower_bound_local, 0.0, sampleRate/2.0,
    timeDurationE * nAve};
```

```
        } else {
            axisBounds = new double[]{0.0, sampleRate/2.0, timeDurationE * nAve,
freq_lower_bound_local};
        }
        spectrogramPlot.setCanvas(canvasWidth, canvasHeight, axisBounds);

        analyzerParamCache = analyzerParam;
    }

    // Call this when settings changed.
    void setupPlot(AnalyzerParameters analyzerParam) {
        setupAxes(analyzerParam);
        spectrogramPlot.setupSpectrogram(analyzerParam);
    }

    void setAxisModeLinear(String mode) {
        boolean b = mode.equals("linear");  // see also layout/*.xml
        ScreenPhysicalMapping.Type mapType;
        GridLabel.Type gridType;
        if (b) {
            mapType = ScreenPhysicalMapping.Type.LINEAR;
            gridType = GridLabel.Type.FREQ;
        } else {
            mapType = ScreenPhysicalMapping.Type.LOG;
            gridType = mode.equals("note") ? GridLabel.Type.FREQ_NOTE :
GridLabel.Type.FREQ_LOG;
        }
        spectrumPlot   .setFreqAxisMode(mapType, freq_lower_bound_for_log, gridType);
        spectrogramPlot.setFreqAxisMode(mapType, freq_lower_bound_for_log, gridType);
        if (showMode == PlotMode.SPECTRUM) {
            xZoom  = spectrumPlot.axisX.getZoom();
            xShift = spectrumPlot.axisX.getShift();
        } else if (showMode == PlotMode.SPECTROGRAM) {
            if (spectrogramPlot.showFreqAlongX) {
                xZoom  = spectrogramPlot.axisFreq.getZoom();
                xShift = spectrogramPlot.axisFreq.getShift();
            } else {
                yZoom  = spectrogramPlot.axisFreq.getZoom();
                yShift = spectrogramPlot.axisFreq.getShift();
            }
        }
    }

    public void setShowFreqAlongX(boolean b) {
        spectrogramPlot.setShowFreqAlongX(b);

        if (showMode == PlotMode.SPECTRUM) return;

        if (spectrogramPlot.showFreqAlongX) {
            xZoom  = spectrogramPlot.axisFreq.getZoom();
            xShift = spectrogramPlot.axisFreq.getShift();
            yZoom  = spectrogramPlot.axisTime.getZoom();
            yShift = spectrogramPlot.axisTime.getShift();
        } else {
            xZoom  = spectrogramPlot.axisTime.getZoom();
            xShift = spectrogramPlot.axisTime.getShift();
            yZoom  = spectrogramPlot.axisFreq.getZoom();
            yShift = spectrogramPlot.axisFreq.getShift();
        }
    }

    // Note: Assume setupPlot() was called once.
    public void switch2Spectrum() {
        if (showMode == PlotMode.SPECTRUM) {
            return;
        }
        Log.v(TAG, "switch2Spectrum()");
        // execute when switch from Spectrogram mode to Spectrum mode
        showMode = PlotMode.SPECTRUM;
        xZoom  = spectrogramPlot.axisFreq.getZoom();
        xShift = spectrogramPlot.axisFreq.getShift();
        if (! spectrogramPlot.showFreqAlongX) {
            xShift = 1 - 1/xZoom - xShift;
        }
        spectrumPlot.axisX.setZoomShift(xZoom, xShift);

        yZoom  = spectrumPlot.axisY.getZoom();
        yShift = spectrumPlot.axisY.getShift();
    }

    // Note: Assume setupPlot() was called once.
    public void switch2Spectrogram() {
        if (showMode == PlotMode.SPECTRUM && canvasHeight > 0) { // canvasHeight==0
means the program is just start
            Log.v(TAG, "switch2Spectrogram()");
            if (spectrogramPlot.showFreqAlongX) {
                // no need to change x scaling
                yZoom  = spectrogramPlot.axisTime.getZoom();
        yShift = spectrogramPlot.axisTime.getShift();
        spectrogramPlot.axisFreq.setZoomShift(xZoom, xShift);
      } else {
        //noinspection SuspiciousNameCombination
        yZoom = xZoom;
        yShift = 1 - 1/xZoom - xShift;         // axisFreq is reverted
        xZoom  = spectrogramPlot.axisTime.getZoom();
        xShift = spectrogramPlot.axisTime.getShift();
        spectrogramPlot.axisFreq.setZoomShift(yZoom, yShift);
      }
    }
    spectrogramPlot.spectrogramBMP.updateAxis(spectrogramPlot.axisFreq);
    spectrogramPlot.prepare();
    showMode = PlotMode.SPECTROGRAM;
  }

  double[] setViewRange(double[] _ranges, double[] rangesDefault) {
    // See AnalyzerActivity::getViewPhysicalRange() for ranges[]
    if (_ranges.length < VIEW_RANGE_DATA_LENGTH) {
      Log.i(TAG, "setViewRange(): invalid input.");
      return null;
    }

    // do not modify input parameter
    double[] ranges = new double[VIEW_RANGE_DATA_LENGTH];
    System.arraycopy(_ranges, 0, ranges, 0, VIEW_RANGE_DATA_LENGTH);

    if (rangesDefault != null) {
      // Sanity check
      if (rangesDefault.length != 2 * VIEW_RANGE_DATA_LENGTH) {
        Log.i(TAG, "setViewRange(): invalid input.");
        return null;
      }
      for (int i = 0; i < 6; i += 2) {
        if (ranges[i  ] > ranges[i+1]) {                    // order reversed
          double t = ranges[i]; ranges[i] = ranges[i+1]; ranges[i+1] = t;
        }
        if (ranges[i  ] < rangesDefault[i+6]) ranges[i  ] = rangesDefault[i+6];  // lower  than
lower bound
        if (ranges[i+1] < rangesDefault[i+6]) ranges[i+1] = rangesDefault[i+7];  // all lower  than
lower bound?
        if (ranges[i  ] > rangesDefault[i+7]) ranges[i  ] = rangesDefault[i+6];  // all higher than
upper bound?
```

```
                if (ranges[i+1] > rangesDefault[i+7]) ranges[i+1] = rangesDefault[i+7];  // higher than
upper bound
                if (ranges[i    ] == ranges[i + 1] || Double.isNaN(ranges[i]) || Double.isNaN(ranges[i +
1])) {  // invalid input value
                    ranges[i    ] = rangesDefault[i];
                    ranges[i + 1] = rangesDefault[i + 1];
                }
            }
        }

        // Set range
        if (showMode == PlotMode.SPECTRUM) {
            spectrumPlot.axisX.setViewBounds(ranges[0], ranges[1]);
            spectrumPlot.axisY.setViewBounds(ranges[3], ranges[2]);  // reversed
        } else if (showMode == PlotMode.SPECTROGRAM) {
            if (spectrogramPlot.getSpectrogramMode() == SpectrogramPlot.TimeAxisMode.SHIFT) {
                spectrogramPlot.axisTime.setViewBounds(ranges[5], ranges[4]);
            } else {
                spectrogramPlot.axisTime.setViewBounds(ranges[4], ranges[5]);
            }
            if (spectrogramPlot.showFreqAlongX) {
                spectrogramPlot.axisFreq.setViewBounds(ranges[0], ranges[1]);
            } else {
                spectrogramPlot.axisFreq.setViewBounds(ranges[1], ranges[0]);
            }
            spectrogramPlot.spectrogramBMP.updateAxis(spectrogramPlot.axisFreq);
        }

        // Set zoom shift for view
        if (showMode == PlotMode.SPECTRUM) {
            xZoom  = spectrumPlot.axisX.getZoom();
            xShift = spectrumPlot.axisX.getShift();
            yZoom  = spectrumPlot.axisY.getZoom();
            yShift = spectrumPlot.axisY.getShift();
        } else if (showMode == PlotMode.SPECTROGRAM) {
            if (spectrogramPlot.showFreqAlongX) {
                xZoom  = spectrogramPlot.axisFreq.getZoom();
                xShift = spectrogramPlot.axisFreq.getShift();
                yZoom  = spectrogramPlot.axisTime.getZoom();
                yShift = spectrogramPlot.axisTime.getShift();
            } else {
                yZoom  = spectrogramPlot.axisFreq.getZoom();
                yShift = spectrogramPlot.axisFreq.getShift();
                xZoom  = spectrogramPlot.axisTime.getZoom();
                xShift = spectrogramPlot.axisTime.getShift();
            }
        }

        return ranges;
    }

    private void updateAxisZoomShift() {
        if (showMode == PlotMode.SPECTRUM) {
            spectrumPlot   .setZooms(xZoom, xShift, yZoom, yShift);
        } else {
            spectrogramPlot.setZooms(xZoom, xShift, yZoom, yShift);
        }
    }

    public void setSmoothRender(boolean b) {
        spectrogramPlot.setSmoothRender(b);
    }

    public PlotMode getShowMode() {
        return showMode;
    }

    public void setTimeMultiplier(int nAve) {
        spectrogramPlot.setTimeMultiplier(nAve);
    }

    public void setShowTimeAxis(boolean bSTA) {
        spectrogramPlot.setShowTimeAxis(bSTA);
    }

    public void setSpectrogramModeShifting(boolean b) {
        spectrogramPlot.setSpectrogramModeShifting(b);
    }

    void setLogAxisMode(boolean b) {
        SpectrogramBMP.LogAxisPlotMode mode =
SpectrogramBMP.LogAxisPlotMode.REPLOT;
        if (!b) {
            mode = SpectrogramBMP.LogAxisPlotMode.SEGMENT;
        }
        spectrogramPlot.spectrogramBMP.setLogAxisMode(mode);
    }
```

```
//    void setSpectrogramBMPWidth(int w) {
//        spectrogramPlot.spectrogramBMP.setBmpWidth(w);
//    }

    void setColorMap(String colorMapName) {
        spectrogramPlot.setColorMap(colorMapName);
    }

    static boolean isBusy() {
        return isBusy;
    }

    static void setIsBusy(boolean b) { isBusy = b; }

    private final FPSCounter fpsCounter = new FPSCounter("AnalyzerGraphic");
//  long t_old;

    @Override
    protected void onDraw(Canvas c) {
        fpsCounter.inc();
        isBusy = true;
        if (showMode == PlotMode.SPECTRUM) {
            spectrumPlot.addCalibCurve(analyzerParamCache.micGainDB, null,
analyzerParamCache.calibName);
            spectrumPlot.drawSpectrumPlot(c, savedDBSpectrum);
        } else {
            spectrogramPlot.drawSpectrogramPlot(c);
        }
        isBusy = false;
    }

    // All FFT data will enter this view through this interface
    // Will be called in another thread (SamplingLoop)
    public void saveSpectrum(double[] db) {
        synchronized (savedDBSpectrum) {  // TODO: need lock on savedDBSpectrum, but how?
            if (savedDBSpectrum == null || savedDBSpectrum.length != db.length) {
                savedDBSpectrum = new double[db.length];
            }
            System.arraycopy(db, 0, savedDBSpectrum, 0, db.length);
        }
        // TODO: Run on another thread? Lock on data ? Or use CompletionService?
        if (showMode == PlotMode.SPECTROGRAM) {
            spectrogramPlot.saveRowSpectrumAsColor(savedDBSpectrum);
        }
    }

    void setSpectrumDBLowerBound(double b) {
        spectrumPlot.axisY.vUpperBound = b;
    }

    void setSpectrogramDBLowerBound(double b) {
        spectrogramPlot.setSpectrogramDBLowerBound(b);
    }

    public void setShowLines(boolean b) {
        spectrumPlot.showLines = b;
    }

    private boolean intersects(float x, float y) {
        getLocationOnScreen(myLocation);
        return x >= myLocation[0] && y >= myLocation[1] &&
                x < myLocation[0] + getWidth() && y < myLocation[1] + getHeight();
    }

    // return true if the coordinate (x,y) is inside graphView
    public boolean setCursor(float x, float y) {
        if (intersects(x, y)) {
            x = x - myLocation[0];
            y = y - myLocation[1];
            // Convert to coordinate in axis
            if (showMode == PlotMode.SPECTRUM) {
                spectrumPlot.setCursor(x, y);
            } else {
                spectrogramPlot.setCursor(x, y);
            }
            return true;
        } else {
            return false;
        }
    }

    public double getCursorFreq() {
        if (showMode == PlotMode.SPECTRUM) {
            return spectrumPlot.getCursorFreq();
        } else {
            return spectrogramPlot.getCursorFreq();
        }
    }
```

```
public double getCursorDB() {
   if (showMode == PlotMode.SPECTRUM) {
      return spectrumPlot.getCursorDB();
   } else {
      return 0;
   }
}

public void hideCursor() {
   spectrumPlot.hideCursor();
   spectrogramPlot.hideCursor();
}

double[] getViewPhysicalRange() {
   double[] r = new double[12];
   if (getShowMode() == AnalyzerGraphic.PlotMode.SPECTRUM) {
      // fL, fU, dBL dBU, time L, time U
      r[0] = spectrumPlot.axisX.vMinInView();
      r[1] = spectrumPlot.axisX.vMaxInView();
      r[2] = spectrumPlot.axisY.vMaxInView(); // reversed
      r[3] = spectrumPlot.axisY.vMinInView();
      r[4] = 0;
      r[5] = 0;

      // Limits of fL, fU, dBL dBU, time L, time U
      r[6] = spectrumPlot.axisX.vLowerBound;
      r[7] = spectrumPlot.axisX.vUpperBound;
      r[8] = AnalyzerGraphic.minDB;
      r[9] = AnalyzerGraphic.maxDB;
      r[10]= 0;
      r[11]= 0;
   } else {
      r[0] = spectrogramPlot.axisFreq.vMinInView();
      r[1] = spectrogramPlot.axisFreq.vMaxInView();
      if (r[0] > r[1]) { double t=r[0]; r[0]=r[1]; r[1]=t; }
      r[2] = spectrogramPlot.spectrogramBMP.dBLowerBound;
      r[3] = spectrogramPlot.spectrogramBMP.dBUpperBound;
      r[4] = spectrogramPlot.axisTime.vMinInView();
      r[5] = spectrogramPlot.axisTime.vMaxInView();

      r[6] = spectrogramPlot.axisFreq.vLowerBound;
      r[7] = spectrogramPlot.axisFreq.vUpperBound;
      if (r[6] > r[7]) { double t=r[6]; r[6]=r[7]; r[7]=t; }
```

```
            r[8] = AnalyzerGraphic.minDB;
            r[9] = AnalyzerGraphic.maxDB;
            r[10]= spectrogramPlot.axisTime.vLowerBound;
            r[11]= spectrogramPlot.axisTime.vUpperBound;
        }
        for (int i=6; i<r.length; i+=2) {
            if (r[i] > r[i+1]) {
                double t = r[i]; r[i] = r[i+1]; r[i+1] = t;
            }
        }
        return r;
    }

    public double getXZoom() {
        return xZoom;
    }

    public double getYZoom() {
        return yZoom;
    }

    public double getXShift() {
        return xShift;
    }

    public double getYShift() {
        return yShift;
    }

    public double getCanvasWidth() {
        if (showMode == PlotMode.SPECTRUM) {
            return canvasWidth;
        } else {
            return canvasWidth - spectrogramPlot.labelBeginX;
        }
    }

    public double getCanvasHeight() {
        if (showMode == PlotMode.SPECTRUM) {
            return canvasHeight;
        } else {
            return spectrogramPlot.labelBeginY;
        }
    }
    private double clamp(double x, double min, double max) {
        if (x > max) {
            return max;
        } else if (x < min) {
            return min;
        } else {
            return x;
        }
    }

    private double clampXShift(double offset) {
        return clamp(offset, 0f, 1 - 1 / xZoom);
    }

    private double clampYShift(double offset) {
        if (showMode == PlotMode.SPECTRUM) {
            // limit view to minDB ~ maxDB, assume linear in dB scale
            return clamp(offset, (maxDB - spectrumPlot.axisY.vLowerBound) /
spectrumPlot.axisY.diffVBounds(),
                    (minDB - spectrumPlot.axisY.vLowerBound) / spectrumPlot.axisY.diffVBounds() - 1
/ yZoom);
        } else {
            // strict restrict, y can be frequency or time.
            return clamp(offset, 0f, 1 - 1 / yZoom);
        }
    }

    public void setXShift(double offset) {
        xShift = clampXShift(offset);
        updateAxisZoomShift();
    }

    public void setYShift(double offset) {
        yShift = clampYShift(offset);
        updateAxisZoomShift();
    }

    public void resetViewScale() {
        xShift = 0;
        xZoom = 1;
        yShift = 0;
        yZoom = 1;
        updateAxisZoomShift();
    }

    private double xMidOld = 100;
    private double xDiffOld = 100;
    private double xZoomOld = 1;
    private double xShiftOld = 0;
    private double yMidOld = 100;
    private double yDiffOld = 100;
    private double yZoomOld = 1;
    private double yShiftOld = 0;

    // record the coordinate frame state when starting scaling
    public void setShiftScaleBegin(double x1, double y1, double x2, double y2) {
        xMidOld = (x1+x2)/2f;
        xDiffOld = Math.abs(x1-x2);
        xZoomOld  = xZoom;
        xShiftOld = xShift;
        yMidOld = (y1+y2)/2f;
        yDiffOld = Math.abs(y1-y2);
        yZoomOld  = yZoom;
        yShiftOld = yShift;
    }

    // Do the scaling according to the motion event getX() and getY() (getPointerCount()==2)
    public void setShiftScale(double x1, double y1, double x2, double y2) {
        double limitXZoom;
        double limitYZoom;
        if (showMode == PlotMode.SPECTRUM) {
            limitXZoom =  spectrumPlot.axisX.diffVBounds()/200f;  // limit to 200 Hz a screen
            limitYZoom = -spectrumPlot.axisY.diffVBounds()/6f;    // limit to 6 dB a screen
        } else {
            int nTimePoints = spectrogramPlot.nTimePoints;
            if (spectrogramPlot.showFreqAlongX) {
                limitXZoom = spectrogramPlot.axisFreq.diffVBounds()/200f;
                limitYZoom = nTimePoints>10 ? nTimePoints / 10 : 1;
            } else {
                limitXZoom = nTimePoints>10 ? nTimePoints / 10 : 1;
                limitYZoom = spectrogramPlot.axisFreq.diffVBounds()/200f;
            }
        }
        limitXZoom = Math.abs(limitXZoom);
        limitYZoom = Math.abs(limitYZoom);
//        Log.i(TAG, "setShiftScale: limit: xZ="+limitXZoom+"  yZ="+limitYZoom);
```

```
        if (canvasWidth*0.13f < xDiffOld) {  // if fingers are not very close in x direction, do scale in
x direction
            xZoom  = clamp(xZoomOld * Math.abs(x1-x2)/xDiffOld, 1f, limitXZoom);
        }
        xShift = clampXShift(xShiftOld + (xMidOld/xZoomOld - (x1+x2)/2f/xZoom) / canvasWidth);
        if (canvasHeight*0.13f < yDiffOld) {  // if fingers are not very close in y direction, do scale in
y direction
            yZoom  = clamp(yZoomOld * Math.abs(y1-y2)/yDiffOld, 1f, limitYZoom);
        }
        yShift = clampYShift(yShiftOld + (yMidOld/yZoomOld - (y1+y2)/2f/yZoom) / canvasHeight);
//        Log.i(TAG, "setShiftScale: xZ="+xZoom+"  xS="+xShift+"  yZ="+yZoom+"  yS="+yShift);
        updateAxisZoomShift();
    }

    private Ready readyCallback = null;      // callback to caller when rendering is complete

    public void setReady(Ready ready) {
        this.readyCallback = ready;
    }

    interface Ready {
        void ready();
    }

    @Override
    protected void onSizeChanged (int w, int h, int oldw, int oldh) {
        Log.i(TAG, "onSizeChanged(): canvas (" + oldw + "," + oldh + ") -> (" + w + "," + h + ")");
        isBusy = true;
        this.canvasWidth = w;
        this.canvasHeight = h;
        spectrumPlot   .setCanvas(w, h, null);
        spectrogramPlot.setCanvas(w, h, null);
        if (h > 0 && readyCallback != null) {
            readyCallback.ready();
        }
        isBusy = false;
    }

  /*
   * Save freqAxisAlongX, cursors, zooms, and current spectrogram/spectrum.
   * All other properties will be set in onCreate() and onResume().
   * Will be called after onPause() or between onCreat() and on onResume()
   * Ref. https://developer.android.com/guide/topics/ui/settings.html#CustomSaveState
   */
```

```
    @Override
    protected Parcelable onSaveInstanceState() {
        Log.i(TAG, "onSaveInstanceState(): xShift = " + xShift + "  xZoom = " + xZoom + "  yShift =
" + yShift + "  yZoom = " + yZoom);
        Parcelable parentState = super.onSaveInstanceState();
        SavedState state = new SavedState(parentState);

        state.freqAxisAlongX = spectrogramPlot.showFreqAlongX ? 1 : 0;

        state.cFreqSpum  = spectrumPlot.cursorFreq;
        state.cFreqSpam  = spectrogramPlot.cursorFreq;
        state.cDb  = spectrumPlot.cursorDB;
        state.xZ  = xZoom;
        state.xS  = xShift;
        state.yZ  = yZoom;
        state.yS  = yShift;
        state.SpumXZ = spectrumPlot.axisX.getZoom();
        state.SpumXS = spectrumPlot.axisX.getShift();
        state.SpumYZ = spectrumPlot.axisY.getZoom();
        state.SpumYS = spectrumPlot.axisY.getShift();
        state.SpamFZ = spectrogramPlot.axisFreq.getZoom();
        state.SpamFS = spectrogramPlot.axisFreq.getShift();
        state.SpamTZ = spectrogramPlot.axisTime.getZoom();
        state.SpamTS = spectrogramPlot.axisTime.getShift();

        state.tmpS     = savedDBSpectrum;

        state.nFreq    = spectrogramPlot.nFreqPoints;
        state.nTime    = spectrogramPlot.nTimePoints;
        state.iTimePinter = spectrogramPlot.spectrogramBMP.spectrumStore.iTimePointer;

        final short[] tmpSC    = spectrogramPlot.spectrogramBMP.spectrumStore.dbShortArray;

        // Get byte[] representation of short[]
        // Note no ByteBuffer view of ShortBuffer, see
        //   http://bugs.java.com/bugdatabase/view_bug.do?bug_id=4489356
        byte[] input = new byte[tmpSC.length * 2];
        for (int i = 0; i < tmpSC.length; i++) {
            input[2*i  ] = (byte)(tmpSC[i] & 0xff);
            input[2*i+1] = (byte)(tmpSC[i] >> 8);
        }

        // Save spectrumStore.dbShortArray to a file.
        File tmpSCPath = new File(context.getCacheDir(), "spectrogram_short.raw");
        try {
            OutputStream fout = new FileOutputStream(tmpSCPath);
            fout.write(input);
            fout.close();
        } catch (IOException e) {
            Log.w("SavedState:", "writeToParcel(): Fail to save state to file.");
        }

        return state;
    }

    // Will be called after setup(), during AnalyzerActivity.onCreate()?
    @Override
    public void onRestoreInstanceState(Parcelable state) {
        if (state instanceof SavedState) {
            SavedState s = (SavedState) state;
            super.onRestoreInstanceState(s.getSuperState());

            spectrogramPlot.showFreqAlongX = s.freqAxisAlongX == 1;

            spectrumPlot.cursorFreq    = s.cFreqSpum;
            spectrogramPlot.cursorFreq = s.cFreqSpam;
            spectrumPlot.cursorDB      = s.cDb;
            xZoom  = s.xZ;
            xShift = s.xS;
            yZoom  = s.yZ;
            yShift = s.yS;
            spectrumPlot.axisX.setZoomShift(s.SpumXZ, s.SpumXS);
            spectrumPlot.axisY.setZoomShift(s.SpumYZ, s.SpumYS);
            spectrogramPlot.axisFreq.setZoomShift(s.SpamFZ, s.SpamFS);
            spectrogramPlot.axisTime.setZoomShift(s.SpamTZ, s.SpamTS);

            savedDBSpectrum = s.tmpS;

            spectrogramPlot.nFreqPoints = s.nFreq;
            spectrogramPlot.nTimePoints = s.nTime;

            final SpectrogramBMP sBMP = spectrogramPlot.spectrogramBMP;
            final SpectrogramBMP.SpectrumCompressStore sBMPS = sBMP.spectrumStore;
            sBMPS.nFreq = s.nFreq;  // prevent reinitialize of LogFreqSpectrogramBMP
            sBMPS.nTime = s.nTime;
            sBMPS.iTimePointer = s.iTimePinter;
```

```
            // Read temporary saved spectrogram
            byte[] input = new byte[(s.nFreq+1) * s.nTime * 2]; // length of
spectrumStore.dbShortArray
            int bytesRead = -1;
            File tmpSCPath = new File(context.getCacheDir(), "spectrogram_short.raw");
            try {
                InputStream fin = new FileInputStream(tmpSCPath);
                bytesRead = fin.read(input);
                fin.close();
            } catch (IOException e) {
                Log.w("SavedState:", "writeToParcel(): Fail to save state to file.");
            }

            if (bytesRead != input.length) {  // fail to get saved spectrogram, have a new start
                sBMPS.nFreq = 0;
                sBMPS.nTime = 0;
                sBMPS.iTimePointer = 0;
            } else {  // we have data!
                short[] tmpSC = new short[input.length/2];
                for (int i = 0; i < tmpSC.length; i++) {
                    tmpSC[i] = (short)(input[2*i] + (input[2*i+1] << 8));
                }
                sBMPS.dbShortArray = tmpSC;
                sBMP.rebuildLinearBMP();
            }

            Log.i(TAG, "onRestoreInstanceState(): xShift = " + xShift + "  xZoom = " + xZoom + "
yShift = " + yShift + "  yZoom = " + yZoom);
        } else {
            Log.i(TAG, "onRestoreInstanceState(): not SavedState?!");
            super.onRestoreInstanceState(state);
        }
    }

    private static class SavedState extends BaseSavedState {
        int freqAxisAlongX;
        double cFreqSpum;
        double cFreqSpam;
        double cDb;
        double xZ;
        double xS;
        double yZ;
        double yS;
        double SpumXZ;
        double SpumXS;
        double SpumYZ;
        double SpumYS;
        double SpamFZ;
        double SpamFS;
        double SpamTZ;
        double SpamTS;

        double[] tmpS;

        int nFreq;
        int nTime;
        int iTimePinter;

        SavedState(Parcelable state) {
            super(state);
        }

        private SavedState(Parcel in) {
            super(in);
            freqAxisAlongX = in.readInt();
            cFreqSpum  = in.readDouble();
            cFreqSpam  = in.readDouble();
            cDb  = in.readDouble();
            xZ   = in.readDouble();
            xS   = in.readDouble();
            yZ   = in.readDouble();
            yS   = in.readDouble();
            SpumXZ = in.readDouble();
            SpumXS = in.readDouble();
            SpumYZ = in.readDouble();
            SpumYS = in.readDouble();
            SpamFZ = in.readDouble();
            SpamFS = in.readDouble();
            SpamTZ = in.readDouble();
            SpamTS = in.readDouble();

            tmpS = in.createDoubleArray();

            nFreq       = in.readInt();
            nTime       = in.readInt();
            iTimePinter = in.readInt();
        }
```

```
        @Override
        public void writeToParcel(Parcel out, int flags) {
            super.writeToParcel(out, flags);
            out.writeInt(freqAxisAlongX);
            out.writeDouble(cFreqSpum);
            out.writeDouble(cFreqSpam);
            out.writeDouble(cDb);
            out.writeDouble(xZ);
            out.writeDouble(xS);
            out.writeDouble(yZ);
            out.writeDouble(yS);
            out.writeDouble(SpumXZ);
            out.writeDouble(SpumXS);
            out.writeDouble(SpumYZ);
            out.writeDouble(SpumYS);
            out.writeDouble(SpamFZ);
            out.writeDouble(SpamFS);
            out.writeDouble(SpamTZ);
            out.writeDouble(SpamTS);

            out.writeDoubleArray(tmpS);

            out.writeInt(nFreq);
            out.writeInt(nTime);
            out.writeInt(iTimePinter);
        }

        // Standard creator object using an instance of this class
        public static final Parcelable.Creator<SavedState> CREATOR = new
Parcelable.Creator<SavedState>() {
            @Override
            public SavedState createFromParcel(Parcel in) {
                return new SavedState(in);
            }

            @Override
            public SavedState[] newArray(int size) {
                return new SavedState[size];
            }
        };
    }
}
```

AnalyzerParameters

```
/* Copyright 2017 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.content.res.Resources;
import android.media.MediaRecorder;
import android.util.Log;

/**
 * Basic properties of Analyzer.
 */

class AnalyzerParameters {
    final int RECORDER_AGC_OFF = MediaRecorder.AudioSource.VOICE_RECOGNITION;
    int audioSourceId = RECORDER_AGC_OFF;
    int sampleRate = 16000;
    int fftLen = 2048;
    int hopLen = 1024;
    double overlapPercent = 0;  // = (1 - hopLen/fftLen) * 100%//AV changed from 50 to 0
    String wndFuncName;
    int nFFTAverage = 2;
    boolean isAWeighting = false;
    final int BYTE_OF_SAMPLE = 2;
    final double SAMPLE_VALUE_MAX = 32767.0;   // Maximum signal value
    double spectrogramDuration = 4.0;

    double[] micGainDB = null;  // should have fftLen/2+1 elements, i.e. include DC.
    String calibName = null;

    AnalyzerParameters(Resources res) {
        getAudioSourceNameFromIdPrepare(res);
    }

    String[] audioSourceNames;
    int[] audioSourceIDs;
    private void getAudioSourceNameFromIdPrepare(Resources res) {
        audioSourceNames   = res.getStringArray(R.array.audio_source);
        String[] sasid = res.getStringArray(R.array.audio_source_id);
        audioSourceIDs = new int[audioSourceNames.length];
        for (int i = 0; i < audioSourceNames.length; i++) {
            audioSourceIDs[i] = Integer.parseInt(sasid[i]);
        }
    }

    // Get audio source name from its ID
    // Tell me if there is better way to do it.
    String getAudioSourceNameFromId(int id) {
        for (int i = 0; i < audioSourceNames.length; i++) {
            if (audioSourceIDs[i] == id) {
                return audioSourceNames[i];
            }
        }
        Log.i("AnalyzerParameters", "getAudioSourceName(): non-standard entry.");
        return ((Integer)(id)).toString();
    }

    String getAudioSourceName() {
        return getAudioSourceNameFromId(audioSourceId);
    }
}
```

AnalyzerUtil

```
 */

package github.bewantbe.audio_analyzer_for_android;

import android.content.Context;
import android.content.SharedPreferences;
import android.media.AudioFormat;
import android.media.AudioRecord;
import android.media.MediaRecorder;
import android.os.Build;
import android.support.annotation.NonNull;
import android.util.Log;

import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

import static java.lang.Math.abs;
import static java.lang.Math.round;
import static java.util.Arrays.sort;

/**
 * Utility functions for audio analyzer.
 */

class AnalyzerUtil {
    private static String TAG = "AnalyzerUtil";
    private static final String[] LP = {"C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#", "B"};

    static double freq2pitch(double f) {
        return 69 + 12 * Math.log(f/440.0)/Math.log(2);  // MIDI pitch
    }

    static double pitch2freq(double p) {
        return Math.pow(2, (p - 69)/12) * 440.0;
    }

    static void pitch2Note(StringBuilder a, double p, int prec_frac, boolean tightMode) {
        int pi = (int) Math.round(p);
        int po = (int) Math.floor(pi/12.0);
        int pm = pi-po*12;
        a.append(LP[pm]);
        SBNumFormat.fillInInt(a, po-1);
        if (LP[pm].length() == 1 && !tightMode) {
            a.append(' ');
        }
        double cent = Math.round(100 * (p - pi) * Math.pow(10, prec_frac)) * Math.pow(10, -
prec_frac);
        if (!tightMode) {
            SBNumFormat.fillInNumFixedWidthSignedFirst(a, cent, 2, prec_frac);
        } else {
            if (cent != 0) {
                a.append(cent < 0 ? '-' : '+');
                SBNumFormat.fillInNumFixedWidthPositive(a, Math.abs(cent), 2, prec_frac, '\0');
            }
        }
    }

    // Convert frequency to pitch
    // Fill with sFill until length is 6. If sFill=="", do not fill
    static void freq2Cent(StringBuilder a, double f, String sFill) {
        if (f<=0 || Double.isNaN(f) || Double.isInfinite(f)) {
            a.append("      ");
            return;
        }
        int len0 = a.length();
        // A4 = 440Hz
        double p = freq2pitch(f);
        pitch2Note(a, p, 0, false);
        while (a.length()-len0 < 6 && sFill!=null && sFill.length()>0) {
            a.append(sFill);
        }
    }

    // used to detect if the data is unchanged
    private double[] cmpDB;
    void sameTest(double[] data) {
        // test
        if (cmpDB == null || cmpDB.length != data.length) {
            Log.i(TAG, "sameTest(): new");
            cmpDB = new double[data.length];
        } else {
            boolean same = true;
            for (int i=0; i<data.length; i++) {
                if (!Double.isNaN(cmpDB[i]) && !Double.isInfinite(cmpDB[i]) && cmpDB[i] != data[i]) {
                    same = false;
                    break;
                }
            }
            if (same) {
                Log.i(TAG, "sameTest(): same data row!!");
            }
            System.arraycopy(data, 0, cmpDB, 0, data.length);
        }
    }

    static boolean isAlmostInteger(double x) {
        // return x % 1 == 0;
        double i = round(x);
        if (i == 0) {
            return abs(x) < 1.2e-7;  // 2^-23 = 1.1921e-07
        } else {
            return abs(x - i) / i < 1.2e-7;
        }
    }

    /**
     * Return a array of verified audio sampling rates.
     * @param requested: the sampling rates to be verified
     */
    static String[] validateAudioRates(String[] requested) {
        ArrayList<String> validated = new ArrayList<>();
        for (String s : requested) {
            int rate;
            String[] sv = s.split("::");
            if (sv.length == 1) {
                rate = Integer.parseInt(sv[0]);
            } else {
                rate = Integer.parseInt(sv[1]);
            }
            if (rate != 0) {
                int recBufferSize = AudioRecord.getMinBufferSize(rate,
                        AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT);
                if (recBufferSize != AudioRecord.ERROR_BAD_VALUE) {
                    // Extra exam to high sampling rate, by opening it.
                    if (rate > 48000) {
                        AudioRecord record;
                        try {
                            record = new AudioRecord(MediaRecorder.AudioSource.DEFAULT, rate,
                                    AudioFormat.CHANNEL_IN_MONO,
AudioFormat.ENCODING_PCM_16BIT, recBufferSize);
```

```
                        } catch (IllegalArgumentException e) {
                            continue;
                        }
                        if (record.getState() == AudioRecord.STATE_INITIALIZED) {
                            validated.add(s);
                        }
                    } else {
                        validated.add(s);
                    }
                }
            } else {
                validated.add(s);
            }
        }
        return validated.toArray(new String[0]);
    }

    static double parseDouble(String st) {
        try {
            return Double.parseDouble(st);
        } catch (NumberFormatException e) {
            return 0.0/0.0;  // nan
        }
    }

    // Thanks http://stackoverflow.com/questions/16319237/cant-put-double-sharedpreferences
    static SharedPreferences.Editor putDouble(final SharedPreferences.Editor edit, final String
key, final double value) {
        return edit.putLong(key, Double.doubleToRawLongBits(value));
    }
    static double getDouble(final SharedPreferences prefs, final String key, final double
defaultValue) {
        return Double.longBitsToDouble(prefs.getLong(key,
Double.doubleToLongBits(defaultValue)));
    }

    final int[]    stdSourceId;  // how to make it final?
    final int[]    stdSourceApi;
    final String[] stdSourceName;
    final String[] stdAudioSourcePermission;
    AnalyzerUtil(Context context) {
        stdSourceId   = context.getResources().getIntArray(R.array.std_audio_source_id);
        stdSourceApi  = context.getResources().getIntArray(R.array.std_audio_source_api_level);
    stdSourceName          =
context.getResources().getStringArray(R.array.std_audio_source_name);
    stdAudioSourcePermission =
context.getResources().getStringArray(R.array.std_audio_source_permission);
  }

  // filterLevel = 0: no filter
  //          & 1: leave only standard sources
  //          & 2: leave only permitted sources (&1)
  //          & 4: leave only sources coincide the API level (&1)
  int[] GetAllAudioSource(int filterLevel) {
    // Use reflection to get all possible audio source (in compilation environment)
    ArrayList<Integer> iList = new ArrayList<>();
    Class<MediaRecorder.AudioSource> clazz = MediaRecorder.AudioSource.class;
    Field[] arr = clazz.getFields();
    for (Field f : arr) {
      if (f.getType().equals(int.class)) {
        try {
          int id = (int)f.get(null);
          iList.add(id);
          Log.i("Sources id:", "" + id);
        } catch (IllegalAccessException e) {}
      }
    }
    Collections.sort(iList);

    // Filter unnecessary audio source
    Iterator<Integer> iterator;
    ArrayList<Integer> iListRet = iList;
    if (filterLevel > 0) {
      iListRet = new ArrayList<>();
      iterator = iList.iterator();
      for (int i = 0; i < iList.size(); i++) {
        int id = iterator.next();
        int k = arrayContainInt(stdSourceId, id); // get the index to standard source if possible
        if (k < 0) continue;
        if ((filterLevel & 2) > 0 && !stdAudioSourcePermission[k].equals("")) continue;
        if ((filterLevel & 4) > 0 && stdSourceApi[k] > Build.VERSION.SDK_INT) continue;
        iListRet.add(id);
      }
    }

    // Return an int array
    int[] ids = new int[iListRet.size()];
    iterator = iListRet.iterator();
    for (int i = 0; i < ids.length; i++) {
        ids[i] = iterator.next();
    }
    return ids;
}

String getAudioSourceName(int id) {
    int k = arrayContainInt(stdSourceId, id);
    if (k >= 0) {
        return stdSourceName[k];
    } else {
        return "(unknown id=" + id +")";
    }
}

// Java s**ks
static int arrayContainInt(int[] arr, int v) {
    if (arr == null) return -1;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == v) return i;
    }
    return -1;
}

static int arrayContainString(String[] arr, String v) {
    if (arr == null) return -1;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i].equals(v)) return i;
    }
    return -1;
}

static boolean isSorted(double[] a) {
    if (a == null || a.length <= 1) {
        return true;
    }
    double d = a[0];
    for (int i = 1; i < a.length; i++) {
        if (d > a[i]) {
            return false;
        }
        d = a[i];
    }
```

```
        return true;
    }

    static double[] interpLinear(double[] xFixed, double[] yFixed, double[] xInterp) {
        if (xFixed == null || yFixed == null || xInterp == null) {
            return null;
        }
        if (xFixed.length == 0 || yFixed.length == 0 || xInterp.length == 0) {
            return new double[0];
        }
        if (xFixed.length != yFixed.length) {
            Log.e(TAG, "Input data length mismatch");
        }

//        if (!isSorted(xFixed)) {
//            sort(xFixed);
//            yFixed = yFixed(x_id);
//        }
        if (!isSorted(xInterp)) {
            sort(xInterp);
        }
        double[] yInterp = new double[xInterp.length];

        int xiId = 0;
        while (xiId < xInterp.length && xInterp[xiId] < xFixed[0]) {
            yInterp[xiId] = yFixed[0];
            xiId++;
        }

        for (int xfId = 1; xfId < xFixed.length; xfId++) {
            while (xiId < xInterp.length && xInterp[xiId] < xFixed[xfId]) {
                // interpolation using (xFixed(x_id - 1), yFixed(xfId - 1)) and (xFixed(x_id),
yFixed(xfId))
                yInterp[xiId] = (yFixed[xfId] - yFixed[xfId - 1]) / (xFixed[xfId] - xFixed[xfId - 1]) *
(xInterp[xiId] - xFixed[xfId - 1]) + yFixed[xfId - 1];
                xiId++;
            }
        }

        while (xiId < xInterp.length) {
            yInterp[xiId] = yFixed[yFixed.length - 1];
            xiId++;
        }
        return yInterp;
    }

    // Binary search for nearest element.
    // bLower == true : return lower  index if no exact match.
    // bLower == false: return higher index if no exact match.
    // x must be sorted (ascending).
    static int binarySearchElem(double[] x, double v, boolean bLower) {
        if (x == null || x.length == 0) return -1;
        int l = 0;
        int h = x.length - 1;
        if (v == x[l]) {
            return l;
        }
        if (v < x[l]) {
            if (bLower) {
                return l;  // or? -1;
            } else {
                return l;
            }
        }
        if (v == x[h]) {
            return h;
        }
        if (v > x[h]) {
            if (bLower) {
                return h;
            } else {
                return h;  // or? -1;
            }
        }
        // x[0] < v < x[len-1]
        int m;
        while (l+1 < h) {
            m = (l + h) / 2;
            if (v == x[m]) {
                return m;
            }
            if (v < x[m]) {
                h = m;
            } else {
                l = m;
            }
        }
        // Now we must have l+1 == h, and no exact match was found.
        if (bLower) {
            return l;
        } else {
            return h;
        }
    }

    static class PeakHoldAndFall {
        double[] v_peak = new double[0];
        double[] t_peak = new double[0];
        double t_hold = 2.0;
        double drop_speed = 20.0;

        PeakHoldAndFall() {}

        PeakHoldAndFall(double hold_time, double _drop_speed) {
            t_hold = hold_time;
            drop_speed = _drop_speed;
        }

        void addCurrentValue(@NonNull double[] v_curr, double dt) {
            if (v_curr.length != v_peak.length) {
                v_peak = new double[v_curr.length];
                System.arraycopy(v_curr, 0, v_peak, 0, v_curr.length);
                t_peak = new double[v_curr.length];
                // t_peak initialized to 0.0 by default (Java).
            } else {
                for (int k = 0; k < v_peak.length; k++) {
                    double t_now = t_peak[k] + dt;
                    double v_drop = 0;
                    // v_peak falling in quadratic speed.
                    if (t_peak[k] > t_hold) {
                        v_drop = (t_peak[k] - t_hold + t_now - t_hold) / 2 * dt;
                    } else if (t_now > t_hold) {
                        v_drop = (t_now - t_hold) / 2 * dt;
                    }
                    t_peak[k] = t_now;
                    v_peak[k] -= drop_speed * v_drop;
                    // New peak arrived.
                    if (v_peak[k] < v_curr[k]) {
                        v_peak[k] = v_curr[k];
                        t_peak[k] = 0;
                    }
                }
            }
        }
    }
}
```

AnalyzerViews

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.res.Resources;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Typeface;
import android.os.Build;
import android.os.Handler;
import android.os.SystemClock;
import android.support.annotation.NonNull;
import android.text.Html;
import android.text.method.ScrollingMovementMethod;
import android.text.method.LinkMovementMethod;
import android.util.Log;
import android.util.TypedValue;
import android.view.Display;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.PopupWindow;
import android.widget.TextView;
import android.widget.Toast;

import android.os.Handler;

import com.wowwee.bluetoothrobotcontrollib.chip.ChipRobot;
import com.wowwee.bluetoothrobotcontrollib.chip.ChipRobotFinder;


/**
 * Operate the views in the UI here.
 * Should run on UI thread in general.
 */

class AnalyzerViews extends Activity {
    final String TAG = "AnalyzerViews";
    private final AnalyzerActivity activity;
    final AnalyzerGraphic graphView;
    String prefDirect = toDogActivity.prefDirect;
    SharedPreferences dogPref = toDogActivity.dogPref;
    SharedPreferences.Editor dogEdit = toDogActivity.dogEdit;


    private float DPRatio;
    private float listItemTextSize = 20;        // see R.dimen.button_text_fontsize
    private float listItemTitleTextSize = 12;   // see R.dimen.button_text_small_fontsize
    private double fpsLimit = 8;

    public static StringBuilder textCur = new StringBuilder("");  // for textCurChar
    private StringBuilder textRMS  = new StringBuilder("");
    private StringBuilder textPeak = new StringBuilder("");
    private StringBuilder textRec = new StringBuilder("");

    TextView testingChar;
```

```
private char[] textRMSChar;   // for text in R.id.textview_RMS
public static char[] textCurChar;   // for text in R.id.textview_cur
private char[] textPeakChar;  // for text in R.id.textview_peak
private char[] textRecChar;   // for text in R.id.textview_rec
public static String isATextCur;

public static String test;

PopupWindow popupMenuSampleRate;
PopupWindow popupMenuFFTLen;
PopupWindow popupMenuAverage;

boolean bWarnOverrun = true;

AnalyzerViews(AnalyzerActivity _activity) {
  activity = _activity;
  graphView = (AnalyzerGraphic) activity.findViewById(R.id.plot);

  Resources res = activity.getResources();
  listItemTextSize      = res.getDimension(R.dimen.button_text_fontsize);
  listItemTitleTextSize = res.getDimension(R.dimen.button_text_small_fontsize);
  DPRatio = res.getDisplayMetrics().density;

  textRMSChar  = new char[res.getString(R.string.textview_RMS_text).length()];
  textCurChar  = new char[res.getString(R.string.textview_cur_text).length()];
  textRecChar  = new char[res.getString(R.string.textview_rec_text).length()];
  textPeakChar = new char[res.getString(R.string.textview_peak_text).length()];
  isATextCur = res.getString(R.string.testingString);


  /// initialize pop up window items list
  // http://www.codeofaninja.com/2013/04/show-listview-as-drop-down-android.html
  popupMenuSampleRate = popupMenuCreate( AnalyzerUtil.validateAudioRates(
      res.getStringArray(R.array.sample_rates)), R.id.button_sample_rate);
  popupMenuFFTLen = popupMenuCreate(
      res.getStringArray(R.array.fft_len), R.id.button_fftlen);
  popupMenuAverage = popupMenuCreate(
      res.getStringArray(R.array.fft_ave_num), R.id.button_average);

  setTextViewFontSize();
}

// Set text font size of textview_cur and textview_peak
// according to space left
```

```
  //@SuppressWarnings("deprecation")
  private void setTextViewFontSize() {
    TextView tv = (TextView) activity.findViewById(R.id.textview_cur);
    // At this point tv.getWidth(), tv.getLineCount() will return 0

    Display display = activity.getWindowManager().getDefaultDisplay();
    // pixels left
    float px = display.getWidth() -
activity.getResources().getDimension(R.dimen.textview_RMS_layout_width) - 5;

    float fs = tv.getTextSize();  // size in pixel

    // shrink font size if it can not fit in one line.
    final String text = activity.getString(R.string.textview_peak_text);
    // note: mTestPaint.measureText(text) do not scale like sp.
    Paint mTestPaint = new Paint();
    mTestPaint.setTextSize(fs);
    mTestPaint.setTypeface(Typeface.MONOSPACE);
    while (mTestPaint.measureText(text) > px && fs > 5) {
      fs -= 0.5;
      mTestPaint.setTextSize(fs);
    }

    ((TextView)
activity.findViewById(R.id.textview_cur)).setTextSize(TypedValue.COMPLEX_UNIT_PX, fs);
    ((TextView)
activity.findViewById(R.id.textview_peak)).setTextSize(TypedValue.COMPLEX_UNIT_PX, fs);
  }

  // Prepare the spectrum and spectrogram plot (from scratch or full reset)
  // Should be called before samplingThread starts.
  void setupView(AnalyzerParameters analyzerParam) {
    graphView.setupPlot(analyzerParam);
  }

  // Will be called by SamplingLoop (in another thread)
  void update(final double[] spectrumDBcopy) {
    graphView.saveSpectrum(spectrumDBcopy);
    activity.runOnUiThread(new Runnable() {
      @Override
      public void run() {
        // data will get out of synchronize here
        invalidateGraphView();
      }
    });
}

private double wavSecOld = 0;      // used to reduce frame rate
void updateRec(double wavSec) {
    if (wavSecOld > wavSec) {
        wavSecOld = wavSec;
    }
    if (wavSec - wavSecOld < 0.1) {
        return;
    }
    wavSecOld = wavSec;
    activity.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            // data will get out of synchronize here
            invalidateGraphView(AnalyzerViews.VIEW_MASK_RecTimeLable);
        }
    });
}

void notifyWAVSaved(final String path) {
    String text = "WAV saved to " + path;
    notifyToast(text);
}

void notifyToast(final String st) {
    activity.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Context context = activity.getApplicationContext();
            Toast toast = Toast.makeText(context, st, Toast.LENGTH_SHORT);
            toast.show();
        }
    });
}

private long lastTimeNotifyOverrun = 0;
void notifyOverrun() {
    if (!bWarnOverrun) {
        return;
    }
    long t = SystemClock.uptimeMillis();
    if (t - lastTimeNotifyOverrun > 6000) {
            lastTimeNotifyOverrun = t;
            activity.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Context context = activity.getApplicationContext();
                    String text = "Recorder buffer overrun!\nYour cell phone is too slow.\nTry lower
sampling rate or higher average number.";
                    Toast toast = Toast.makeText(context, text, Toast.LENGTH_LONG);
                    toast.show();
                }
            });
        }
    }

    void showInstructions() {
        TextView tv = new TextView(activity);
        tv.setMovementMethod(LinkMovementMethod.getInstance());
        tv.setTextSize(TypedValue.COMPLEX_UNIT_SP, 15);
        tv.setText(fromHtml(activity.getString(R.string.instructions_text)));
        PackageInfo pInfo = null;
        String version = "\n" + activity.getString(R.string.app_name) + "  Version: ";
        try {
            pInfo = activity.getPackageManager().getPackageInfo(activity.getPackageName(), 0);
            version += pInfo.versionName;
        } catch (PackageManager.NameNotFoundException e) {
            version += "(Unknown)";
        }
        tv.append(version);
        new AlertDialog.Builder(activity)
                .setTitle(R.string.instructions_title)
                .setView(tv)
                .setNegativeButton(R.string.dismiss, null)
                .create().show();
    }

    void showPermissionExplanation(int resId) {
        TextView tv = new TextView(activity);
        tv.setMovementMethod(new ScrollingMovementMethod());
        tv.setText(fromHtml(activity.getString(resId)));
        new AlertDialog.Builder(activity)
                .setTitle(R.string.permission_explanation_title)
                .setView(tv)
                .setNegativeButton(R.string.dismiss, null)
                .create().show();
    }

    // Thanks http://stackoverflow.com/questions/37904739/html-fromhtml-deprecated-in-android-
n
    @SuppressWarnings("deprecation")
    public static android.text.Spanned fromHtml(String source) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.N) {
            return Html.fromHtml(source, Html.FROM_HTML_MODE_LEGACY); // or
Html.FROM_HTML_MODE_COMPACT
        } else {
            return Html.fromHtml(source);
        }
    }

    void enableSaveWavView(boolean bSaveWav) {
        //if (bSaveWav) {
            ((TextView) activity.findViewById(R.id.textview_rec)).setHeight((int)(19*DPRatio));
        //} else {
        //    ((TextView) activity.findViewById(R.id.textview_rec)).setHeight((int)(0*DPRatio));
        //}
    }

    @SuppressWarnings("deprecation")
    void showPopupMenu(View view) {
        // popup menu position
        // In API 19, we can use showAsDropDown(View anchor, int xoff, int yoff, int gravity)
        // The problem in showAsDropDown (View anchor, int xoff, int yoff) is
        // it may show the window in wrong direction (so that we can't see it)
        int[] wl = new int[2];
        view.getLocationInWindow(wl);
        int x_left = wl[0];
        int y_bottom = activity.getWindowManager().getDefaultDisplay().getHeight() - wl[1];
        int gravity = android.view.Gravity.START | android.view.Gravity.BOTTOM;

        switch (view.getId()) {
            case R.id.button_sample_rate:
                popupMenuSampleRate.showAtLocation(view, gravity, x_left, y_bottom);
//                popupMenuSampleRate.showAsDropDown(view, 0, 0);
                break;
            case R.id.button_fftlen:
                popupMenuFFTLen.showAtLocation(view, gravity, x_left, y_bottom);
//                popupMenuFFTLen.showAsDropDown(view, 0, 0);
                break;
            case R.id.button_average:
                popupMenuAverage.showAtLocation(view, gravity, x_left, y_bottom);
//                popupMenuAverage.showAsDropDown(view, 0, 0);
                break;
        }
    }

    // Maybe put this PopupWindow into a class
    private PopupWindow popupMenuCreate(String[] popUpContents, int resId) {

        // initialize a pop up window type
        PopupWindow popupWindow = new PopupWindow(activity);

        // the drop down list is a list view
        ListView listView = new ListView(activity);

        // set our adapter and pass our pop up window contents
        ArrayAdapter<String> aa = popupMenuAdapter(popUpContents);
        listView.setAdapter(aa);

        // set the item click listener
        listView.setOnItemClickListener(activity);

        // button resource ID, so we can trace back which button is pressed
        listView.setTag(resId);

        // get max text width
        Paint mTestPaint = new Paint();
        mTestPaint.setTextSize(listItemTextSize);
        float w = 0;  // max text width in pixel
        float wi;
        for (String popUpContent : popUpContents) {
            String sts[] = popUpContent.split("::");
            if (sts.length == 0) continue;
            String st = sts[0];
            if (sts.length == 2 && sts[1].equals("0")) {
                mTestPaint.setTextSize(listItemTitleTextSize);
                wi = mTestPaint.measureText(st);
                mTestPaint.setTextSize(listItemTextSize);
            } else {
                wi = mTestPaint.measureText(st);
            }
            if (w < wi) {
                w = wi;
            }
        }

        // left and right padding, at least +7, or the whole app will stop respond, don't know why
        w = w + 23 * DPRatio;
        if (w < 40 * DPRatio) {
            w = 40 * DPRatio;
        }

        // some other visual settings
        popupWindow.setFocusable(true);
        popupWindow.setHeight(WindowManager.LayoutParams.WRAP_CONTENT);
        // Set window width according to max text width
        popupWindow.setWidth((int)w);
        // also set button width
        ((Button) activity.findViewById(resId)).setWidth((int)(w + 5 * DPRatio));
        // Set the text on button in loadPreferenceForView()

        // set the list view as pop up window content
        popupWindow.setContentView(listView);

        return popupWindow;
    }

    /*
     * adapter where the list values will be set
     */
    private ArrayAdapter<String> popupMenuAdapter(String itemTagArray[]) {
        return new ArrayAdapter<String>(activity, android.R.layout.simple_list_item_1,
itemTagArray) {
            @NonNull
            @Override
            public View getView(int position, View convertView, @NonNull ViewGroup parent) {
                // setting the ID and text for every items in the list
                String item = getItem(position);
                String[] itemArr = item.split("::");
                String text = itemArr[0];
                String id = itemArr[1];

                // visual settings for the list item
                TextView listItem = new TextView(activity);

                if (id.equals("0")) {
                    listItem.setText(text);
                    listItem.setTag(id);
```

```
                     listItem.setTextSize(listItemTitleTextSize / DPRatio);
                    listItem.setPadding(5, 5, 5, 5);
                    listItem.setTextColor(Color.GREEN);
                    listItem.setGravity(android.view.Gravity.CENTER);
                } else {
                    listItem.setText(text);
                    listItem.setTag(id);
                    listItem.setTextSize(listItemTextSize / DPRatio);
                    listItem.setPadding(5, 5, 5, 5);
                    listItem.setTextColor(Color.WHITE);
                    listItem.setGravity(android.view.Gravity.CENTER);
                }

                return listItem;
            }
        };
    }

    private void refreshCursorLabel() {
        //double f1 = graphView.getCursorFreq();

        textCur.setLength(0);
        textCur.append(SamplingLoop.classIndicator.toString());//AV 02/07/2018
        /*textCur.append(activity.getString(R.string.text_cur));
        SBNumFormat.fillInNumFixedWidthPositive(textCur, f1, 5, 1);
        textCur.append("Hz(");
        AnalyzerUtil.freq2Cent(textCur, f1, " ");
        textCur.append(") ");
        SBNumFormat.fillInNumFixedWidth(textCur, graphView.getCursorDB(), 3, 1);
        textCur.append("dB");
        */
        textCur.getChars(0, Math.min(textCur.length(), textCurChar.length), textCurChar, 0);
        ((TextView) activity.findViewById(R.id.textview_cur)).setText(textCurChar, 0,
Math.min(textCur.length(), textCurChar.length));
        test=textCurChar.toString();
        setCurText();
    }

    private void refreshRMSLabel(double dtRMSFromFT) {
        textRMS.setLength(0);
        textRMS.append("RMS:dB \n");
        SBNumFormat.fillInNumFixedWidth(textRMS, 20*Math.log10(dtRMSFromFT), 3, 1);
        textRMS.getChars(0, Math.min(textRMS.length(), textRMSChar.length), textRMSChar, 0);
        TextView tv = (TextView) activity.findViewById(R.id.textview_RMS);
        tv.setText(textRMSChar, 0, textRMSChar.length);
        tv.invalidate();
    }

    private void refreshPeakLabel(double maxAmpFreq, double maxAmpDB) {
        textPeak.setLength(0);
        textPeak.append(activity.getString(R.string.text_peak));
        SBNumFormat.fillInNumFixedWidthPositive(textPeak, maxAmpFreq, 5, 1);
        textPeak.append("Hz(");
        AnalyzerUtil.freq2Cent(textPeak, maxAmpFreq, " ");
        textPeak.append(") ");
        SBNumFormat.fillInNumFixedWidth(textPeak, maxAmpDB, 3, 1);
        textPeak.append("dB");
        textPeak.getChars(0, Math.min(textPeak.length(), textPeakChar.length), textPeakChar, 0);

        TextView tv = (TextView) activity.findViewById(R.id.textview_peak);
        tv.setText(textPeakChar, 0, textPeakChar.length);
        tv.invalidate();
    }

    private void refreshRecTimeLable(double wavSec, double wavSecRemain) {
        // consist with @string/textview_rec_text
        textRec.setLength(0);
        textRec.append(activity.getString(R.string.text_rec));
        SBNumFormat.fillTime(textRec, wavSec, 1);
        textRec.append(activity.getString(R.string.text_remain));
        SBNumFormat.fillTime(textRec, wavSecRemain, 0);
        textRec.getChars(0, Math.min(textRec.length(), textRecChar.length), textRecChar, 0);
        ((TextView) activity.findViewById(R.id.textview_rec))
                .setText(textRecChar, 0, Math.min(textRec.length(), textRecChar.length));
    }

    private long timeToUpdate = SystemClock.uptimeMillis();
    private volatile boolean isInvalidating = false;

    // Invalidate graphView in a limited frame rate
    void invalidateGraphView() {
        invalidateGraphView(-1);
    }

    private static final int VIEW_MASK_graphView     = 1<<0;
    private static final int VIEW_MASK_textview_RMS  = 1<<1;
    private static final int VIEW_MASK_textview_peak = 1<<2;
```

```
    private static final int VIEW_MASK_CursorLabel   = 1<<3;
    private static final int VIEW_MASK_RecTimeLable  = 1<<4;

    private void invalidateGraphView(int viewMask) {
        if (isInvalidating) {
            return ;
        }
        isInvalidating = true;
        long frameTime;                      // time delay for next frame
        if (graphView.getShowMode() != AnalyzerGraphic.PlotMode.SPECTRUM) {
            frameTime = (long)(1000/fpsLimit);  // use a much lower frame rate for spectrogram
        } else {
            frameTime = 1000/60;
        }
        long t = SystemClock.uptimeMillis();
        //  && !graphView.isBusy()
        if (t >= timeToUpdate) {    // limit frame rate
            timeToUpdate += frameTime;
            if (timeToUpdate < t) {            // catch up current time
                timeToUpdate = t+frameTime;
            }
            idPaddingInvalidate = false;
            // Take care of synchronization of graphView.spectrogramColors and iTimePointer,
            // and then just do invalidate() here.
            if ((viewMask & VIEW_MASK_graphView) != 0)
                graphView.invalidate();
            // RMS
            if ((viewMask & VIEW_MASK_textview_RMS) != 0)
                refreshRMSLabel(activity.dtRMSFromFT);
            // peak frequency
            if ((viewMask & VIEW_MASK_textview_peak) != 0)
                refreshPeakLabel(activity.maxAmpFreq, activity.maxAmpDB);
            if ((viewMask & VIEW_MASK_CursorLabel) != 0)
                refreshCursorLabel();

            if ((viewMask & VIEW_MASK_RecTimeLable) != 0 && activity.samplingThread != null)
                refreshRecTimeLable(activity.samplingThread.wavSec,
activity.samplingThread.wavSecRemain);
        } else {
            if (! idPaddingInvalidate) {
                idPaddingInvalidate = true;
                paddingViewMask = viewMask;
                paddingInvalidateHandler.postDelayed(paddingInvalidateRunnable, timeToUpdate - t
+ 1);
```

```
            } else {
                paddingViewMask |= viewMask;
            }
        }
        isInvalidating = false;
    }

    void setFpsLimit(double _fpsLimit) {
        fpsLimit = _fpsLimit;
    }

    private volatile boolean idPaddingInvalidate = false;
    private volatile int paddingViewMask = -1;
    private Handler paddingInvalidateHandler = new Handler();

    // Am I need to use runOnUiThread() ?
    private final Runnable paddingInvalidateRunnable = new Runnable() {
        @Override
        public void run() {
            if (idPaddingInvalidate) {
                // It is possible that t-timeToUpdate <= 0 here, don't know why
                invalidateGraphView(paddingViewMask);
            }
        }
    };

    private void setCurText(){
        ((TextView) activity.findViewById(R.id.testingLetter)).setText(textCurChar, 0,
Math.min(textCur.length(), textCurChar.length));

        if (textCur.length() > 5){



            if (textCur.substring(0,5).contains("aaa") &&
(activity.samplingThread.peak3Ratio3Chunks > 50)){
                activity.rewardDog();
            }

    }}
```

```
}
```

AxisTickLabels

```
package github.bewantbe.audio_analyzer_for_android;

import android.graphics.Canvas;
import android.graphics.Paint;

class AxisTickLabels {
    private static final String[] axisNames = {"Hz", "dB", "Sec", "Hz", "Pitch"};  // See
GridLabel.Type

    // Draw ticks and labels for a axisMap.
    // labelPaint should use fixed width font
    static void draw(Canvas c, ScreenPhysicalMapping axisMap, GridLabel gridLabel,
                     float axisBeginX, float axisBeginY, int directionI, int labelSide,
                     Paint labelPaint, Paint gridPaint, Paint rulerBrightPaint) {
        String axisName = axisNames[gridLabel.getGridType().getValue()];

        float textHeight = labelPaint.getFontMetrics(null);
        float labelLargeLen = 0.7f * textHeight;
        float labelSmallLen = 0.6f * labelLargeLen;

        // directionI: 0:+x, 1:+y, 2:-x, 3:-y
        // labelSide:  1: label at positive side of axis, -1: otherwise
        boolean drawOnXAxis = directionI % 2 == 0;
        int directionSign = directionI <=1 ? 1 : -1;

        // Plot axis marks
        float posAlongAxis;
        for (int k = 0; k < 2; k ++) {
            float labelLen   = (k == 0 ? labelSmallLen : labelLargeLen) * labelSide;
            Paint tickPainter = k == 0 ? gridPaint : rulerBrightPaint;
            double[] values   = k == 0 ? gridLabel.ticks : gridLabel.values;
            for (int i = 0; i < values.length; i++) {
                posAlongAxis = (float)axisMap.pixelFromV(values[i]) * directionSign;
                if (drawOnXAxis) {
                    c.drawLine(axisBeginX + posAlongAxis, axisBeginY,
                               axisBeginX + posAlongAxis, axisBeginY + labelLen, tickPainter);
                } else {
```

```
          c.drawLine(axisBeginX,           axisBeginY + posAlongAxis,
                axisBeginX + labelLen, axisBeginY + posAlongAxis, tickPainter);
        }
      }
    }
    // Straight line
    if (drawOnXAxis) {
      c.drawLine(axisBeginX, axisBeginY, axisBeginX + (float)axisMap.nCanvasPixel * (1-
directionI), axisBeginY, labelPaint);
    } else {
      c.drawLine(axisBeginX, axisBeginY, axisBeginX, axisBeginY +
(float)axisMap.nCanvasPixel * (2-directionI), labelPaint);
    }

    // Plot labels
    float widthDigit = labelPaint.measureText("0");
    float widthAxisName = widthDigit * axisName.length();
    float widthAxisNameExt = widthAxisName +.5f*widthDigit;  // with a safe boundary

    // For drawOnXAxis == true
    float axisNamePosX = directionSign==1
        ? - widthAxisNameExt + (float)axisMap.nCanvasPixel
        : - widthAxisNameExt;
    // For drawOnXAxis == false
    // always show y-axis name at the smaller (in pixel) position.
    float axisNamePosY = directionSign==1
        ? textHeight
        : textHeight - (float)axisMap.nCanvasPixel;
    if (gridLabel.getGridType() == GridLabel.Type.DB) {
      // For dB axis, show axis name far from 0dB (directionSign==1)
      axisNamePosY = (float)axisMap.nCanvasPixel - 0.8f*widthDigit;
    }

    float labelPosY = axisBeginY + ( labelSide == 1 ? 0.1f*labelLargeLen + textHeight : -
0.3f*labelLargeLen);
    float labelPosX;
    int notShowNextLabel = 0;

    for(int i = 0; i < gridLabel.strings.length; i++) {
      posAlongAxis = (float)axisMap.pixelFromV(gridLabel.values[i]) * directionSign;
      float thisDigitWidth = drawOnXAxis ? widthDigit*gridLabel.strings[i].length() +
0.3f*widthDigit
                        : -textHeight;
      float axisNamePos = drawOnXAxis ? axisNamePosX
```

```
                      : axisNamePosY;
      float axisNameLen = drawOnXAxis ? widthAxisNameExt
                      : -textHeight;

      // Avoid label overlap:
      // (1) No overlap to axis name like "Hz";
      // (2) If no (1), no overlap to important label 1, 10, 100, 1000, 10000, 1k, 10k;
      // (3) If no (1) and (2), no overlap to previous label.
      if (isIntvOverlap(posAlongAxis, thisDigitWidth,
          axisNamePos, axisNameLen)) {
        continue;  // case (1)
      }
      if (notShowNextLabel > 0) {
        notShowNextLabel--;
        continue;  // case (3)
      }
      int j = i+1;
      while (j < gridLabel.strings.length) {
        float nextDigitPos = (float)axisMap.pixelFromV(gridLabel.values[j]) * directionSign;
        float nextDigitWidth = drawOnXAxis ? widthDigit*gridLabel.strings[j].length() +
0.3f*widthDigit
                         : -textHeight;
        if (! isIntvOverlap(posAlongAxis, thisDigitWidth,
            nextDigitPos, nextDigitWidth)) {
          break;  // no overlap of case (3)
        }
        notShowNextLabel++;
        if (gridLabel.isImportantLabel(j)) {
          // do not show label i (case (2))
          // but also check case (1) for label j
          if (! isIntvOverlap(nextDigitPos, nextDigitWidth,
              axisNamePos, axisNameLen)) {
            notShowNextLabel = -1;
            break;
          }
        }
        j++;
      }
      if (notShowNextLabel == -1) {
        notShowNextLabel = j - i - 1;  // show the label in case (2)
        continue;
      }

      // Now safe to draw label
```

```
            if (drawOnXAxis) {
                c.drawText(gridLabel.chars[i], 0, gridLabel.strings[i].length(),
                           axisBeginX + posAlongAxis, labelPosY, labelPaint);
            } else {
                labelPosX = labelSide == -1
                        ? axisBeginX - 0.5f * labelLargeLen - widthDigit * gridLabel.strings[i].length()
                        : axisBeginX + 0.5f * labelLargeLen;
                c.drawText(gridLabel.chars[i], 0, gridLabel.strings[i].length(),
                           labelPosX, axisBeginY + posAlongAxis, labelPaint);
            }
        }
        if (drawOnXAxis) {
            c.drawText(axisName, axisBeginX + axisNamePosX, labelPosY, labelPaint);
        } else {
            labelPosX = labelSide == -1
                    ? axisBeginX - 0.5f * labelLargeLen - widthAxisName
                    : axisBeginX + 0.5f * labelLargeLen;
            c.drawText(axisName, labelPosX, axisBeginY + axisNamePosY, labelPaint);
        }

    }

    // Return true if two intervals [pos1, pos1+len1] [pos2, pos2+len2] overlaps.
    private static boolean isIntvOverlap(float pos1, float len1, float pos2, float len2) {
        if (len1 < 0) { pos1 -= len1; len1 = -len1; }
        if (len2 < 0) { pos2 -= len2; len2 = -len2; }
        return pos1 <= pos2 && pos2 <= pos1+len1 || pos2 <= pos1 && pos1 <= pos2+len2;
    }
}
```

besselCal

```
package github.bewantbe.audio_analyzer_for_android;

class besselCal {
/*
Copyright © 1999 CERN - European Organization for Nuclear Research.
Permission to use, copy, modify, distribute and sell this software and its documentation for any
purpose
is hereby granted without fee, provided that the above copyright notice appear in all copies and
that both that copyright notice and this permission notice appear in supporting documentation.
CERN makes no representations about the suitability of this software for any purpose.
It is provided "as is" without expressed or implied warranty.
*/
```

```
// https://github.com/carlsonp/Colt/blob/master/src/cern/jet/math/Arithmetic.java
// package cern.jet.math;
  /**
   * Evaluates the series of Chebyshev polynomials Ti at argument x/2.
   * The series is given by
   * <pre>
   *       N-1
   *        - '
   *  y  =   >   coef[i] T (x/2)
   *        -            i
   *       i=0
   * </pre>
   * Coefficients are stored in reverse order, i.e. the zero
   * order term is last in the array.  Note N is the number of
   * coefficients, not the order.
   * <p>
   * If coefficients are for the interval a to b, x must
   * have been transformed to x -> 2(2x - b - a)/(b-a) before
   * entering the routine.  This maps x from (a, b) to (-1, 1),
   * over which the Chebyshev polynomials are defined.
   * <p>
   * If the coefficients are for the inverted interval, in
   * which (a, b) is mapped to (1/b, 1/a), the transformation
   * required is x -> 2(2ab/x - b - a)/(b-a).  If b is infinity,
   * this becomes x -> 4a/x - 1.
   * <p>
   * SPEED:
   * <p>
   * Taking advantage of the recurrence properties of the
   * Chebyshev polynomials, the routine requires one more
   * addition per loop than evaluating a nested polynomial of
   * the same degree.
   *
   * @param x argument to the polynomial.
   * @param coef the coefficients of the polynomial.
   * @param N the number of coefficients.
   */
  static double chbevl(double x, double coef[], int N) throws ArithmeticException {
    double b0, b1, b2;

    int p = 0;
    int i;

    b0 = coef[p++];
    b1 = 0.0;
    i = N - 1;

    do {
      b2 = b1;
      b1 = b0;
      b0 = x * b1  -  b2  + coef[p++];
    } while( --i > 0);

    return( 0.5*(b0-b2) );
  }

// https://github.com/carlsonp/Colt/blob/master/src/cern/jet/math/Bessel.java
// package cern.jet.math;

  private static final double[] A_i0 = {
        -4.41534164647933937950E-18,
        3.33079451882223809783E-17,
        -2.43127984654795469359E-16,
        1.71539128555513303061E-15,
        -1.16853328779934516808E-14,
        7.67618549860493561688E-14,
        -4.85644678311192946090E-13,
        2.95505266312963983461E-12,
        -1.72682629144155570723E-11,
        9.67580903537323691224E-11,
        -5.18979560163526290666E-10,
        2.65982372468238665035E-9,
        -1.30002500998624804212E-8,
        6.04699502254191894932E-8,
        -2.67079385394061173391E-7,
        1.11738753912010371815E-6,
        -4.41673835845875056359E-6,
        1.64484480707288970893E-5,
        -5.75419501008210370398E-5,
        1.88502885095841655729E-4,
        -5.76375574538582365885E-4,
        1.63947561694133579842E-3,
        -4.32430999505057594430E-3,
        1.05464603945949983183E-2,
        -2.37374148058994688156E-2,
        4.93052842396707084878E-2,
        -9.49010970480476444210E-2,
        1.71620901522208775349E-1,
        -3.04682672343198398683E-1,
        6.76795274409476084995E-1
};

/**
 * Chebyshev coefficients for exp(-x) sqrt(x) I0(x)
 * in the inverted interval [8,infinity].
 *
 * lim(x->inf){ exp(-x) sqrt(x) I0(x) } = 1/sqrt(2pi).
 */
private static final double[] B_i0 = {
        -7.23318048787475395456E-18,
        -4.83050448594418207126E-18,
        4.46562142029675999901E-17,
        3.46122286769746109310E-17,
        -2.82762398051658348494E-16,
        -3.42548561967721913462E-16,
        1.77256013305652638360E-15,
        3.81168066935262242075E-15,
        -9.55484669882830764870E-15,
        -4.15056934728722208663E-14,
        1.54008621752140982691E-14,
        3.85277838274214270114E-13,
        7.18012445138366623367E-13,
        -1.79417853150680611778E-12,
        -1.32158118404477131188E-11,
        -3.14991652796324136454E-11,
        1.18891471078464383424E-11,
        4.94060238822496958910E-10,
        3.39623202570838634515E-9,
        2.26666899049817806459E-8,
        2.04891858946906374183E-7,
        2.89137052083475648297E-6,
        6.88975834691682398426E-5,
        3.36911647825569408990E-3,
        8.04490411014108831608E-1
};

/**
 * Returns the modified Bessel function of order 0 of the
 * argument.
 * <p>
 * The function is defined as <tt>i0(x) = j0( ix )</tt>.
 * <p>
```

```
   * The range is partitioned into the two intervals [0,8] and
   * (8, infinity).  Chebyshev polynomial expansions are employed
   * in each interval.
   *
   * @param x the value to compute the bessel function of.
   */
  static double i0(double x) throws ArithmeticException {
    double y;
    if( x < 0 ) x = -x;
    if( x <= 8.0 ) {
      y = (x/2.0) - 2.0;
      return( Math.exp(x) * chbevl( y, A_i0, 30 ) );
    }

    return(  Math.exp(x) * chbevl( 32.0/x - 2.0, B_i0, 25 ) / Math.sqrt(x) );
  }

}
```

CalibrationLoad

```
package github.bewantbe.audio_analyzer_for_android;

import android.content.Context;
import android.net.Uri;
import android.util.Log;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Iterator;

/**
 * Load calibration file.
 */

class CalibrationLoad {
    final static String TAG = "CalibrationLoad";

    double[] freq = new double[0];
    double[] gain = new double[0];
    double centralFreq = 1000;
double centralGain = -37.4;
String name = null;

void loadFile(Uri calibUri, Context context) {
    String calibPath = calibUri.getPath();
    BufferedReader br;
    InputStream inputStream;
    try {
        inputStream = context.getContentResolver().openInputStream(calibUri);
    } catch (FileNotFoundException e) {
        Log.e(TAG, "no calib file found: " + calibPath);
        return;
    }
    if (inputStream == null) {
        Log.e(TAG, "open calib file fail: " + calibPath);
        return;
    }
    br = new BufferedReader(new InputStreamReader(inputStream));

    String line = null;
    try {
        int lineCount = 0;
        ArrayList<Double> freqList = new ArrayList<>();
        ArrayList<Double> amplitudeDBList = new ArrayList<>();
        while (true) {
            line = br.readLine();
            if (line == null) break;
            lineCount ++;
            line = line.trim();
            if (line.length() == 0) {  // skip empty line
                continue;
            }
            char c = line.charAt(0);
            if ('0' <= c && c <= '9' || c == '.' || c == '-') {  // Should be a number
                // 20.00   -4.2
                String[] strs = line.split("[ \t]+");
                if (strs.length != 2) {
                    Log.w(TAG, "Fail to parse line " + lineCount + " :" + line);
                    continue;
                }
                freqList       .add(Double.valueOf(strs[0]));
                amplitudeDBList.add(Double.valueOf(strs[1]));
            } else if (line.charAt(0) == '*') {  // Dayton Audio txt/cal or miniDSP cal file
                // parse Only the Dayton Audio header:
```

```
          //*1000Hz          -37.4
          String[] strs = line.substring(1).split("Hz[ \t]+");
          if (strs.length == 2) {
            Log.i(TAG, "Dayton Audio header");
            centralFreq = Double.valueOf(strs[0]);
            centralGain = Double.valueOf(strs[1]);
          }
          // miniDSP cal header:
          //* miniDSP PMIK-1 calibration file, serial: 8000234, format: cal
          // Skip
        } else if (line.charAt(0) == '"') {
          // miniDSP txt header:
          //"miniDSP PMIK-1 calibration file, serial: 8000234, format: frd"
          // Skip
          // miniDSP frd header:
          //"miniDSP PMIK-1 calibration file, serial: 8000234, format: frd"
          // Skip
        } else if (line.charAt(0) == '#') {
          // Shell style comment line
          // Skip
        } else {
          Log.e(TAG, "Bad calibration file.");
          freqList.clear();
          amplitudeDBList.clear();
          break;
        }
      }
      br.close();

      freq = new double[freqList.size()];
      gain = new double[freqList.size()];
      Iterator itr  = freqList.iterator();
      Iterator itr2 = amplitudeDBList.iterator();
      for (int j=0; itr.hasNext(); j++ ) {
        freq[j] = (double)itr.next();
        gain[j] = (double)itr2.next();
      }
    } catch (IOException e) {
      Log.e(TAG, "Fail to read file: " + calibPath);
    }
    name = calibPath.substring(calibPath.lastIndexOf("/")+1);
  }
}
```

ChipBaseFragment

```
package github.bewantbe.audio_analyzer_for_android;
import android.support.v4.app.Fragment;

import com.wowwee.bluetoothrobotcontrollib.BluetoothRobotConstants;
import com.wowwee.bluetoothrobotcontrollib.chip.ChipRobot;

/**
 * Created by davidchan on 24/3/2017.
 */

public class ChipBaseFragment extends Fragment implements ChipRobot.ChipRobotInterface {

    @Override
    public void chipDeviceReady(ChipRobot chip) {

    }

    @Override
    public void chipDeviceDisconnected(ChipRobot chip) {

    }

    @Override
    public void chipDidReceiveVolumeLevel(ChipRobot chip, byte volume) {

    }

    @Override
    public void chipDidReceivedBatteryLevel(ChipRobot chip, float batteryLevel, byte
chargingStatus, byte chargerType) {

    }

    @Override
    public void chipDidReceiveDogVersionWithBodyHardware(int chipVerBodyHardwareVer, int
chipVerHeadHardware, int chipVerMechanic, int chipVerBleSpiFlash, int
chipVerNuvotonSpiFlash, int chipVerBleBootloader, int chipVerBleApromFirmware, int
chipVerNuvotonBootloaderFirmware, int chipVerNuvotonApromFirmware, int
chipVerNuvotonVR) {

    }

    @Override
    public void chipDidSendAdultOrKidSpeed() {

    }

    @Override
    public void chipDidReceiveAdultOrKidSpeed(byte speed) {

    }

    @Override
    public void chipDidReceiveEyeRGBBrightness(byte value) {

    }

    @Override
    public void chipDidReceiveCurrentClock(int year, int month, int day, int hour, int minute, int
seconds, int dayOfWeek) {

    }

    @Override
    public void chipDidReceiveCurrentAlarm(int year, int month, int day, int hour, int minute) {

    }

    @Override
    public void chipDidReceiveBodyconStatus(int status) {

    }
}
```

ColorMapArray

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.util.Log;

// See also res/values/arrays.xml
class ColorMapArray {

    static int[] selectColorMap(String colorMapName) {
        switch (colorMapName.toLowerCase()) {
            case "hot":
                return hot;
            case "jet":
                return jet;
            case "gray":
                return gray;
            case "blackbody_uniform":
                return blackbody_uniform;
            case "parula":
                return parula;
            case "magma":
                return magma;
            case "inferno":
                return inferno;
            case "plasma":
                return plasma;
            case "viridis":
                return viridis;
            default:
                Log.i("ColorMapArray:", "No this color map: " + colorMapName);
                return hot;
        }
    }

    // Generated by GNU Octave
    //   c=[0 0 0; hot(0xff)];
    //   v=int32(floor(flipud(c)*255.99)*[0x10000 0x100 1]');
    //   s=reshape(sprintf('0x%06x, ', v), 10*8, [])(1:end-1,:)';  s(end)=' '
    // Seems that Octave changed their hot map (to match Matlab?), this is the new map.
    static final int[] hot = {
        0xffffff, 0xfffffc, 0xfffff8, 0xfffff4, 0xfffff0, 0xffffec, 0xffffe8, 0xffffe4,
        0xffffe0, 0xffffdc, 0xffffd8, 0xffffd4, 0xffffd0, 0xffffcc, 0xffffc8, 0xffffc4,
    0xffffc0, 0xffffbd, 0xffffb9, 0xffffb5, 0xffffb1, 0xffffad, 0xffffa9, 0xffffa5,
    0xffffa1, 0xffff9d, 0xffff99, 0xffff95, 0xffff91, 0xffff8d, 0xffff89, 0xffff85,
    0xffff81, 0xffff7e, 0xffff7a, 0xffff76, 0xffff72, 0xffff6e, 0xffff6a, 0xffff66,
    0xffff62, 0xffff5e, 0xffff5a, 0xffff56, 0xffff52, 0xffff4e, 0xffff4a, 0xffff46,
    0xffff42, 0xffff3f, 0xffff3b, 0xffff37, 0xffff33, 0xffff2f, 0xffff2b, 0xffff27,
    0xffff23, 0xffff1f, 0xffff1b, 0xffff17, 0xffff13, 0xffff0f, 0xffff0b, 0xffff07,
    0xffff03, 0xffff00, 0xfffd00, 0xfffa00, 0xfff700, 0xfff500, 0xfff200, 0xffef00,
    0xffed00, 0xffea00, 0xffe700, 0xffe500, 0xffe200, 0xffdf00, 0xffdc00, 0xffda00,
    0xffd700, 0xffd400, 0xffd200, 0xffcf00, 0xffcc00, 0xffca00, 0xffc700, 0xffc400,
    0xffc200, 0xffbf00, 0xffbc00, 0xffb900, 0xffb700, 0xffb400, 0xffb100, 0xffaf00,
    0xffac00, 0xffa900, 0xffa700, 0xffa400, 0xffa100, 0xff9e00, 0xff9c00, 0xff9900,
    0xff9600, 0xff9400, 0xff9100, 0xff8e00, 0xff8c00, 0xff8900, 0xff8600, 0xff8400,
    0xff8100, 0xff7e00, 0xff7b00, 0xff7900, 0xff7600, 0xff7300, 0xff7100, 0xff6e00,
    0xff6b00, 0xff6900, 0xff6600, 0xff6300, 0xff6100, 0xff5e00, 0xff5b00, 0xff5800,
    0xff5600, 0xff5300, 0xff5000, 0xff4e00, 0xff4b00, 0xff4800, 0xff4600, 0xff4300,
    0xff4000, 0xff3d00, 0xff3b00, 0xff3800, 0xff3500, 0xff3300, 0xff3000, 0xff2d00,
    0xff2b00, 0xff2800, 0xff2500, 0xff2300, 0xff2000, 0xff1d00, 0xff1a00, 0xff1800,
    0xff1500, 0xff1200, 0xff1000, 0xff0d00, 0xff0a00, 0xff0800, 0xff0500, 0xff0200,
    0xff0000, 0xfd0000, 0xfa0000, 0xf70000, 0xf50000, 0xf20000, 0xef0000, 0xed0000,
    0xea0000, 0xe70000, 0xe50000, 0xe20000, 0xdf0000, 0xdc0000, 0xda0000, 0xd70000,
    0xd40000, 0xd20000, 0xcf0000, 0xcc0000, 0xca0000, 0xc70000, 0xc40000, 0xc20000,
    0xbf0000, 0xbc0000, 0xb90000, 0xb70000, 0xb40000, 0xb10000, 0xaf0000, 0xac0000,
    0xa90000, 0xa70000, 0xa40000, 0xa10000, 0x9e0000, 0x9c0000, 0x990000, 0x960000,
    0x940000, 0x910000, 0x8e0000, 0x8c0000, 0x890000, 0x860000, 0x840000, 0x810000,
    0x7e0000, 0x7b0000, 0x790000, 0x760000, 0x730000, 0x710000, 0x6e0000, 0x6b0000,
    0x690000, 0x660000, 0x630000, 0x610000, 0x5e0000, 0x5b0000, 0x580000, 0x560000,
    0x530000, 0x500000, 0x4e0000, 0x4b0000, 0x480000, 0x460000, 0x430000, 0x400000,
    0x3d0000, 0x3b0000, 0x380000, 0x350000, 0x330000, 0x300000, 0x2d0000, 0x2b0000,
    0x280000, 0x250000, 0x230000, 0x200000, 0x1d0000, 0x1a0000, 0x180000, 0x150000,
    0x120000, 0x100000, 0x0d0000, 0x0a0000, 0x080000, 0x050000, 0x020000, 0x000000
};

// c=[0 0 0; jet(0xff)];
static final int[] jet = {
    0x830000, 0x870000, 0x8b0000, 0x8f0000, 0x930000, 0x970000, 0x9b0000, 0x9f0000,
    0xa30000, 0xa70000, 0xab0000, 0xaf0000, 0xb30000, 0xb70000, 0xbb0000, 0xbf0000,
    0xc30000, 0xc70000, 0xcb0000, 0xcf0000, 0xd30000, 0xd70000, 0xdb0000, 0xdf0000,
    0xe30000, 0xe70000, 0xeb0000, 0xef0000, 0xf30000, 0xf70000, 0xfb0000, 0xff0000,
    0xff0300, 0xff0700, 0xff0b00, 0xff0f00, 0xff1300, 0xff1700, 0xff1b00, 0xff1f00,
    0xff2300, 0xff2700, 0xff2b00, 0xff2f00, 0xff3300, 0xff3700, 0xff3b00, 0xff3f00,
    0xff4300, 0xff4700, 0xff4b00, 0xff4f00, 0xff5300, 0xff5700, 0xff5b00, 0xff5f00,
    0xff6300, 0xff6700, 0xff6b00, 0xff6f00, 0xff7300, 0xff7700, 0xff7b00, 0xff7f00,
    0xff8300, 0xff8700, 0xff8b00, 0xff8f00, 0xff9300, 0xff9700, 0xff9b00, 0xff9f00,
    0xffa300, 0xffa700, 0xffab00, 0xffaf00, 0xffb300, 0xffb700, 0xffbb00, 0xffbf00,
    0xffc300, 0xffc700, 0xffcb00, 0xffcf00, 0xffd300, 0xffd700, 0xffdb00, 0xffdf00,
    0xffe300, 0xffe700, 0xffeb00, 0xffef00, 0xfff300, 0xfff700, 0xfffb00, 0xffff00,
    0xfbff03, 0xf7ff07, 0xf3ff0b, 0xefff0f, 0xebff13, 0xe7ff17, 0xe3ff1b, 0xdfff1f,
    0xdbff23, 0xd7ff27, 0xd3ff2b, 0xcfff2f, 0xcbff33, 0xc7ff37, 0xc3ff3b, 0xbfff3f,
    0xbbff43, 0xb7ff47, 0xb3ff4b, 0xafff4f, 0xabff53, 0xa7ff57, 0xa3ff5b, 0x9fff5f,
    0x9bff63, 0x97ff67, 0x93ff6b, 0x8fff6f, 0x8bff73, 0x87ff77, 0x83ff7b, 0x7fff7f,
    0x7bff83, 0x77ff87, 0x73ff8b, 0x6fff8f, 0x6bff93, 0x67ff97, 0x63ff9b, 0x5fff9f,
    0x5bffa3, 0x57ffa7, 0x53ffab, 0x4fffaf, 0x4bffb3, 0x47ffb7, 0x43ffbb, 0x3fffbf,
    0x3bffc3, 0x37ffc7, 0x33ffcb, 0x2fffcf, 0x2bffd3, 0x27ffd7, 0x23ffdb, 0x1fffdf,
    0x1bffe3, 0x17ffe7, 0x13ffeb, 0x0fffef, 0x0bfff3, 0x07fff7, 0x03fffb, 0x00ffff,
    0x00fbff, 0x00f7ff, 0x00f3ff, 0x00efff, 0x00ebff, 0x00e7ff, 0x00e3ff, 0x00dfff,
    0x00dbff, 0x00d7ff, 0x00d3ff, 0x00cfff, 0x00cbff, 0x00c7ff, 0x00c3ff, 0x00bfff,
    0x00bbff, 0x00b7ff, 0x00b3ff, 0x00afff, 0x00abff, 0x00a7ff, 0x00a3ff, 0x009fff,
    0x009bff, 0x0097ff, 0x0093ff, 0x008fff, 0x008bff, 0x0087ff, 0x0083ff, 0x007fff,
    0x007bff, 0x0077ff, 0x0073ff, 0x006fff, 0x006bff, 0x0067ff, 0x0063ff, 0x005fff,
    0x005bff, 0x0057ff, 0x0053ff, 0x004fff, 0x004bff, 0x0047ff, 0x0043ff, 0x003fff,
    0x003bff, 0x0037ff, 0x0033ff, 0x002fff, 0x002bff, 0x0027ff, 0x0023ff, 0x001fff,
    0x001bff, 0x0017ff, 0x0013ff, 0x000fff, 0x000bff, 0x0007ff, 0x0003ff, 0x0000ff,
    0x0000fb, 0x0000f7, 0x0000f3, 0x0000ef, 0x0000eb, 0x0000e7, 0x0000e3, 0x0000df,
    0x0000db, 0x0000d7, 0x0000d3, 0x0000cf, 0x0000cb, 0x0000c7, 0x0000c3, 0x0000bf,
    0x0000bb, 0x0000b7, 0x0000b3, 0x0000af, 0x0000ab, 0x0000a7, 0x0000a3, 0x00009f,
    0x00009b, 0x000097, 0x000093, 0x00008f, 0x00008b, 0x000087, 0x000083, 0x000000
};

// c=[0 0 0; gray(0xff)];
static final int[] gray = {
    0xffffff, 0xfefefe, 0xfdfdfd, 0xfcfcfc, 0xfbfbfb, 0xfafafa, 0xf9f9f9, 0xf8f8f8,
    0xf7f7f7, 0xf6f6f6, 0xf5f5f5, 0xf4f4f4, 0xf3f3f3, 0xf2f2f2, 0xf1f1f1, 0xf0f0f0,
    0xefefef, 0xeeeeee, 0xededed, 0xececec, 0xebebeb, 0xeaeaea, 0xe9e9e9, 0xe8e8e8,
    0xe7e7e7, 0xe6e6e6, 0xe5e5e5, 0xe4e4e4, 0xe3e3e3, 0xe2e2e2, 0xe1e1e1, 0xe0e0e0,
    0xdfdfdf, 0xdedede, 0xdddddd, 0xdcdcdc, 0xdbdbdb, 0xdadada, 0xd9d9d9, 0xd8d8d8,
    0xd7d7d7, 0xd6d6d6, 0xd5d5d5, 0xd4d4d4, 0xd3d3d3, 0xd2d2d2, 0xd1d1d1, 0xd0d0d0,
    0xcfcfcf, 0xcecece, 0xcdcdcd, 0xcccccc, 0xcbcbcb, 0xcacaca, 0xc9c9c9, 0xc8c8c8,
    0xc7c7c7, 0xc6c6c6, 0xc5c5c5, 0xc4c4c4, 0xc3c3c3, 0xc2c2c2, 0xc1c1c1, 0xc0c0c0,
    0xbfbfbf, 0xbebebe, 0xbdbdbd, 0xbcbcbc, 0xbbbbbb, 0xbababa, 0xb9b9b9, 0xb8b8b8,
    0xb7b7b7, 0xb6b6b6, 0xb5b5b5, 0xb4b4b4, 0xb3b3b3, 0xb2b2b2, 0xb1b1b1, 0xb0b0b0,
    0xafafaf, 0xaeaeae, 0xadadad, 0xacacac, 0xababab, 0xaaaaaa, 0xa9a9a9, 0xa8a8a8,
    0xa7a7a7, 0xa6a6a6, 0xa5a5a5, 0xa4a4a4, 0xa3a3a3, 0xa2a2a2, 0xa1a1a1, 0xa0a0a0,
    0x9f9f9f, 0x9e9e9e, 0x9d9d9d, 0x9c9c9c, 0x9b9b9b, 0x9a9a9a, 0x999999, 0x989898,
    0x979797, 0x969696, 0x959595, 0x949494, 0x939393, 0x929292, 0x919191, 0x909090,
    0x8f8f8f, 0x8e8e8e, 0x8d8d8d, 0x8c8c8c, 0x8b8b8b, 0x8a8a8a, 0x898989, 0x888888,
    0x878787, 0x868686, 0x858585, 0x848484, 0x838383, 0x828282, 0x818181, 0x7f7f7f,
    0x7e7e7e, 0x7d7d7d, 0x7c7c7c, 0x7b7b7b, 0x7a7a7a, 0x797979, 0x787878, 0x777777,
    0x767676, 0x757575, 0x747474, 0x737373, 0x727272, 0x717171, 0x707070, 0x6f6f6f,
    0x6e6e6e, 0x6d6d6d, 0x6c6c6c, 0x6b6b6b, 0x6a6a6a, 0x696969, 0x686868, 0x676767,
    0x666666, 0x656565, 0x646464, 0x636363, 0x626262, 0x616161, 0x606060, 0x5f5f5f,
    0x5e5e5e, 0x5d5d5d, 0x5c5c5c, 0x5b5b5b, 0x5a5a5a, 0x595959, 0x585858, 0x575757,
    0x565656, 0x555555, 0x545454, 0x535353, 0x525252, 0x515151, 0x505050, 0x4f4f4f,
    0x4e4e4e, 0x4d4d4d, 0x4c4c4c, 0x4b4b4b, 0x4a4a4a, 0x494949, 0x484848, 0x474747,
    0x464646, 0x454545, 0x444444, 0x434343, 0x424242, 0x414141, 0x404040, 0x3f3f3f,
    0x3e3e3e, 0x3d3d3d, 0x3c3c3c, 0x3b3b3b, 0x3a3a3a, 0x393939, 0x383838, 0x373737,
    0x363636, 0x353535, 0x343434, 0x333333, 0x323232, 0x313131, 0x303030, 0x2f2f2f,
    0x2e2e2e, 0x2d2d2d, 0x2c2c2c, 0x2b2b2b, 0x2a2a2a, 0x292929, 0x282828, 0x272727,
    0x262626, 0x252525, 0x242424, 0x232323, 0x222222, 0x212121, 0x202020, 0x1f1f1f,
    0x1e1e1e, 0x1d1d1d, 0x1c1c1c, 0x1b1b1b, 0x1a1a1a, 0x191919, 0x181818, 0x171717,
    0x161616, 0x151515, 0x141414, 0x131313, 0x121212, 0x111111, 0x101010, 0x0f0f0f,
    0x0e0e0e, 0x0d0d0d, 0x0c0c0c, 0x0b0b0b, 0x0a0a0a, 0x090909, 0x080808, 0x070707,
    0x060606, 0x050505, 0x040404, 0x030303, 0x020202, 0x010101, 0x000000, 0x000000
  };

  // Generated by Octave script util/blackbody.m and cm_list.py
  // With help of https://github.com/bewantbe/colormap_uniformize
  static final int[] blackbody_uniform = {
      0xd2e2ff, 0xd4e1ff, 0xd5e1ff, 0xd6e0ff, 0xd7e0ff, 0xd8dffd, 0xd8dffb, 0xd9dff9,
      0xd9dff7, 0xd9def5, 0xdadef4, 0xdadef2, 0xdbdef0, 0xdbddee, 0xdbdded, 0xdcddeb,
      0xdcdcea, 0xdddce8, 0xdddce7, 0xdddce5, 0xdedbe4, 0xdedbe3, 0xdedbe1, 0xdfdbe0,
      0xdfdadf, 0xdfdade, 0xdfdadd, 0xe0dadc, 0xe0dada, 0xe0d9d9, 0xe1d9d8, 0xe1d9d6,
      0xe1d8d4, 0xe2d8d2, 0xe2d7d0, 0xe3d7ce, 0xe3d6cc, 0xe4d6c9, 0xe4d5c7, 0xe5d5c5,
      0xe5d4c3, 0xe6d4c0, 0xe6d3be, 0xe7d2bb, 0xe8d2b9, 0xe8d1b6, 0xe9d0b4, 0xe9cfb1,
      0xeaceae, 0xeaceac, 0xebcda9, 0xebcca6, 0xeccba4, 0xeccaa1, 0xedc99e, 0xedc89b,
      0xeec798, 0xeec695, 0xefc592, 0xefc48f, 0xefc38d, 0xf0c28a, 0xf0c087, 0xf1bf84,
      0xf1be81, 0xf1bd7d, 0xf2bb7a, 0xf2ba77, 0xf2b974, 0xf2b771, 0xf3b66e, 0xf3b46b,
      0xf3b368, 0xf3b165, 0xf3b062, 0xf3ae5f, 0xf3ad5c, 0xf4ab59, 0xf4a956, 0xf4a852,
      0xf3a64f, 0xf3a44c, 0xf3a349, 0xf3a146, 0xf39f43, 0xf39d40, 0xf39c3d, 0xf29a3a,
      0xf29837, 0xf29634, 0xf19530, 0xf1932d, 0xf1912a, 0xf08f27, 0xf08d23, 0xef8b20,
      0xef8a1c, 0xee8818, 0xee8614, 0xed840f, 0xed8209, 0xec8003, 0xec7e00, 0xeb7d00,
      0xeb7b00, 0xea7900, 0xe97700, 0xe97600, 0xe87400, 0xe77200, 0xe77000, 0xe66f00,
      0xe56d00, 0xe56b00, 0xe46a00, 0xe36800, 0xe26600, 0xe16400, 0xe16200,
0xe06100,
      0xdf5f00, 0xde5d00, 0xdd5b00, 0xdc5a00, 0xdc5800, 0xdb5600, 0xda5400, 0xd95200,
      0xd85100, 0xd74f00, 0xd64d00, 0xd54b00, 0xd44900, 0xd34800, 0xd24600, 0xd14400,
      0xd04200, 0xcf4000, 0xce3e00, 0xcd3c00, 0xcc3a00, 0xcb3800, 0xc93600, 0xc83400,
      0xc73200, 0xc63000, 0xc52e00, 0xc42c00, 0xc22a00, 0xc12700, 0xc02500, 0xbf2300,
      0xbd2000, 0xbc1e00, 0xbb1b00, 0xb91800, 0xb81600, 0xb71200, 0xb50f00, 0xb40b00,
      0xb20600, 0xb10200, 0xaf0000, 0xad0000, 0xab0000, 0xa90000, 0xa60000, 0xa40000,
      0xa20000, 0xa00000, 0x9e0000, 0x9c0000, 0x9a0000, 0x980000, 0x960000,
0x940000,
      0x910000, 0x8f0000, 0x8d0000, 0x8b0000, 0x890000, 0x870000, 0x850000, 0x830000,
        0x810000, 0x7f0000, 0x7d0000, 0x7b0000, 0x790000, 0x770000, 0x750000, 0x730000,
        0x710000, 0x6f0000, 0x6d0000, 0x6b0000, 0x690000, 0x670000, 0x650000, 0x630000,
        0x610000, 0x5f0000, 0x5d0000, 0x5b0000, 0x590000, 0x570000, 0x550000, 0x530000,
        0x510000, 0x4f0000, 0x4d0000, 0x4b0000, 0x4a0000, 0x480000, 0x460000, 0x440000,
        0x420000, 0x400000, 0x3e0000, 0x3c0000, 0x3a0000, 0x390000, 0x370000,
0x350000,
        0x330000, 0x310000, 0x2f0000, 0x2e0000, 0x2c0000, 0x2a0000, 0x280000, 0x260000,
        0x250000, 0x230000, 0x210000, 0x200000, 0x1e0000, 0x1c0000, 0x1b0000,
0x190000,
        0x170000, 0x160000, 0x140000, 0x130000, 0x110000, 0x100000, 0x0e0000,
0x0d0000,
        0x0b0000, 0x0a0000, 0x090000, 0x070000, 0x060000, 0x050000, 0x050000, 0x000000
  };

  // Matlab's new default colormap (since 2014b)
  // http://blogs.mathworks.com/steve/2014/10/13/a-new-colormap-for-matlab-part-1-
introduction/
  // https://www.mathworks.com/content/dam/mathworks/tag-
team/Objects/r/81137_92238v00_RainbowColorMap_57312.pdf
  // c=[0 0 0; parula(255)];
  // v=int32(floor(flipud(c)*255.99)*[2^16 2^8 1]');
  // s=reshape(sprintf('0x%06x, ', v), 10*8, []);
  // s=s(1:end-1,:)';  s(end)=' '
  static final int[] parula = {
      0xf9fb0d, 0xf9f90f, 0xf8f710, 0xf7f512, 0xf7f313, 0xf6f114, 0xf6ef15, 0xf5ed16,
      0xf5ec18, 0xf5ea19, 0xf5e81a, 0xf5e61b, 0xf5e51c, 0xf5e31e, 0xf5e21f, 0xf6e020,
      0xf6df21, 0xf6dd22, 0xf7dc23, 0xf7da24, 0xf8d925, 0xf8d726, 0xf9d627, 0xf9d528,
      0xfad329, 0xfbd22a, 0xfbd12b, 0xfcd02c, 0xfcce2d, 0xfdcd2e, 0xfdcc2f, 0xfeca30,
      0xfec932, 0xffc833, 0xffc634, 0xffc535, 0xffc437, 0xffc238, 0xffc139, 0xffc03b,
      0xfebf3d, 0xfdbe3e, 0xfcbd40, 0xfbbc42, 0xf9bb43, 0xf8ba45, 0xf6ba46, 0xf4ba48,
      0xf2b949, 0xf0b94a, 0xeeb94b, 0xecb94c, 0xeab94d, 0xe8b94f, 0xe6b950, 0xe4ba50,
      0xe2ba51, 0xe0ba52, 0xdeba53, 0xdcba54, 0xdaba55, 0xd8ba56, 0xd6bb57, 0xd4bb58,
      0xd2bb59, 0xd0bb59, 0xcebb5a, 0xccbc5b, 0xcabc5c, 0xc8bc5d, 0xc6bc5e, 0xc3bc5f,
      0xc1bc5f, 0xbfbd60, 0xbdbd61, 0xbbbd62, 0xb9bd63, 0xb7bd64, 0xb4bd65, 0xb2be66,
      0xb0be66, 0xaebe67, 0xabbe68, 0xa9be69, 0xa7be6a, 0xa4be6b, 0xa2bf6c, 0xa0bf6d,
      0x9dbf6e, 0x9bbf6f, 0x98bf70, 0x96bf71, 0x93bf72, 0x91bf73, 0x8ebf74, 0x8cbf75,
      0x89bf76, 0x87bf77, 0x84bf78, 0x81bf79, 0x7ebf7a, 0x7cbf7c, 0x79bf7d, 0x76bf7e,
      0x73bf7f, 0x70bf81, 0x6dbf82, 0x6abf83, 0x67bf85, 0x64be86, 0x61be87, 0x5ebe89,
      0x5bbe8a, 0x58be8c, 0x55bd8d, 0x52bd8f, 0x50bd90, 0x4dbc92, 0x4abc93, 0x47bc95,
      0x44bb96, 0x42bb98, 0x3fba9a, 0x3cba9b, 0x3aba9d, 0x37b99e, 0x35b9a0, 0x32b8a1,
      0x30b8a2, 0x2eb7a4, 0x2bb7a5, 0x29b6a7, 0x27b6a8, 0x25b5aa, 0x23b5ab, 0x21b4ac,
      0x1fb4ae, 0x1db3af, 0x1bb3b0, 0x19b2b2, 0x17b1b3, 0x15b1b4, 0x14b0b5, 0x12b0b7,
      0x10afb8, 0x0fafb9, 0x0daeba, 0x0cadbb, 0x0badbd, 0x0aacbe, 0x09acbf, 0x08abc0,
      0x07aac1, 0x06aac2, 0x06a9c3, 0x06a8c4, 0x05a8c5, 0x05a7c6, 0x05a6c7, 0x05a6c8,
      0x05a5c9, 0x05a4ca, 0x05a3cb, 0x05a2cb, 0x06a2cc, 0x06a1cd, 0x06a0ce, 0x069fce,
      0x069ecf, 0x069dd0, 0x079cd0, 0x079bd1, 0x079ad1, 0x0899d1, 0x0997d2, 0x0a96d2,
      0x0b95d2, 0x0c94d2, 0x0d92d2, 0x0e91d2, 0x0f90d2, 0x108fd2, 0x108dd2, 0x118cd3,
      0x128bd3, 0x128ad3, 0x1388d3, 0x1387d3, 0x1486d4, 0x1485d4, 0x1484d4, 0x1483d5,
      0x1482d5, 0x1481d6, 0x1380d6, 0x137fd7, 0x137ed7, 0x127dd8, 0x127cd8, 0x117bd9,
      0x117ada, 0x1079da, 0x0f78db, 0x0f77db, 0x0e76dc, 0x0d75dc, 0x0c75dd, 0x0b74dd,
      0x0a73de, 0x0972de, 0x0871df, 0x0670df, 0x056ee0, 0x046de0, 0x036ce1, 0x036be1,
      0x026ae1, 0x0169e1, 0x0168e2, 0x0167e2, 0x0165e2, 0x0264e1, 0x0362e1, 0x0561e1,
      0x085fe0, 0x0b5ddf, 0x105bdd, 0x1459dc, 0x1857d9, 0x1c54d6, 0x2052d3, 0x2450d0,
      0x274ecd, 0x294cca, 0x2c4ac6, 0x2d48c3, 0x2f46c0, 0x3145bd, 0x3243b9, 0x3341b6,
      0x3440b3, 0x343eb0, 0x353cad, 0x353ba9, 0x3639a6, 0x3638a3, 0x3636a0, 0x36359d,
      0x36339a, 0x363296, 0x363093, 0x352f90, 0x352d8d, 0x352c8a, 0x352a87, 0x000000
  };

  // color maps ('magma', 'inferno', 'plasma', 'viridis') from matplotlib in Python
  // https://bids.github.io/colormap/
  // See file util/cm_list.py for generating these color maps.

  static final int[] magma = {
        0xfcfdbf, 0xfcfbbd, 0xfcfabc, 0xfcf8ba, 0xfdf6b8, 0xfdf4b6, 0xfdf2b4, 0xfdf1b2,
        0xfdefb0, 0xfdedae, 0xfdebac, 0xfde9ab, 0xfde7a9, 0xfde6a7, 0xfde4a5, 0xfee2a3,
        0xfee0a2, 0xfedea0, 0xfedc9e, 0xfedb9c, 0xfed99b, 0xfed799, 0xfed597, 0xfed395,
        0xfed194, 0xfed092, 0xfece90, 0xffcc8f, 0xffca8d, 0xffc88c, 0xffc68a, 0xffc588,
        0xffc387, 0xffc185, 0xffbf84, 0xffbd82, 0xffbb81, 0xffb97f, 0xffb87e, 0xffb67c,
        0xffb47b, 0xffb27a, 0xffb078, 0xffae77, 0xffac76, 0xffab74, 0xffa973, 0xfea772,
        0xfea570, 0xfea36f, 0xfea16e, 0xfe9f6d, 0xfe9d6c, 0xfe9c6b, 0xfe9a6a, 0xfe9869,
        0xfd9668, 0xfd9467, 0xfd9266, 0xfd9065, 0xfd8e64, 0xfd8c63, 0xfc8b62, 0xfc8961,
        0xfc8761, 0xfc8560, 0xfb835f, 0xfb815f, 0xfb7f5e, 0xfa7d5e, 0xfa7b5d, 0xfa795d,
        0xf9785c, 0xf9765c, 0xf8745c, 0xf8725c, 0xf7705c, 0xf76e5c, 0xf66c5c, 0xf56b5c,
        0xf5695c, 0xf4675c, 0xf3655c, 0xf3645c, 0xf2625d, 0xf1605d, 0xf05f5d, 0xef5d5e,
        0xee5b5e, 0xee5a5f, 0xed585f, 0xeb5760, 0xea5561, 0xe95461, 0xe85362, 0xe75163,
        0xe65064, 0xe54f64, 0xe34e65, 0xe24c66, 0xe14b67, 0xdf4a67, 0xde4968, 0xdd4869,
        0xdb476a, 0xda466b, 0xd8456b, 0xd7446c, 0xd5436d, 0xd4436e, 0xd2426e, 0xd1416f,
        0xcf4070, 0xce3f71, 0xcc3f71, 0xca3e72, 0xc93d73, 0xc73d73, 0xc63c74, 0xc43b75,
        0xc23b75, 0xc13a76, 0xbf3976, 0xbd3977, 0xbc3878, 0xba3878, 0xb93779, 0xb73779,
        0xb5367a, 0xb4357a, 0xb2357b, 0xb0347b, 0xaf347b, 0xad337c, 0xab337c, 0xaa327d,
        0xa8327d, 0xa6317d, 0xa5317e, 0xa3307e, 0xa1307e, 0xa02f7f, 0x9e2f7f, 0x9d2e7f,
        0x9b2e7f, 0x992d80, 0x982c80, 0x962c80, 0x942b80, 0x932b80, 0x912a81, 0x902a81,
        0x8e2981, 0x8c2981, 0x8b2881, 0x892881, 0x882781, 0x862781, 0x842681,
0x832581,
        0x812581, 0x802482, 0x7e2482, 0x7c2382, 0x7b2382, 0x792282, 0x782181,
0x762181,
        0x752081, 0x732081, 0x711f81, 0x701e81, 0x6e1e81, 0x6d1d81, 0x6b1c81, 0x6a1c81,
        0x681b81, 0x671a80, 0x651a80, 0x631980, 0x621980, 0x601880, 0x5f177f, 0x5d177f,
        0x5c167f, 0x5a157e, 0x58157e, 0x57147e, 0x55147d, 0x54137d, 0x52127c, 0x51127c,
        0x4f117b, 0x4d117b, 0x4c107a, 0x4a1079, 0x481078, 0x470f78, 0x450f77, 0x430f76,
        0x420f75, 0x400f74, 0x3e0f72, 0x3d0f71, 0x3b0f70, 0x390f6e, 0x370f6c, 0x360f6b,
        0x341069, 0x321067, 0x311065, 0x2f1063, 0x2d1161, 0x2b115e, 0x2a115c, 0x28115a,
        0x271157, 0x251155, 0x241153, 0x221150, 0x21114e, 0x1f114b, 0x1e1149, 0x1c1046,
        0x1b1044, 0x1a1042, 0x190f3f, 0x170f3d, 0x160e3a, 0x150e38, 0x140d36, 0x130d33,
        0x110c31, 0x100c2f, 0x0f0b2d, 0x0e0a2a, 0x0d0a28, 0x0c0926, 0x0b0824, 0x0a0722,
        0x090720, 0x08061d, 0x07051b, 0x060519, 0x050417, 0x040415, 0x040313,
0x030311,
        0x02020f, 0x02020d, 0x01010b, 0x010109, 0x010007, 0x000006, 0x000004, 0x000000
  };

  // PS: Adobe Audition-like color map
  static final int[] inferno = {
        0xfdffa5, 0xfbfea1, 0xf9fd9d, 0xf8fc9a, 0xf7fb96, 0xf6f992, 0xf5f88e, 0xf4f78a,
        0xf3f586, 0xf3f482, 0xf2f37d, 0xf2f179, 0xf2ef75, 0xf2ee71, 0xf2ec6d, 0xf2ea69,
        0xf2e965, 0xf3e761, 0xf3e55d, 0xf4e359, 0xf4e156, 0xf4df52, 0xf5dd4f, 0xf5db4c,
        0xf6d949, 0xf6d845, 0xf7d642, 0xf7d43f, 0xf8d23d, 0xf8d03a, 0xf9ce37, 0xf9cc34,
        0xf9ca32, 0xfac82f, 0xfac62d, 0xfac42a, 0xfbc228, 0xfbc025, 0xfbbe23, 0xfbbc21,
        0xfcba1e, 0xfcb81c, 0xfcb61a, 0xfcb418, 0xfcb216, 0xfcb014, 0xfcae12, 0xfcac10,
        0xfcaa0e, 0xfca90d, 0xfca70b, 0xfca50a, 0xfca308, 0xfca107, 0xfc9f07, 0xfc9d06,
        0xfc9b06, 0xfc9906, 0xfb9806, 0xfb9606, 0xfb9406, 0xfb9207, 0xfa9008, 0xfa8e08,
        0xfa8d09, 0xf98b0b, 0xf9890c, 0xf8870d, 0xf8850e, 0xf88410, 0xf78211, 0xf78012,
        0xf67e14, 0xf67d15, 0xf57b16, 0xf47918, 0xf47719, 0xf3761a, 0xf3741c, 0xf2721d,
        0xf1711e, 0xf16f20, 0xf06e21, 0xef6c22, 0xee6a23, 0xed6925, 0xed6726, 0xec6627,
        0xeb6428, 0xea632a, 0xe9612b, 0xe8602c, 0xe75e2d, 0xe65d2f, 0xe55b30, 0xe45a31,
        0xe35932, 0xe25733, 0xe15635, 0xe05536, 0xdf5337, 0xde5238, 0xdd5139, 0xdc4f3b,
        0xdb4e3c, 0xd94d3d, 0xd84c3e, 0xd74a3f, 0xd64940, 0xd54841, 0xd34743, 0xd24644,
        0xd14545, 0xcf4446, 0xce4347, 0xcd4248, 0xcb4149, 0xca404a, 0xc93f4b, 0xc73e4c,
        0xc63d4d, 0xc43c4e, 0xc33b4f, 0xc23a50, 0xc03951, 0xbf3852, 0xbd3853, 0xbc3754,
        0xba3655, 0xb93556, 0xb73457, 0xb63458, 0xb43359, 0xb33259, 0xb1315a,
0xb0315b,
        0xae305c, 0xad2f5d, 0xab2f5d, 0xaa2e5e, 0xa82d5f, 0xa62d60, 0xa52c60, 0xa32b61,
        0xa22b62, 0xa02a62, 0x9f2a63, 0x9d2964, 0x9b2864, 0x9a2865, 0x982765, 0x972766,
        0x952667, 0x942567, 0x922568, 0x902468, 0x8f2468, 0x8d2369, 0x8c2369, 0x8a226a,
        0x88216a, 0x87216b, 0x85206b, 0x84206b, 0x821f6c, 0x801f6c, 0x7f1e6c, 0x7d1d6c,
        0x7c1d6d, 0x7a1c6d, 0x781c6d, 0x771b6d, 0x751b6e, 0x741a6e, 0x721a6e, 0x70196e,
        0x6f186e, 0x6d186e, 0x6c176e, 0x6a176e, 0x68166e, 0x67156e, 0x65156e, 0x64146e,
        0x62146e, 0x61136e, 0x5f126e, 0x5d126e, 0x5c116e, 0x5a116e, 0x59106e, 0x570f6d,
        0x550f6d, 0x540e6d, 0x520e6d, 0x500d6c, 0x4f0d6c, 0x4d0c6c, 0x4c0c6b, 0x4a0b6b,
        0x480b6a, 0x470a6a, 0x450a69, 0x430a68, 0x420968, 0x400967, 0x3e0966,
0x3c0965,
        0x3b0964, 0x390963, 0x370962, 0x360960, 0x34095f, 0x32095e, 0x300a5c, 0x2e0a5a,
        0x2d0a59, 0x2b0a57, 0x290b55, 0x270b53, 0x260b51, 0x240b4e, 0x220b4c, 0x210c4a,
      0x1f0c48, 0x1d0c45, 0x1c0c43, 0x1a0c40, 0x190b3e, 0x170b3b, 0x160b39, 0x140b37,
      0x130a34, 0x120a32, 0x10092f, 0x0f092d, 0x0e082b, 0x0d0828, 0x0c0726, 0x0a0724,
      0x090621, 0x08061f, 0x07051d, 0x06041b, 0x050418, 0x040316, 0x040314, 0x030212,
      0x020210, 0x02010e, 0x01010b, 0x010109, 0x010007, 0x000006, 0x000004, 0x000000
  };

  static final int[] plasma = {
      0xf0f921, 0xf1f724, 0xf1f625, 0xf2f426, 0xf2f226, 0xf3f127, 0xf4ef27, 0xf4ed27,
      0xf5eb26, 0xf6ea26, 0xf6e826, 0xf7e625, 0xf7e425, 0xf8e325, 0xf8e125, 0xf9df24,
      0xf9de24, 0xfadc24, 0xfada24, 0xfbd924, 0xfbd724, 0xfbd524, 0xfcd424, 0xfcd224,
      0xfcd024, 0xfdcf25, 0xfdcd25, 0xfdcb25, 0xfdca26, 0xfdc826, 0xfec727, 0xfec527,
      0xfec328, 0xfec228, 0xfec029, 0xfebf29, 0xfebd2a, 0xfebc2b, 0xfeba2b, 0xfeb92c,
      0xfeb72d, 0xfeb52e, 0xfeb42e, 0xfeb22f, 0xfeb130, 0xfeaf31, 0xfdae31, 0xfdac32,
      0xfdab33, 0xfda934, 0xfda835, 0xfca636, 0xfca536, 0xfca437, 0xfca238, 0xfba139,
      0xfb9f3a, 0xfb9e3b, 0xfb9c3c, 0xfa9b3c, 0xfa9a3d, 0xf9983e, 0xf9973f, 0xf99540,
      0xf89441, 0xf89342, 0xf79142, 0xf79043, 0xf78f44, 0xf68d45, 0xf68c46, 0xf58b47,
      0xf58948, 0xf48849, 0xf48749, 0xf3854a, 0xf2844b, 0xf2834c, 0xf1814d, 0xf1804e,
      0xf07f4f, 0xf07e50, 0xef7c50, 0xee7b51, 0xee7a52, 0xed7953, 0xed7754, 0xec7655,
      0xeb7556, 0xeb7456, 0xea7257, 0xe97158, 0xe87059, 0xe86f5a, 0xe76e5b, 0xe66c5c,
      0xe66b5c, 0xe56a5d, 0xe4695e, 0xe3685f, 0xe36660, 0xe26561, 0xe16462, 0xe06362,
      0xe06263, 0xdf6064, 0xde5f65, 0xdd5e66, 0xdc5d67, 0xdc5c68, 0xdb5b69, 0xda5969,
      0xd9586a, 0xd8576b, 0xd7566c, 0xd7556d, 0xd6546e, 0xd5536f, 0xd45170, 0xd35070,
      0xd24f71, 0xd14e72, 0xd04d73, 0xd04c74, 0xcf4b75, 0xce4976, 0xcd4877, 0xcc4778,
      0xcb4679, 0xca457a, 0xc9447a, 0xc8437b, 0xc7427c, 0xc6407d, 0xc53f7e, 0xc43e7f,
      0xc33d80, 0xc23c81, 0xc13b82, 0xc03a83, 0xbf3984, 0xbe3785, 0xbd3686, 0xbc3587,
      0xbb3488, 0xba3389, 0xb93289, 0xb8318a, 0xb72f8b, 0xb52e8c, 0xb42d8d, 0xb32c8e,
      0xb22b8f, 0xb12a90, 0xb02991, 0xaf2892, 0xad2693, 0xac2594, 0xab2495, 0xaa2395,
      0xa92296, 0xa72197, 0xa62098, 0xa51e99, 0xa41d9a, 0xa21c9a, 0xa11b9b, 0xa01a9c,
      0x9e199d, 0x9d189d, 0x9c179e, 0x9a159f, 0x9914a0, 0x9813a0, 0x9612a1, 0x9511a1,
      0x9410a2, 0x920fa3, 0x910ea3, 0x900da4, 0x8e0ca4, 0x8d0ba5, 0x8b09a5, 0x8a08a6,
      0x8807a6, 0x8707a6, 0x8606a7, 0x8405a7, 0x8304a7, 0x8104a7, 0x8003a8,
0x7e03a8,
      0x7d02a8, 0x7b02a8, 0x7a01a8, 0x7801a8, 0x7701a8, 0x7500a9, 0x7400a9,
0x7200a9,
      0x7100a8, 0x6f00a8, 0x6e00a8, 0x6c00a8, 0x6a00a8, 0x6900a8, 0x6700a8, 0x6600a7,
      0x6400a7, 0x6300a7, 0x6100a7, 0x5f00a6, 0x5e00a6, 0x5c00a6, 0x5b00a5, 0x5901a5,
      0x5801a5, 0x5601a4, 0x5401a4, 0x5301a3, 0x5101a3, 0x5002a2, 0x4e02a2,
0x4c02a1,
      0x4b02a1, 0x4902a0, 0x4702a0, 0x46039f, 0x44039e, 0x42039e, 0x41039d, 0x3f039c,
      0x3d039c, 0x3c039b, 0x3a049a, 0x38049a, 0x360499, 0x350498, 0x330497, 0x310497,
      0x2f0496, 0x2d0495, 0x2b0594, 0x2a0593, 0x280592, 0x260592, 0x230591, 0x210590,
      0x1f058f, 0x1d068e, 0x1b068d, 0x18068c, 0x16068a, 0x130689, 0x100788, 0x000000
  };
```

```
    static final int[] viridis = {
        0xfee724, 0xfbe723, 0xf9e721, 0xf7e61f, 0xf4e61e, 0xf2e61c, 0xefe61b, 0xede51a,
        0xeae519, 0xe8e519, 0xe5e418, 0xe2e418, 0xe0e418, 0xdde318, 0xdbe318,
0xd8e319,
        0xd5e21a, 0xd3e21b, 0xd0e21c, 0xcde11d, 0xcbe11e, 0xc8e120, 0xc5e021, 0xc3e023,
        0xc0df24, 0xbddf26, 0xbbdf27, 0xb8de29, 0xb5de2b, 0xb3dd2d, 0xb0dd2e, 0xaddd30,
        0xabdc32, 0xa8dc33, 0xa5db35, 0xa3db37, 0xa0da39, 0x9dda3a, 0x9bd93c, 0x98d93e,
        0x95d83f, 0x93d841, 0x90d743, 0x8ed744, 0x8bd646, 0x89d548, 0x86d549, 0x84d44b,
        0x81d44c, 0x7fd34e, 0x7cd24f, 0x7ad251, 0x77d152, 0x75d054, 0x73d055, 0x70cf57,
        0x6ece58, 0x6cce5a, 0x69cd5b, 0x67cc5c, 0x65cc5e, 0x62cb5f, 0x60ca60, 0x5ec962,
        0x5cc963, 0x5ac864, 0x58c765, 0x55c666, 0x53c668, 0x51c569, 0x4fc46a, 0x4dc36b,
        0x4bc26c, 0x49c26d, 0x48c16e, 0x46c06f, 0x44bf70, 0x42be71, 0x40bd72, 0x3ebd73,
        0x3dbc74, 0x3bbb75, 0x39ba76, 0x38b977, 0x36b878, 0x35b779, 0x33b779,
0x32b67a,
        0x30b57b, 0x2fb47c, 0x2eb37c, 0x2cb27d, 0x2bb17e, 0x2ab07f, 0x29af7f, 0x28af80,
        0x27ae81, 0x26ad81, 0x25ac82, 0x24ab82, 0x23aa83, 0x23a983, 0x22a884,
0x21a785,
        0x21a685, 0x20a586, 0x20a486, 0x1fa386, 0x1fa387, 0x1fa287, 0x1fa188, 0x1ea088,
        0x1e9f88, 0x1e9e89, 0x1e9d89, 0x1e9c89, 0x1e9b8a, 0x1e9a8a, 0x1e998a, 0x1e988b,
        0x1f978b, 0x1f968b, 0x1f958b, 0x1f948c, 0x1f938c, 0x20938c, 0x20928c, 0x20918c,
        0x20908d, 0x218f8d, 0x218e8d, 0x218d8d, 0x228c8d, 0x228b8d, 0x228a8d, 0x23898e,
        0x23888e, 0x23878e, 0x24868e, 0x24858e, 0x25848e, 0x25838e, 0x25828e,
0x26828e,
        0x26818e, 0x26808e, 0x277f8e, 0x277e8e, 0x287d8e, 0x287c8e, 0x287b8e, 0x297a8e,
        0x29798e, 0x29788e, 0x2a778e, 0x2a768e, 0x2b758e, 0x2b748e, 0x2b738e,
0x2c728e,
        0x2c718e, 0x2d708e, 0x2d708e, 0x2d6f8e, 0x2e6e8e, 0x2e6d8e, 0x2f6c8e, 0x2f6b8e,
        0x2f6a8e, 0x30698e, 0x30688e, 0x31678e, 0x31668e, 0x32658e, 0x32648e, 0x33638d,
        0x33628d, 0x33618d, 0x34608d, 0x345f8d, 0x355e8d, 0x355d8d, 0x365c8d, 0x365b8d,
        0x37598c, 0x37588c, 0x38578c, 0x38568c, 0x39558c, 0x39548c, 0x3a538b, 0x3a528b,
        0x3b518b, 0x3b508b, 0x3c4f8a, 0x3c4e8a, 0x3d4d8a, 0x3d4b8a, 0x3e4a89, 0x3e4989,
        0x3f4889, 0x3f4788, 0x404688, 0x404588, 0x414387, 0x414287, 0x414186, 0x424086,
        0x423f85, 0x433e85, 0x433c84, 0x433b84, 0x443a83, 0x443983, 0x443882, 0x453681,
        0x453581, 0x453480, 0x46337f, 0x46317e, 0x46307e, 0x462f7d, 0x472e7c, 0x472c7b,
        0x472b7a, 0x472a79, 0x472979, 0x482778, 0x482677, 0x482576, 0x482475,
0x482274,
        0x482173, 0x482071, 0x481e70, 0x481d6f, 0x481c6e, 0x481a6d, 0x48196c, 0x48186a,
        0x481669, 0x481568, 0x471466, 0x471265, 0x471164, 0x470f62, 0x470e61, 0x460c60,
        0x460b5e, 0x46095d, 0x46085b, 0x45065a, 0x450558, 0x450357, 0x440255, 0x000000
    };
}
```

FPSCounter

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
```

```
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.os.SystemClock;
import android.util.Log;

// Frames Per Second Counter

class FPSCounter {
  private long frameCount;
  private long timeOld, timeUpdateInterval;  // in ms
  private double fps;
  private String TAG_OUTSIDE;

  FPSCounter(String TAG) {
    timeUpdateInterval = 2000;
    TAG_OUTSIDE = TAG;
    frameCount = 0;
    timeOld = SystemClock.uptimeMillis();
  }

  // call this when number of frames plus one
  void inc() {
    frameCount++;
    long timeNow = SystemClock.uptimeMillis();
    if (timeOld + timeUpdateInterval <= timeNow) {
      fps = 1000 * (double) frameCount / (timeNow - timeOld);
      Log.d(TAG_OUTSIDE, "FPS: " + Math.round(100*fps)/100.0 +
            " (" + frameCount + "/" + (timeNow - timeOld) + "ms)");
      timeOld = timeNow;
      frameCount = 0;
    }
  }

  public double getFPS() {
    return fps;
  }
}
```

GridLabel

```
/* Copyright 2017 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.util.Log;

import static java.lang.Math.abs;
import static java.lang.Math.ceil;
import static java.lang.Math.exp;
import static java.lang.Math.floor;
import static java.lang.Math.log;
import static java.lang.Math.log10;
import static java.lang.Math.pow;
import static java.lang.Math.round;
import static java.lang.Math.sqrt;

/**
 * Generate grid label (marker)
 */

class GridLabel {
    private final static String TAG = "GridLabel:";
    // Never null!
    double[] values = new double[0];  // TODO: use a better name?
    double[] ticks  = new double[0];  // TODO: use a better name?

    enum Type {  // java's enum type is inconvenient
        FREQ(0), DB(1), TIME(2), FREQ_LOG(3), FREQ_NOTE(4);

        private final int value;
        private Type(int value) { this.value = value; }
        public int getValue() { return value; }
    }

    private Type gridType;
    private double gridDensity;

    GridLabel(Type _gridType, double _gridDensity) {
        gridType = _gridType;
        gridDensity = _gridDensity;
    }

    Type getGridType() { return gridType; }

    void setGridType(GridLabel.Type gt) { gridType = gt; }
    void setDensity(double _gridDensity) { gridDensity = _gridDensity; }

    // return position of grid lines, there are roughly gridDensity lines for the bigger grid
    private static int genLinearGridPoints(double[][] gridPointsArray, double startValue, double
endValue,
                                            double gridDensity, Type scale_mode) {
        if (Double.isInfinite(startValue+endValue) || Double.isNaN(startValue+endValue)) {
            Log.e(TAG, "genLinearGridPoints(): value invalid");
            return 0;
        }
        if (startValue == endValue) {
            Log.e(TAG, "genLinearGridPoints(): startValue == endValue");
            return 0;
        }
        if (startValue > endValue) {
            double t = endValue;
            endValue = startValue;
            startValue = t;
        }
        if (scale_mode == Type.FREQ || scale_mode == Type.TIME) {
            if (gridDensity < 3.2) {
                // 3.2 >= 2 * 5/sqrt(2*5), so that there are at least 2 bigger grid.
                // The constant here is because: if gridIntervalGuess = sqrt(2*5), then gridIntervalBig
= 5
                // i.e. grid size become larger by factor 5/sqrt(2*5).
                // By setting gridDensity = 3.2, we can make sure minimum gridDensity > 2
                gridDensity = 3.2;
            }
        } else {
            if (gridDensity < 3.5) {  // similar discussion as above
                gridDensity = 3.5;      // 3.5 >= 2 * 3/sqrt(1*3)
            }
        }
        double intervalValue = endValue - startValue;
        double gridIntervalGuess = intervalValue / gridDensity;
        double gridIntervalBig;
        double gridIntervalSmall;

        // Determine a suitable grid interval from guess
        if (scale_mode == Type.FREQ || scale_mode == Type.TIME || intervalValue <= 1) {  //
Linear scale (Hz, Time)
            double exponent = pow(10, floor(log10(gridIntervalGuess)));
            double fraction = gridIntervalGuess / exponent;
            // grid interval is 1, 2, 5, 10, ...
            if (fraction < sqrt(1*2)) {
                gridIntervalBig   = 1;
                gridIntervalSmall = 0.2;
            } else if (fraction < sqrt(2*5)) {
                gridIntervalBig   = 2;
                gridIntervalSmall = 1.0;
            } else if (fraction < sqrt(5*10)) {
                gridIntervalBig   = 5;
                gridIntervalSmall = 1;
            } else {
                gridIntervalBig   = 10;
                gridIntervalSmall = 2;
            }
            gridIntervalBig   *= exponent;
            gridIntervalSmall *= exponent;
        } else {  // dB scale
            if (gridIntervalGuess > sqrt(36*12)) {
                gridIntervalBig   = 36;
                gridIntervalSmall = 12;
            } else if (gridIntervalGuess > sqrt(12*6)) {
                gridIntervalBig   = 12;
                gridIntervalSmall = 2;
            } else if (gridIntervalGuess > sqrt(6*3)) {
                gridIntervalBig   = 6;
                gridIntervalSmall = 1;
            } else if (gridIntervalGuess > sqrt(3*1)) {
                gridIntervalBig   = 3;
                gridIntervalSmall = 1;
            } else {
                gridIntervalBig   = 1;
                gridIntervalSmall = 1.0/6;
            }
        }

        if (gridPointsArray == null || gridPointsArray.length != 2) {
            Log.e(TAG, "genLinearGridPoints(): empty array!!");
            return 0;
        }

        // Reallocate if number of grid lines are different
        // Then fill in the gird line coordinates. Assuming the grid lines starting from 0
        double gridStartValueBig   = ceil(startValue / gridIntervalBig)   * gridIntervalBig;
        int nGrid = (int) floor((endValue - gridStartValueBig) / gridIntervalBig) + 1;
        if (nGrid != gridPointsArray[0].length) {
            gridPointsArray[0] = new double[nGrid];
        }
        double[] bigGridPoints = gridPointsArray[0];
        for (int i = 0; i < nGrid; i++) {
            bigGridPoints[i] = gridStartValueBig + i*gridIntervalBig;
        }

        double gridStartValueSmall = ceil(startValue / gridIntervalSmall) * gridIntervalSmall;
        nGrid = (int) floor((endValue - gridStartValueSmall) / gridIntervalSmall) + 1;
        if (nGrid != gridPointsArray[1].length) {    // reallocate space when need
            gridPointsArray[1] = new double[nGrid];
        }
        double[] smallGridPoints = gridPointsArray[1];
        for (int i = 0; i < nGrid; i++) {
            smallGridPoints[i] = gridStartValueSmall + i*gridIntervalSmall;
        }
        return (int)floor(log10(gridIntervalBig));
    }

    // TODO: might be merge to genLinearGridPoints.
    private static int genLogarithmicGridPoints(double[][] gridPointsArray, double startValue,
double endValue,
                                               double gridDensity) {
        if (Double.isInfinite(startValue + endValue) || Double.isNaN(startValue + endValue)) {
            Log.e(TAG, "genLogarithmicGridPoints(): value invalid");
            return 0;
        }
        if (startValue == endValue) {
            Log.e(TAG, "genLogarithmicGridPoints(): startValue == endValue");
            return 0;
        }
        if (startValue <=0 || endValue <= 0) {
            Log.e(TAG, "genLogarithmicGridPoints(): startValue <=0 || endValue <= 0 !!");
            return 0;
        }
        if (startValue > endValue) {
            double t = endValue;
            endValue = startValue;
            startValue = t;
        }

        int cntTick = 0;
        int cntVal = 0;
        if (endValue / startValue > 100) {
            // Major:  1, 10, 100, ...
            // Minor:  1, 2, 3, ... , 9, 10, 20, 30, ...
            double gapChangingPoint = pow(10, floor(log10(startValue)));
            double gridIntervalSmall = ceil(startValue / gapChangingPoint) * gapChangingPoint;
//            Log.i(TAG, "startValue = " + startValue + "  gapChangingPoint = " + gapChangingPoint
+ "  gridIntervalSmall = " + gridIntervalSmall);

            double b1 = pow(10, ceil(log10(startValue)));
            double b2 = pow(10, floor(log10(endValue)));
            int nGridBig   = (int)(floor(log10(endValue)) - ceil(log10(startValue)) + 1);
            int nGridSmall = (int)(floor((b1 - startValue) / gapChangingPoint)
                    + 9 * (nGridBig - 1)
                    + floor((endValue - b2) / b2)) + 1;
            if (nGridBig != gridPointsArray[0].length) {
                gridPointsArray[0] = new double[nGridBig];
            }
            if (nGridSmall != gridPointsArray[1].length) {
                gridPointsArray[1] = new double[nGridSmall];
            }
            while (gapChangingPoint <= endValue) {
```

```
        while (gridIntervalSmall < 10*gapChangingPoint && gridIntervalSmall <= endValue) {
          gridPointsArray[1][cntTick++] = gridIntervalSmall;
          gridIntervalSmall += gapChangingPoint;
        }
        if (gapChangingPoint >= startValue) {
          gridPointsArray[0][cntVal++] = gapChangingPoint;
        }
        gapChangingPoint *= 10;
      }
      return Integer.MAX_VALUE;
    } else if (endValue / startValue > 10) {
      // Major:  1, 2, 3, ... , 9, 10, 20, 30, ...
      // Minor:  1, 1.5, 2, 2.5, ..., 9, 9.5, 10, 15, 20 25, ...
      double gapChangingPoint = pow(10, floor(log10(startValue)));
      double gapChangingPointd2 = gapChangingPoint / 2;
      double gridIntervalSmall = ceil(startValue / gapChangingPoint) * gapChangingPoint;
      double gridIntervalSmall2 = ceil(startValue / gapChangingPointd2) *
gapChangingPointd2;

      double b1 = pow(10, ceil(log10(startValue)));
      double b2 = pow(10, floor(log10(endValue)));
      int nGridBig   = (int)(floor(log10(endValue)) - ceil(log10(startValue)) + 1);
      int nGridSmall = (int)(floor((b1 - startValue) / gapChangingPoint)
          + 9 * (nGridBig - 1)
          + floor((endValue - b2) / b2)) + 1;
      int nGridSmall2 = (int)(floor((b1 - startValue) / gapChangingPointd2)
          + 18 * (nGridBig - 1)
          + floor((endValue - b2) / (b2/2))) + 1;
      if (nGridSmall != gridPointsArray[0].length) {
        gridPointsArray[0] = new double[nGridSmall];
      }
      if (nGridSmall2 != gridPointsArray[1].length) {
        gridPointsArray[1] = new double[nGridSmall2];
      }
      while (gapChangingPoint <= endValue) {
        while (gridIntervalSmall < 10*gapChangingPoint && gridIntervalSmall <= endValue) {
          gridPointsArray[0][cntVal++] = gridIntervalSmall;
          gridIntervalSmall += gapChangingPoint;
        }
        while (gridIntervalSmall2 < 10*gapChangingPoint && gridIntervalSmall2 <= endValue)
{
          gridPointsArray[1][cntTick++] = gridIntervalSmall2;
          gridIntervalSmall2 += gapChangingPointd2;
        }
```

```
                gapChangingPoint *= 10;
                gapChangingPointd2 = gapChangingPoint / 2;
            }
            return Integer.MAX_VALUE;
        } else {
            // Linear increment.
            // limit the largest major gap <= 1/3 screen width
            if (gridDensity < 3) {
//                Log.i(TAG, "genLogarithmicGridPoints(): low gridDensity = " + gridDensity);
                gridDensity = 3;
            }
            // reduce gridDensity when endValue/startValue is large
            gridDensity /= log(49 * endValue/startValue + 1) / log(50);
            double gridIntervalGuess = pow(endValue/startValue, 1/gridDensity);
//            Log.i(TAG, "  gridIntervalGuess = " + gridIntervalGuess + "  gridDensity = " +
gridDensity + "  s = " + startValue + "  e = " + endValue);
            gridIntervalGuess = (gridIntervalGuess-1) * startValue;
//            Log.i(TAG, "  gridIntervalGuess2 = " + gridIntervalGuess);

            double gridIntervalBig, gridIntervalSmall;
            double exponent = pow(10, floor(log10(gridIntervalGuess)));
            double fraction = gridIntervalGuess / exponent;
            // grid interval is 1, 2, 5, 10, ...
            if (fraction < sqrt(1*2)) {
                gridIntervalBig   = 1;
                gridIntervalSmall = 0.2;
            } else if (fraction < sqrt(2*5)) {
                gridIntervalBig   = 2;
                gridIntervalSmall = 1.0;
            } else if (fraction < sqrt(5*10)) {
                gridIntervalBig   = 5;
                gridIntervalSmall = 1;
            } else {
                gridIntervalBig   = 10;
                gridIntervalSmall = 2;
            }
            gridIntervalBig   *= exponent;
            gridIntervalSmall *= exponent;

            double gridStartValueBig   = ceil(startValue / gridIntervalBig)   * gridIntervalBig;
            int nGrid = (int) floor((endValue - gridStartValueBig) / gridIntervalBig) + 1;
            if (nGrid != gridPointsArray[0].length) {
                gridPointsArray[0] = new double[nGrid];
            }
            double[] bigGridPoints = gridPointsArray[0];
            for (int i = 0; i < nGrid; i++) {
                bigGridPoints[i] = gridStartValueBig + i*gridIntervalBig;
            }

            double gridStartValueSmall = ceil(startValue / gridIntervalSmall) * gridIntervalSmall;
            nGrid = (int) floor((endValue - gridStartValueSmall) / gridIntervalSmall) + 1;
            if (nGrid != gridPointsArray[1].length) {    // reallocate space when need
                gridPointsArray[1] = new double[nGrid];
            }
            double[] smallGridPoints = gridPointsArray[1];
            for (int i = 0; i < nGrid; i++) {
                smallGridPoints[i] = gridStartValueSmall + i*gridIntervalSmall;
            }
            return (int)floor(log10(gridIntervalBig));
        }
    }

    private static double[] majorPitch = new double[]{0, 2, 4, 5, 7, 9, 11};

    // majorPitchCount[ceil(p)] is the number of notes should show (p is right boundary)
    // 7 - majorPitchCount[ceil(p)] for left boundary
    private static int[] majorPitchCount = new int[]{0, 1, 1, 2, 2, 3, 4, 4, 5, 5, 6, 6, 7, 8};
    private static int[] isMajorPitch = new int[]{1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1};

    private static double mod(double v, double m) { return v - floor(v/m)*m; };

    private static int genMusicNoteGridPoints(double[][] gridPointsArray, double startValue,
double endValue,
                                                double gridDensity, int start_note) {
        // For Type.FREQ_NOTE
        if (Double.isInfinite(startValue+endValue) || Double.isNaN(startValue+endValue)) {
            Log.e(TAG, "genLinearGridPoints(): value invalid");
            return 0;
        }
        if (startValue == endValue) {
            Log.e(TAG, "genLinearGridPoints(): startValue == endValue");
            return 0;
        }
        if (startValue > endValue) {
            double t = endValue;
            endValue = startValue;
            startValue = t;
        }
```

```
startValue = AnalyzerUtil.freq2pitch(startValue);
endValue   = AnalyzerUtil.freq2pitch(endValue);

double intervalValue = endValue - startValue;
double gridIntervalGuess = intervalValue / gridDensity;
double gridIntervalBig = 0;
double gridIntervalSmall = 0;

boolean skipGridCal = false;

if (gridIntervalGuess > 5) {
    gridIntervalBig = 12;
    gridIntervalSmall = 1;
} else if (gridIntervalGuess > 1.2) {
    gridIntervalBig = 1;
    gridIntervalSmall = 0.5;
} else {
    genLinearGridPoints(gridPointsArray, startValue, endValue, gridDensity, Type.FREQ);
    skipGridCal = true;
}

if (!skipGridCal) {
    int nGrid = 0;
    // start_note == 0 <=> C
    for (int k = 0; k < 2; k++) {
        double gridInterval = k == 0 ? gridIntervalBig : gridIntervalSmall;
        double[] val = gridPointsArray[k];
        if (gridInterval == 1) {
            startValue += start_note;
            endValue += start_note;
            int startOctave = (int) floor(startValue / 12);
            int endOctave   = (int) floor(endValue   / 12);
            nGrid = 7 - majorPitchCount[(int) ceil(startValue - startOctave * 12)]
                    + 7 * (endOctave - startOctave - 1)
                    + majorPitchCount[(int) ceil(endValue - endOctave * 12)];
            if (nGrid != val.length) {
                val = gridPointsArray[k] = new double[nGrid];
            }
            int v = (int) ceil(startValue);
            int cnt = 0;
            while (v < endValue) {
                if (isMajorPitch[(int)mod(v, 12)] == 1) {
                    val[cnt++] = v;
```

```
                        }
                        v++;
                    }
                    startValue -= start_note;
                    endValue -= start_note;
                    for (int i = 0; i < val.length; i++) {
                        val[i] -= start_note;
                    }
                } else {
                    // equal interval
                    double gridStartValue = ceil(startValue / gridInterval) * gridInterval;
                    nGrid = (int) floor((endValue - gridStartValue) / gridInterval) + 1;
                    if (nGrid != val.length) {
                        val = gridPointsArray[k] = new double[nGrid];
                    }
                    for (int i = 0; i < nGrid; i++) {
                        val[i] = gridStartValue + i * gridInterval;
                    }
                }
            }
        }

//        Log.i(TAG, "Note: " + startPitch + "(" + startValue + ") ~ " + endPitch + " (" + endValue + ")
div " + gridPointsArray[0].length);
        for (int k = 0; k < 2; k++) {
            for (int i = 0; i < gridPointsArray[k].length; i++) {
                gridPointsArray[k][i] = AnalyzerUtil.pitch2freq(gridPointsArray[k][i]);
            }
        }
        return 0;
    }

    StringBuilder[] strings = new StringBuilder[0];
    char[][] chars = new char[0][];

    private double[] oldGridBoundary = new double[2];
    private double[][] gridPointsArray = new double[2][];

    // It's so ugly to write these StringBuffer stuff -- in order to reduce garbage
    // Also, since there is no "pass by reference", modify array is also ugly...
    void updateGridLabels(double startValue, double endValue) {
        gridPointsArray[0] = values;
        gridPointsArray[1] = ticks;
        int gapPrecision = 0;
    switch (gridType) {
      case FREQ_LOG:
        gapPrecision = genLogarithmicGridPoints(gridPointsArray, startValue, endValue,
gridDensity);
        break;
      case FREQ_NOTE:
        gapPrecision = genMusicNoteGridPoints(gridPointsArray, startValue, endValue,
gridDensity, 0);
        break;
      default:
        gapPrecision = genLinearGridPoints(gridPointsArray, startValue, endValue,
gridDensity, gridType);
    }
    values = gridPointsArray[0];
    ticks  = gridPointsArray[1];
    boolean needUpdate = false;
    if (values.length != strings.length) {
      strings = new StringBuilder[values.length];
      for (int i = 0; i < values.length; i++) {
        strings[i] = new StringBuilder();
      }
      chars = new char[values.length][];
      for (int i = 0; i < values.length; i++) {
        chars[i] = new char[16];    // hand coded max char length...
      }
      needUpdate = true;
    }
    if (values.length > 0 && (needUpdate || values[0] != oldGridBoundary[0]
        || values[values.length-1] != oldGridBoundary[1])) {
      oldGridBoundary[0] = values[0];
      oldGridBoundary[1] = values[values.length-1];
      for (int i = 0; i < strings.length; i++) {
        strings[i].setLength(0);
        if (gridType == Type.FREQ_NOTE) {
          double p = AnalyzerUtil.freq2pitch(values[i]);
          AnalyzerUtil.pitch2Note(strings[i], p, gapPrecision, true);
        } else {
          if (gapPrecision == Integer.MAX_VALUE) {  // 1000, 10000 -> 1k, 10k
            if (values[i] >= 1000) {
              SBNumFormat.fillInNumFixedFrac(strings[i], values[i] / 1000, 7, 0);
              strings[i].append('k');
            } else {
              SBNumFormat.fillInNumFixedFrac(strings[i], values[i], 7, 0);
            }
```

```
                    } else if (gapPrecision >= 3) {  // use 1k 2k ...
                        SBNumFormat.fillInNumFixedFrac(strings[i], values[i] / 1000, 7, 0);
                        strings[i].append('k');
                    } else if (gapPrecision >= 0) {
                        SBNumFormat.fillInNumFixedFrac(strings[i], values[i], 7, 0);
                    } else {
                        SBNumFormat.fillInNumFixedFrac(strings[i], values[i], 7, -gapPrecision);
                    }
                }
                strings[i].getChars(0, strings[i].length(), chars[i], 0);
            }
        }
    }

    boolean isImportantLabel(int j) {
        // For freq, time
        if (gridType == Type.FREQ_NOTE) {
            // assume C major
            return AnalyzerUtil.isAlmostInteger(AnalyzerUtil.freq2pitch(values[j])/12.0);
        } else {
            return AnalyzerUtil.isAlmostInteger(log10(values[j]));
        }
    }
}
```

InfoRecActivity

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.annotation.TargetApi;
import android.app.Activity;
import android.media.AudioFormat;
import android.media.AudioRecord;
import android.os.Build;
import android.os.Bundle;
import android.support.v4.app.NavUtils;
import android.text.method.ScrollingMovementMethod;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

import java.util.ArrayList;

// Test all (including unknown) recorder sources by open it and read data.

public class InfoRecActivity extends Activity {
    private AnalyzerUtil analyzerUtil;
    private CharSequence testResultSt = null;

    private volatile boolean bShouldStop = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_info_rec);
        // Show the Up button in the action bar.
        setupActionBar();

        analyzerUtil = new AnalyzerUtil(this);
        testResultSt = null;

        final TextView tv = (TextView) findViewById(R.id.info_rec_tv);
        tv.setMovementMethod(new ScrollingMovementMethod());
    }

    /**
     * Set up the {@link android.app.ActionBar}, if the API is available.
     */
    @TargetApi(Build.VERSION_CODES.HONEYCOMB)
    private void setupActionBar() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
            if (getActionBar() != null)
                getActionBar().setDisplayHomeAsUpEnabled(true);
        }
```

```
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		// Inflate the menu; this adds items to the action bar if it is present.
		getMenuInflater().inflate(R.menu.info_rec, menu);
		return true;
	}

	@Override
	public boolean onOptionsItemSelected(MenuItem item) {
		final TextView tv = (TextView) findViewById(R.id.info_rec_tv);
		switch (item.getItemId()) {
			case android.R.id.home:
				// This ID represents the Home or Up button. In the case of this
				// activity, the Up button is shown. Use NavUtils to allow users
				// to navigate up one level in the application structure. For
				// more details, see the Navigation pattern on Android Design:
				//
				//
http://developer.android.com/design/patterns/navigation.html#up-vs-back
				//
				NavUtils.navigateUpFromSameTask(this);
				return true;
			case R.id.rec_tester_std:
				bShouldStop = true;
				runTest(tv, 1);
				break;
			case R.id.rec_tester_support:
				bShouldStop = true;
				runTest(tv, 7);
				break;
			case R.id.rec_tester_all:
				bShouldStop = true;
				runTest(tv, 0);
				break;
		}
		return super.onOptionsItemSelected(item);
	}

	@Override
	protected void onResume() {
		super.onResume();
		final TextView tv = (TextView) findViewById(R.id.info_rec_tv);
```

```
        if (testResultSt != null) {
            tv.setText(testResultSt);
            return;
        }

        runTest(tv, 7);
    }

    private Thread testerThread;

    private void runTest(final TextView tv, final int testLevel) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                if (testerThread != null) {
                    try {
                        testerThread.join();
                    } catch (InterruptedException e) {
                        // ???
                    }
                }
                testResultSt = null;
                setTextData(tv, getString(R.string.rec_tester_hint1));
                bShouldStop = false;
                testerThread = new Thread(new Runnable() {
                    @Override
                    public void run() {
                        TestAudioRecorder(tv, testLevel);
                    }
                });
                testerThread.start();
            }
        }).start();
    }

    private void setTextData(final TextView tv, final String st) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                tv.setText(st);
            }
        });
    }
```

```
            }

    private void appendTextData(final TextView tv, final String st) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                tv.append(st);
            }
        });
    }

    // Show supported sample rate and corresponding minimum buffer size.
    private void TestAudioRecorder(final TextView tv, int testLevel) {
        // All possible sample rate
        String[] sampleRates =
getResources().getStringArray(R.array.std_sampling_rates);
        String st = getString(R.string.rec_tester_col1);

        ArrayList<String> resultMinBuffer = new ArrayList<>();
        for (String sr : sampleRates) {
            int rate = Integer.parseInt(sr.trim());
            int minBufSize = AudioRecord
                    .getMinBufferSize(rate,
AudioFormat.CHANNEL_IN_MONO,
                            AudioFormat.ENCODING_PCM_16BIT);
            if (minBufSize != AudioRecord.ERROR_BAD_VALUE) {
                resultMinBuffer.add(sr);
                // /2.0 due to ENCODING_PCM_16BIT, CHANNEL_IN_MONO
                st += getString(R.string.rec_tester_col2, rate, minBufSize,
1000.0*minBufSize/2.0/rate);
            } else {
                st += sr + getString(R.string.rec_tester_error1);
            }
        }
        sampleRates = resultMinBuffer.toArray(new String[0]);

        appendTextData(tv, st);

        // Get audio source list
        int[] audioSourceId = analyzerUtil.GetAllAudioSource(testLevel);
        ArrayList<String> audioSourceStringList = new ArrayList<>();
        for (int id : audioSourceId) {
            audioSourceStringList.add(analyzerUtil.getAudioSourceName(id));
        }
        String[] audioSourceString = audioSourceStringList.toArray(new String[0]);
        appendTextData(tv, getString(R.string.rec_tester_hint2));
        for (int ias = 0; ias < audioSourceId.length; ias++) {
            if (bShouldStop) return;
            st = getString(R.string.rec_tester_col3, audioSourceString[ias]);
            appendTextData(tv, st);
            for (String sr : sampleRates) {
                if (bShouldStop) return;
                int sampleRate = Integer.parseInt(sr.trim());
                int recBufferSize = AudioRecord.getMinBufferSize(sampleRate,
                        AudioFormat.CHANNEL_IN_MONO,
AudioFormat.ENCODING_PCM_16BIT);
                st = getString(R.string.rec_tester_col3_row1, sampleRate);

                // wait for AudioRecord fully released...
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                AudioRecord record;
                try {
                    record = new AudioRecord(audioSourceId[ias],
sampleRate,
                            AudioFormat.CHANNEL_IN_MONO,
AudioFormat.ENCODING_PCM_16BIT, recBufferSize);
                } catch (IllegalArgumentException e) {
                    st += getString(R.string.rec_tester_col3_error1);
                    record = null;
                }
                if (record != null) {
                    if (record.getState() == AudioRecord.STATE_INITIALIZED)
{
                        int numOfReadShort;
                        try {  // try read some samples.
                            record.startRecording();
                            short[] audioSamples = new
short[recBufferSize];
                            numOfReadShort =
record.read(audioSamples, 0, recBufferSize);
                        } catch (IllegalStateException e) {
                            numOfReadShort = -1;
                        }
                        if (numOfReadShort > 0) {
                            st +=
getString(R.string.rec_tester_col3_succeed);
                        } else if (numOfReadShort == 0) {
                            st +=
getString(R.string.rec_tester_col3_error2);
                        } else {
                            st +=
getString(R.string.rec_tester_col3_error3);
                        }
                        int as = record.getAudioSource();
                        if (as != audioSourceId[ias]) {  // audio source
altered
                            st +=
getString(R.string.rec_tester_col3_hint1,

    analyzerUtil.getAudioSourceName(as));
                        }
                        record.stop();
                    } else {
                        st += getString(R.string.rec_tester_col3_error4);
                    }
                    record.release();
                }
                st += getString(R.string.rec_tester_col3_end);
                appendTextData(tv, st);
            }
        }

        testResultSt = tv.getText();
    }
}
```

MyPreferences

```
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.preference.ListPreference;
import android.preference.Preference;
import android.preference.PreferenceActivity;
import android.util.Log;

import java.util.Arrays;

// I was using an old cell phone -- API level 9 (android 2.3.6),
// so here use PreferenceActivity instead of PreferenceFragment.
// http://developer.android.com/guide/topics/ui/settings.html
@SuppressWarnings("deprecation")
public class MyPreferences extends PreferenceActivity {
    private static final String TAG = "MyPreference";
    private static String[] audioSources;
    private static String[] audioSourcesName;

    private static String getAudioSourceNameFromId(int id) {
        for (int i = 0; i < audioSources.length; i++) {
            if (audioSources[i].equals(String.valueOf(id))) {
                return audioSourcesName[i];
            }
        }
        Log.e(TAG, "getAudioSourceName(): no this entry.");
        return "";
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferences);

        // as soon as the user modifies a preference,
        // the system saves the changes to a default SharedPreferences file
    }

    private SharedPreferences.OnSharedPreferenceChangeListener prefListener =
        new SharedPreferences.OnSharedPreferenceChangeListener() {
        public void onSharedPreferenceChanged(SharedPreferences prefs, String key) {
            Log.i(TAG, key + "=" + prefs);
            if (key == null || key.equals("windowFunction")) {
                Preference connectionPref = findPreference(key);
                connectionPref.setSummary(prefs.getString(key, ""));
            }
            if (key == null || key.equals("audioSource")) {
                String asi = prefs.getString("audioSource",
getString(R.string.audio_source_id_default));
                int audioSourceId = Integer.parseInt(asi);
                Preference connectionPref = findPreference(key);
                connectionPref.setSummary(getAudioSourceNameFromId(audioSourceId));
            }
            if (key == null || key.equals("spectrogramColorMap")) {
                Preference connectionPref = findPreference(key);
                connectionPref.setSummary(prefs.getString(key, ""));
            }
        }
    };

    @Override
    protected void onResume() {
        super.onResume();

        // Get list of default sources
        Intent intent = getIntent();
        final int[] asid =
intent.getIntArrayExtra(AnalyzerActivity.MYPREFERENCES_MSG_SOURCE_ID);
        final String[] as =
intent.getStringArrayExtra(AnalyzerActivity.MYPREFERENCES_MSG_SOURCE_NAME);

        int nExtraSources = 0;
        for (int id : asid) {
            // See SamplingLoop::run() for the magic number 1000
            if (id >= 1000) nExtraSources++;
        }

        // Get list of supported sources
        AnalyzerUtil au = new AnalyzerUtil(this);
        final int[] audioSourcesId = au.GetAllAudioSource(4);
        Log.i(TAG, " n_as = " + audioSourcesId.length);
        Log.i(TAG, " n_ex = " + nExtraSources);
        audioSourcesName = new String[audioSourcesId.length + nExtraSources];
        for (int i = 0; i < audioSourcesId.length; i++) {
            audioSourcesName[i] = au.getAudioSourceName(audioSourcesId[i]);
        }

        // Combine these two sources
        audioSources = new String[audioSourcesName.length];
        int j = 0;
        for (; j < audioSourcesId.length; j++) {
            audioSources[j] = String.valueOf(audioSourcesId[j]);
        }
        for (int i = 0; i < asid.length; i++) {
            // See SamplingLoop::run() for the magic number 1000
            if (asid[i] >= 1000) {
                audioSources[j] = String.valueOf(asid[i]);
                audioSourcesName[j] = as[i];
                j++;
            }
        }

        final ListPreference lp = (ListPreference) findPreference("audioSource");
        lp.setDefaultValue(MediaRecorder.AudioSource.VOICE_RECOGNITION);
        lp.setEntries(audioSourcesName);
        lp.setEntryValues(audioSources);

        getPreferenceScreen().getSharedPreferences()
                .registerOnSharedPreferenceChangeListener(prefListener);
    }

    @Override
    protected void onPause() {
        super.onPause();
        getPreferenceScreen().getSharedPreferences()
                .unregisterOnSharedPreferenceChangeListener(prefListener);
    }
}
```

RangeViewDiaglogC

```
/* Copyright 2017 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
```

```
package github.bewantbe.audio_analyzer_for_android;

import android.annotation.SuppressLint;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.SharedPreferences;
import android.preference.PreferenceManager;
import android.text.Editable;
import android.text.TextWatcher;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.TextView;

import java.text.DecimalFormat;

/**
 * For showing and setting plot ranges,
 * including frequency (Hz) and loudness (dB).
 *
 * Ref. https://www.mkyong.com/android/android-prompt-user-input-dialog-example/
 */

class RangeViewDialogC {
    private static final String TAG = "RangeViewDialogC:";
    private AlertDialog rangeViewDialog = null;
    private View rangeViewView;

    private final AnalyzerActivity ct;
    private final AnalyzerGraphic graphView;
```

```
RangeViewDialogC(AnalyzerActivity _ct, AnalyzerGraphic _graphView) {
    ct = _ct;
    graphView = _graphView;
    buildDialog(ct);
}

// Watch if there is change in the EditText
private class MyTextWatcher implements TextWatcher {
    private EditText mEditText;

    MyTextWatcher(EditText editText) {
        mEditText = editText;
    }

    @Override
    public void beforeTextChanged(CharSequence charSequence, int i, int i1, int i2) {}

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
        mEditText.setTag(true);  // flag that indicate range been changed
    }

    @Override
    public void afterTextChanged(Editable editable) {}
}

private void SetRangeView(boolean loadSaved) {
    if (rangeViewDialog == null) {
        Log.d(TAG, "ShowRangeViewDialog(): rangeViewDialog is not prepared.");
        return;
    }
    double[] vals = graphView.getViewPhysicalRange();

    SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(ct);
    boolean isLock = sharedPref.getBoolean("view_range_lock", false);
    // If locked, load the saved value
    if (isLock || loadSaved) {
        double[] rr = new double[AnalyzerGraphic.VIEW_RANGE_DATA_LENGTH];
        for (int i = 0; i < rr.length; i++) {
            rr[i] = AnalyzerUtil.getDouble(sharedPref, "view_range_rr_" + i, 0.0 / 0.0);
            if (Double.isNaN(rr[i])) {  // not properly initialized
                Log.w(TAG, "LoadPreferences(): rr is not properly initialized");
                rr = null;
```

```
          break;
        }
      }
      if (rr != null)
        System.arraycopy(rr, 0, vals, 0, rr.length);
    }

    DecimalFormat df = new DecimalFormat("#.##");
    ((EditText) rangeViewView.findViewById(R.id.et_freq_setting_lower_bound))
        .setText(df.format(vals[0]));
    ((EditText) rangeViewView.findViewById(R.id.et_freq_setting_upper_bound))
        .setText(df.format(vals[1]));
    ((EditText) rangeViewView.findViewById(R.id.et_db_setting_lower_bound))
        .setText(df.format(vals[2]));
    ((EditText) rangeViewView.findViewById(R.id.et_db_setting_upper_bound))
        .setText(df.format(vals[3]));
    ((TextView) rangeViewView.findViewById(R.id.show_range_tv_fL))
        .setText(ct.getString(R.string.show_range_tv_fL));
    ((TextView) rangeViewView.findViewById(R.id.show_range_tv_fH))
        .setText(ct.getString(R.string.show_range_tv_fH,vals[6],vals[7]));
    ((TextView) rangeViewView.findViewById(R.id.show_range_tv_dBL))
        .setText(ct.getString(R.string.show_range_tv_dBL));
    ((TextView) rangeViewView.findViewById(R.id.show_range_tv_dBH))
        .setText(ct.getString(R.string.show_range_tv_dBH,vals[8],vals[9]));

    ((CheckBox) rangeViewView.findViewById(R.id.show_range_lock)).setChecked(isLock);
  }

  void ShowRangeViewDialog() {
    SetRangeView(false);

    // Listener for test if a field is modified
    int[] resList = {R.id.et_freq_setting_lower_bound, R.id.et_freq_setting_upper_bound,
        R.id.et_db_setting_lower_bound,   R.id.et_db_setting_upper_bound};
    for (int id : resList) {
      EditText et = (EditText) rangeViewView.findViewById(id);
      et.setTag(false);                                    // false = no modified
      et.addTextChangedListener(new MyTextWatcher(et));     // Am I need to remove
previous Listener first?
    }

    rangeViewDialog.show();
  }
```

```
    @SuppressLint("InflateParams")
    private void buildDialog(Context context) {
        LayoutInflater inflater = LayoutInflater.from(context);
        rangeViewView = inflater.inflate(R.layout.dialog_view_range, null);  // null because there is
no parent. https://possiblemobile.com/2013/05/layout-inflation-as-intended/
        rangeViewView.findViewById(R.id.show_range_button_load).setOnClickListener(
            new View.OnClickListener() {
                public void onClick(View v) {
                    SetRangeView(true);
                }
            }
        );
        AlertDialog.Builder freqDialogBuilder = new AlertDialog.Builder(context);
        freqDialogBuilder
                .setView(rangeViewView)
                .setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int id) {
                        boolean isLock = ((CheckBox)
rangeViewView.findViewById(R.id.show_range_lock)).isChecked();
                        double[] rangeDefault = graphView.getViewPhysicalRange();
                        double[] rr = new double[rangeDefault.length / 2];
                        int[] resList = {R.id.et_freq_setting_lower_bound,
R.id.et_freq_setting_upper_bound,
                                R.id.et_db_setting_lower_bound,   R.id.et_db_setting_upper_bound};
                        for (int i = 0; i < resList.length; i++) {
                            EditText et = (EditText) rangeViewView.findViewById(resList[i]);
                            if (et == null) Log.v(TAG, "  EditText[" + i + "] == null");
                            if (et == null) continue;
                            if (et.getTag() == null) Log.v(TAG, "  EditText[" + i + "].getTag == null");
                            if (et.getTag() == null || (boolean)et.getTag() || isLock) {
                                rr[i] = AnalyzerUtil.parseDouble(et.getText().toString());
                            } else {
                                rr[i] = rangeDefault[i];
                                Log.v(TAG, "  EditText[" + i + "] not change. rr[i] = " + rr[i]);
                            }
                        }
                        // Save setting to preference, after sanitized.
                        rr = graphView.setViewRange(rr, rangeDefault);
                        SaveViewRange(rr, isLock);
                        if (isLock) {
                            ct.stickToMeasureMode();
                            ct.viewRangeArray = rr;
                        } else {
                            ct.stickToMeasureModeCancel();
                        }
                    }
                })
                .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {
                        Log.v(TAG, "rangeViewDialog: Canceled");
                    }
                });
//    freqDialogBuilder
//            .setTitle("dialog_title");
        rangeViewDialog = freqDialogBuilder.create();
    }

    private void SaveViewRange(double[] rr, boolean isLock) {
        SharedPreferences sharedPref = PreferenceManager.getDefaultSharedPreferences(ct);
        SharedPreferences.Editor editor = sharedPref.edit();
        for (int i = 0; i < rr.length; i++) {
            AnalyzerUtil.putDouble(editor, "view_range_rr_" + i, rr[i]);  // no editor.putDouble ?
kidding me?
        }
        editor.putBoolean("view_range_lock", isLock);
        editor.commit();
    }
}
```

RecorderMonitor

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;
```

```
import android.os.SystemClock;
import android.util.Log;

/**
 * For checking Recorder running status.
 * Especially buffer overrun, i.e. too few data from the recorder.
 */
class RecorderMonitor {
    private static final String TAG0 = "RecorderMonitor:";
    private final String TAG;
    private long timeUpdateOld, timeUpdateInterval, timeStarted;  // in ms
    private long lastOverrunTime;
    private long nSamplesRead;
    private int sampleRate;
    private int bufferSampleSize;
    private double sampleRateReal;
    private boolean lastCheckOverrun = false;

    RecorderMonitor(int sampleRateIn, int bufferSampleSizeIn, String TAG1) {
        sampleRate = sampleRateIn;
        bufferSampleSize = bufferSampleSizeIn;
        timeUpdateInterval = 2000;
        TAG = TAG1 + TAG0;
    }

    // When start recording, call this
    void start() {
        nSamplesRead = 0;
        lastOverrunTime = 0;
        timeStarted = SystemClock.uptimeMillis();
        timeUpdateOld = timeStarted;
        sampleRateReal = sampleRate;
    }

    // Input number of audio frames that read
    // Return true if an overrun check is performed, otherwise false.
    boolean updateState(int numOfReadShort) {
        long timeNow = SystemClock.uptimeMillis();
        if (nSamplesRead == 0) {      // get overrun checker synchronized
            timeStarted = timeNow - numOfReadShort*1000/sampleRate;
        }
        nSamplesRead += numOfReadShort;
        if (timeUpdateOld + timeUpdateInterval > timeNow) {
            return false;  // do the checks below every timeUpdateInterval ms
        }
        timeUpdateOld += timeUpdateInterval;
        if (timeUpdateOld + timeUpdateInterval <= timeNow) {
          timeUpdateOld = timeNow;  // catch up the time (so that at most one output per
timeUpdateInterval)
        }
        long nSamplesFromTime = (long)((timeNow - timeStarted) * sampleRateReal / 1000);
        double f1 = (double) nSamplesRead / sampleRateReal;
        double f2 = (double) nSamplesFromTime / sampleRateReal;
//    Log.i(TAG, "Buffer"
//       + " should read " + nSamplesFromTime + " (" + Math.round(f2*1000)/1000.0 + "s),"
//       + " actual read " + nSamplesRead + " (" + Math.round(f1*1000)/1000.0 + "s)\n"
//       + " diff " + (nSamplesFromTime-nSamplesRead) + " (" + Math.round((f2-f1)*1000)/1e3 +
"s)"
//       + " sampleRate = " + Math.round(sampleRateReal*100)/100.0);
        // Check if buffer overrun occur
        if (nSamplesFromTime > bufferSampleSize + nSamplesRead) {
          Log.w(TAG, "SamplingLoop::run(): Buffer Overrun occurred !\n"
            + " should read " + nSamplesFromTime + " (" + Math.round(f2*1000)/1000.0 + "s),"
            + " actual read " + nSamplesRead + " (" + Math.round(f1*1000)/1000.0 + "s)\n"
            + " diff " + (nSamplesFromTime-nSamplesRead) + " (" + Math.round((f2-f1)*1000)/1e3
+ "s)"
            + " sampleRate = " + Math.round(sampleRateReal*100)/100.0
            + "\n Overrun counter reset.");
          lastOverrunTime = timeNow;
          nSamplesRead = 0;  // start over
        }
        // Update actual sample rate
        if (nSamplesRead > 10*sampleRate) {
          sampleRateReal = 0.9*sampleRateReal + 0.1*(nSamplesRead * 1000.0 / (timeNow -
timeStarted));
          if (Math.abs(sampleRateReal-sampleRate) > 0.0145*sampleRate) {  // 0.0145 = 25 cent
            Log.w(TAG, "SamplingLoop::run(): Sample rate inaccurate, possible hardware
problem !\n"
                + " should read " + nSamplesFromTime + " (" + Math.round(f2*1000)/1000.0 + "s),"
                + " actual read " + nSamplesRead + " (" + Math.round(f1*1000)/1000.0 + "s)\n"
                + " diff " + (nSamplesFromTime-nSamplesRead) + " (" + Math.round((f2-
f1)*1000)/1e3 + "s)"
                + " sampleRate = " + Math.round(sampleRateReal*100)/100.0
                + "\n Overrun counter reset.");
            nSamplesRead = 0;
          }
        }
        lastCheckOverrun = lastOverrunTime == timeNow;
```

```
    return true;  // state updated during this check
  }

  boolean getLastCheckOverrun() {
  return lastCheckOverrun;
 }

  long getLastOverrunTime() {
  return lastOverrunTime;
 }

  double getSampleRate() {
  return sampleRateReal;
 }
}
```

SamplingLoop

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.media.AudioFormat;
import android.media.AudioRecord;
import android.media.audiofx.AutomaticGainControl;
import android.os.Build;
import android.os.SystemClock;
import android.util.Log;

import java.util.Arrays;

/**
```

```
 * Read a snapshot of audio data at a regular interval, and compute the FFT
 * @author suhler@google.com
 *         bewantbe@gmail.com
 * Ref:
 *   https://developer.android.com/guide/topics/media/mediarecorder.html#example
 *   https://developer.android.com/reference/android/media/audiofx/AutomaticGainControl.html
 *
 * TODO:
 *   See also: High-Performance Audio
 *   https://developer.android.com/ndk/guides/audio/index.html
 *   https://developer.android.com/ndk/guides/audio/aaudio/aaudio.html
 */

class SamplingLoop extends Thread {
    private final String TAG = "SamplingLoop";
    private volatile boolean isRunning = true;
    private volatile boolean isPaused = false;
    private STFT stft;   // use with care
    private final AnalyzerParameters analyzerParam;

    private SineGenerator sineGen1;
    private SineGenerator sineGen2;
    private double[] spectrumDBcopy;   // XXX, transfers data from SamplingLoop to
AnalyzerGraphic

    private final AnalyzerActivity activity;

    volatile double wavSecRemain;
    volatile double wavSec = 0;
    public double peak3Ratio3Chunks;

    /**************AV Definitions Start*******************************************************/
    private static final int WORDDURATION_AFTERDETECTION = 1100;//milliseconds. We will
keep recording sound for this duration after the fist vocalization is detected. Please keep to the
minimum to reduce delay in reward.
    private static final int WORDDURATION_BEFOREDETECTION = 600;//milliseconds
    //private static final int WORDDURATION = WORDDURATION_BEFOREDETECTION +
WORDDURATION_AFTERDETECTION;//milliseconds

    private int nBuffers=0;
    private double[] rmsHistory;
    private static final int THRESHOLDBUFFERDURATION = 1000;// millisecons = 1sec
    public static StringBuilder  classIndicator = new StringBuilder("-");//this is used to indicate
each chunk's classification for the presence of vocalization
```

```
    //private final boolean bReportAnalysis=true;

    private final boolean bRecordChildOnly=true;//based on the valu of peak3Ration over three
buffers
    private double[] peak3RatioHistory;
    private final int THRESHOLD_PEAK3_RATIO_DB=50;
    /**************AV Definitions End*******************************************************/

    SamplingLoop(AnalyzerActivity _activity, AnalyzerParameters _analyzerParam) {
        activity = _activity;
        analyzerParam = _analyzerParam;

        isPaused = ((SelectorText) activity.findViewById(R.id.run)).getValue().equals("stop");
        // Signal sources for testing
        double fq0 = Double.parseDouble(activity.getString(R.string.test_signal_1_freq1));
        double amp0 = Math.pow(10, 1/20.0 *
Double.parseDouble(activity.getString(R.string.test_signal_1_db1)));
        double fq1 = Double.parseDouble(activity.getString(R.string.test_signal_2_freq1));
        double amp1 = Math.pow(10, 1/20.0 *
Double.parseDouble(activity.getString(R.string.test_signal_2_db1)));
        double fq2 = Double.parseDouble(activity.getString(R.string.test_signal_2_freq2));
        double amp2 = Math.pow(10, 1/20.0 *
Double.parseDouble(activity.getString(R.string.test_signal_2_db2)));
        if (analyzerParam.audioSourceId == 1000) {
            sineGen1 = new SineGenerator(fq0, analyzerParam.sampleRate,
analyzerParam.SAMPLE_VALUE_MAX * amp0);
        } else {
            sineGen1 = new SineGenerator(fq1, analyzerParam.sampleRate,
analyzerParam.SAMPLE_VALUE_MAX * amp1);
        }
        sineGen2 = new SineGenerator(fq2, analyzerParam.sampleRate,
analyzerParam.SAMPLE_VALUE_MAX * amp2);
    }

    private void SleepWithoutInterrupt(long millis) {
        try {
            Thread.sleep(millis);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private double baseTimeMs = SystemClock.uptimeMillis();
    private void LimitFrameRate(double updateMs) {
        // Limit the frame rate by wait `delay' ms.
        baseTimeMs += updateMs;
        long delay = (int) (baseTimeMs - SystemClock.uptimeMillis());
//      Log.i(TAG, "delay = " + delay);
        if (delay > 0) {
            try {
                Thread.sleep(delay);
            } catch (InterruptedException e) {
                Log.i(TAG, "Sleep interrupted");  // seems never reached
            }
        } else {
            baseTimeMs -= delay;  // get current time
            // Log.i(TAG, "time: cmp t="+Long.toString(SystemClock.uptimeMillis())
            //              + " v.s. t'=" + Long.toString(baseTimeMs));
        }
    }

    private double[] mdata;

    // Generate test data.
    private int readTestData(short[] a, int offsetInShorts, int sizeInShorts, int id) {
        if (mdata == null || mdata.length != sizeInShorts) {
            mdata = new double[sizeInShorts];
        }
        Arrays.fill(mdata, 0.0);
        switch (id - 1000) {
            case 1:
                sineGen2.getSamples(mdata);
                // No break, so values of mdata added.
            case 0:
                sineGen1.addSamples(mdata);
                for (int i = 0; i < sizeInShorts; i++) {
                    a[offsetInShorts + i] = (short) Math.round(mdata[i]);
                }
                break;
            case 2:
                for (int i = 0; i < sizeInShorts; i++) {
                    a[i] = (short) (analyzerParam.SAMPLE_VALUE_MAX * (2.0*Math.random() - 1));
                }
                break;
            default:
                Log.w(TAG, "readTestData(): No this source id = " + analyzerParam.audioSourceId);
        }
        // Block this thread, so that behave as if read from real device.
        LimitFrameRate(1000.0*sizeInShorts / analyzerParam.sampleRate);
        return sizeInShorts;
    }


    @Override
    public void run() {
        AudioRecord record;

        long tStart = SystemClock.uptimeMillis();
        try {
            activity.graphInit.join();  // TODO: Seems not working as intended....
        } catch (InterruptedException e) {
            Log.w(TAG, "run(): activity.graphInit.join() failed.");
        }
        long tEnd = SystemClock.uptimeMillis();
        if (tEnd - tStart < 500) {
            Log.i(TAG, "wait more.." + (500 - (tEnd - tStart)) + " ms");
            // Wait until previous instance of AudioRecord fully released.
            SleepWithoutInterrupt(500 - (tEnd - tStart));
        }

        int minBytes = AudioRecord.getMinBufferSize(analyzerParam.sampleRate,
AudioFormat.CHANNEL_IN_MONO,
                AudioFormat.ENCODING_PCM_16BIT);
        if (minBytes == AudioRecord.ERROR_BAD_VALUE) {
            Log.e(TAG, "SamplingLoop::run(): Invalid AudioRecord parameter.\n");
            return;
        }

        /*
          Develop -> Reference -> AudioRecord
             Data should be read from the audio hardware in chunks of sizes
             inferior to the total recording buffer size.
         */
        // Determine size of buffers for AudioRecord and AudioRecord::read()
        int readChunkSize = analyzerParam.hopLen;  // Every hopLen one fft result (overlapped
analyze window)
        readChunkSize = Math.min(readChunkSize, 2048);  // read in a smaller chunk, hopefully
smaller delay
        int bufferSampleSize = Math.max(minBytes / analyzerParam.BYTE_OF_SAMPLE,
analyzerParam.fftLen / 2) * 2;
        // tolerate up to about 1 sec.
    bufferSampleSize = (int) Math.ceil(1.0 * analyzerParam.sampleRate / bufferSampleSize) *
bufferSampleSize;

    // Use the mic with AGC turned off. e.g. VOICE_RECOGNITION for measurement
    // The buffer size here seems not relate to the delay.
    // So choose a larger size (~1sec) so that overrun is unlikely.
    try {
      if (analyzerParam.audioSourceId < 1000) {
        record = new AudioRecord(analyzerParam.audioSourceId,
analyzerParam.sampleRate, AudioFormat.CHANNEL_IN_MONO,
            AudioFormat.ENCODING_PCM_16BIT, analyzerParam.BYTE_OF_SAMPLE *
bufferSampleSize);
      } else {
        record = new AudioRecord(analyzerParam.RECORDER_AGC_OFF,
analyzerParam.sampleRate, AudioFormat.CHANNEL_IN_MONO,
            AudioFormat.ENCODING_PCM_16BIT, analyzerParam.BYTE_OF_SAMPLE *
bufferSampleSize);
      }
    } catch (IllegalArgumentException e) {
      Log.e(TAG, "Fail to initialize recorder.");
      activity.analyzerViews.notifyToast("Illegal recorder argument. (change source)");
      return;
    }

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN) {
      // Check Auto-Gain-Control status.
      if (AutomaticGainControl.isAvailable()) {
        AutomaticGainControl agc = AutomaticGainControl.create(
            record.getAudioSessionId());
        if (agc.getEnabled())
          Log.i(TAG, "SamplingLoop::Run(): AGC (automatic gain control): enabled.");
        else
          Log.i(TAG, "SamplingLoop::Run(): AGC (automatic gain control): disabled.");
      } else {
        Log.i(TAG, "SamplingLoop::Run(): AGC (automatic gain control): not available.");
      }
    }

    Log.i(TAG, "SamplingLoop::Run(): Starting recorder... \n" +
        "  source          : " + analyzerParam.getAudioSourceName() + "\n" +
        String.format("  sample rate     : %d Hz (request %d Hz)\n", record.getSampleRate(),
analyzerParam.sampleRate) +
        String.format("  min buffer size : %d samples, %d Bytes\n", minBytes /
analyzerParam.BYTE_OF_SAMPLE, minBytes) +
            String.format("  buffer size     : %d samples, %d Bytes\n", bufferSampleSize,
analyzerParam.BYTE_OF_SAMPLE * bufferSampleSize) +
            String.format("  read chunk size : %d samples, %d Bytes\n", readChunkSize,
analyzerParam.BYTE_OF_SAMPLE * readChunkSize) +
            String.format("  FFT length      : %d\n", analyzerParam.fftLen) +
            String.format("  nFFTAverage     : %d\n", analyzerParam.nFFTAverage));
    analyzerParam.sampleRate = record.getSampleRate();

    if (record.getState() == AudioRecord.STATE_UNINITIALIZED) {
      Log.e(TAG, "SamplingLoop::run(): Fail to initialize AudioRecord()");
      activity.analyzerViews.notifyToast("Fail to initialize recorder.");
      // If failed somehow, leave user a chance to change preference.
      return;
    }

    short[] audioSamples = new short[readChunkSize];
    int numOfReadShort;
    /**************AV Declarations Start*******************************************************/
    //create a buffer for data for the last 1.5 sec
    final int DATACHUNKS_TOSAVE_AFTERDETECTION =
((WORDDURATION_AFTERDETECTION * analyzerParam.sampleRate) / readChunkSize) /
1000;
    final int DATACHUNKS_TOSAVE_BEFOREDETECTION =
((WORDDURATION_BEFOREDETECTION * analyzerParam.sampleRate) / readChunkSize) /
1000;
    final int DATACHUNKS_TOSAVE_TOTAL =
DATACHUNKS_TOSAVE_BEFOREDETECTION +
DATACHUNKS_TOSAVE_AFTERDETECTION;//shall be 15 @8000 samples per sec and  23
@16000 samples per second
    //Log.i(TAG, "DATACHUNKS_TOSAVE_TOTAL=" + DATACHUNKS_TOSAVE_TOTAL);
    short[][] tempBuffers = new short[DATACHUNKS_TOSAVE_TOTAL][readChunkSize];//last
15 buffers (1.5sec) will be recorded here. When there was a vocalization, I will dump this buffers
into a file
    int tempBuffersIndex = 0;
    int nDataChunksToWait = 0;

    //create a buffer for threshold for the last 1 sec
    final int NUMBUFFHIST = ((THRESHOLDBUFFERDURATION *
analyzerParam.sampleRate) / readChunkSize) / 1000;//every buffer is approximately 100ms
    rmsHistory = new double[NUMBUFFHIST];
    for (int i = 0; i < NUMBUFFHIST; i++)
      rmsHistory[i] = 0;//remember means of 9 previous buffers. This is my amplitude
threshold
    Log.i(TAG, "NUMBUFFHIST=" + NUMBUFFHIST);
```

```
        peak3RatioHistory = new double[NUMBUFFHIST];
        for (int i = 0; i < NUMBUFFHIST; i++)
            peak3RatioHistory[i] = 0;//remember ratios of 9 previous buffers. This is my amplitude
threshold

        //for(int i=0; i<200; i++) classIndicator.append("-");//this is used to indicate each chunk's
classification for the presence of vocalization
        /**************AV Declarations End********************************************************/

        stft = new STFT(analyzerParam);
        stft.setAWeighting(analyzerParam.isAWeighting);
        if (spectrumDBcopy == null || spectrumDBcopy.length != analyzerParam.fftLen / 2 + 1) {
            spectrumDBcopy = new double[analyzerParam.fftLen / 2 + 1];
        }

        RecorderMonitor recorderMonitor = new RecorderMonitor(analyzerParam.sampleRate,
bufferSampleSize, "SamplingLoop::run()");
        recorderMonitor.start();

        WavWriter wavWriter = new WavWriter(analyzerParam.sampleRate);

        try {
            record.startRecording();
        }// Start recording
        catch (IllegalStateException e) {
            Log.e(TAG, "Fail to start recording.");
            activity.analyzerViews.notifyToast("Fail to start recording.");
            return;
        }


/************************************************************************************************************/
        // Main loop: When running in this loop (including when paused), you can not change
properties related to recorder: e.g. audioSourceId, sampleRate, bufferSampleSize
        while (isRunning) {
            // Read data
            if (analyzerParam.audioSourceId >= 1000) {
                numOfReadShort = readTestData(audioSamples, 0, readChunkSize,
analyzerParam.audioSourceId);
            } else {
                numOfReadShort = record.read(audioSamples, 0, readChunkSize);
            }   // pulling
        System.arraycopy(audioSamples, 0, tempBuffers[tempBuffersIndex], 0,
readChunkSize);//keep the last 1.5 seconds in memory
        tempBuffersIndex++;
        if (tempBuffersIndex >= DATACHUNKS_TOSAVE_TOTAL) tempBuffersIndex = 0;

        if (recorderMonitor.updateState(numOfReadShort)) {  // performed a check
          if (recorderMonitor.getLastCheckOverrun()) activity.analyzerViews.notifyOverrun();
        }

        if (nDataChunksToWait > 0) {//keep recording for 1 second after a vocalization was
detected
          nDataChunksToWait--;//decrement
          if (nDataChunksToWait == 0) {//all buffers have been filled => push data into the wav
file
            //wavWriter.start();
            //wavSecRemain = wavWriter.secondsLeft();//check for available computer memory
            //Log.i(TAG, "PCM write to file " + wavWriter.getPath());
            //for (int i = 0; i < DATACHUNKS_TOSAVE_TOTAL; i++) {
              //wavWriter.pushAudioShort(tempBuffers[(tempBuffersIndex + i) %
DATACHUNKS_TOSAVE_TOTAL], readChunkSize);
            }
            //wavSec = wavWriter.secondsWritten();
            //activity.analyzerViews.updateRec(wavSec);
            //wavWriter.stop();
            //Log.i(TAG, "SamplingLoop::Run(): Writing an end to the saved wav.");
            //activity.analyzerViews.notifyWAVSaved(wavWriter.relativeDir);

            /********************************The Wav file is ready => Send API to
SpeechAce************************************************************/
            //The Wav file is ready => Send API to SpeechAce
            /********************************************************************************************/
         //  }
        }

        if (isPaused) {
          continue;
        }

        stft.feedData(audioSamples, numOfReadShort);

        // If there is new spectrum data, do plot
        if (stft.nElemSpectrumAmp() >= analyzerParam.nFFTAverage) {
          // Update spectrum or spectrogram
          final double[] spectrumDB = stft.getSpectrumAmpDB();
```

```
            System.arraycopy(spectrumDB, 0, spectrumDBcopy, 0, spectrumDB.length);
            activity.analyzerViews.update(spectrumDBcopy);
//          fpsCounter.inc();

            stft.calculatePeak();
            activity.maxAmpFreq = stft.maxAmpFreq;
            activity.maxAmpDB = stft.maxAmpDB;

            // get RMS
            activity.dtRMS = stft.getRMS();
            activity.dtRMSFromFT = stft.getRMSFromFT(300);//RMS shall not include low
frequency noise. Include only frequency where we can find vocalizations, i.e. above 300Hz


            /**************AV Playground Start********************************************************/
            nBuffers++;
            if (nBuffers == Integer.MAX_VALUE) nBuffers = NUMBUFFHIST;

            //1. remember the RMS of last 8 buffers in order to generate a threshold. The current
buffer is rmsHistory[0]; the oldest buffer is rmsHistory[NUMBUFFHIST-1]
            for (int i = (NUMBUFFHIST - 1); i > 0; i--) {
               rmsHistory[i] = rmsHistory[i - 1];
            }//remember the RMS of last 8 buffers
            rmsHistory[0] = activity.dtRMSFromFT;//the current buffer Math.round(mean);

            //2. determine the threshold for this buffer
            double ampThreshold = 0;
            for (int i = 0; i < NUMBUFFHIST - 1; i++)
               ampThreshold += rmsHistory[i];//do not include the current buffer
            ampThreshold /= (NUMBUFFHIST - 1);//classIndicator = "ampThreshold=" +
String.valueOf(ampThreshold) + "; activity.dtRMSFromFT=" +
String.valueOf(activity.dtRMSFromFT);

            //3. Identify and describe harmonics in freq domain
            //ToDo: consider more harmonics and dynamically identified regions: so the 1st peak
is identified then the 2nd peak to the right, then the peak to the left. That will entail identifying
local minima.
            int freqOfPeak1 = stft.getIndexOfPeakFreq(200, 800); //classIndicator = "maxFreq=" +
String.valueOf(maxFreq);//Log.i(TAG,String.valueOf(spectrumDB[100])+";
"+String.valueOf(spectrumDB[101])+"; "+String.valueOf(spectrumDB[102])+";
"+String.valueOf(spectrumDB[103])+"; "+String.valueOf(spectrumDB[104])+";
"+String.valueOf(spectrumDB[105]));
            int freqOfPeak2 = stft.getIndexOfPeakFreq(800, 1800);
            int freqOfPeak3 = stft.getIndexOfPeakFreq(1800, 3500);
```

```
        double mean = stft.getMeanPSD_dB(300, 3500); // 300
        double max1 = stft.getPeakAmpl_dB(freqOfPeak1);
        double max2 = stft.getPeakAmpl_dB(freqOfPeak2);
        double max3 = stft.getPeakAmpl_dB(freqOfPeak3);
        double ratio1 = stft.getPeakToMeanRatio(freqOfPeak1, 120);//1. The ratio tells me
how high the harmonic stands above background LOCALLY.
        double ratio2 = stft.getPeakToMeanRatio(freqOfPeak2, 500);
        double ratio3 = stft.getPeakToMeanRatio(freqOfPeak3, 500);

        //4. i'd like to make sure that each harmonic is not some kind of not-audible digital
artifact. For that I am using absolute values in dB.
        boolean bP1, bP2, bP3;
        bP1 = bP2 = bP3 = false;//markers for the three peaks standing above the
background;
        double minPeakPower = mean + 20.0;//min peak power in dB
        if (max1 > minPeakPower) bP1 = true;
        if (max2 > minPeakPower) bP2 = true;
        if (max3 > minPeakPower) bP3 = true;

        //5. Weighted ratio: use only harmonics with minimum power. We really need to
remove meaningless ratios that do not represent any sound, these are basically just artifacts
        double weightedRatio = 0;
        if (bP1 && bP2 && bP3) weightedRatio = (ratio1 + ratio2 + ratio3) / 3;
        else if (bP2 && bP3) weightedRatio = (ratio2 + ratio3) / 2;
        else if (bP1 && bP3) weightedRatio = (ratio1 + ratio3) / 2;
        else if (bP1 && bP2) weightedRatio = (ratio1 + ratio2) / 2;
        else if (bP3) weightedRatio = ratio3;
        else if (bP2) weightedRatio = ratio2;
        else if (bP1) weightedRatio = ratio1;
        else weightedRatio = -1;//'m' - none of the peaks stand above background.

        //5B. Store the history of the 3d peak ratio
        for (int i = (NUMBUFFHIST - 1); i > 0; i--) {
            peak3RatioHistory[i] = peak3RatioHistory[i - 1];
        }//remember the ratio3 of last 8 buffers
        peak3RatioHistory[0] = ratio3;//
        peak3Ratio3Chunks = (peak3RatioHistory[0] + peak3RatioHistory[1] +
peak3RatioHistory[2]) / 3;


        //6. Assign the classifier for this buffer based on weightedRatio
        String bC = "";//string to hold the classifier
        if (weightedRatio == -1) bC = "m";//'m' - none of the peaks stand above background.
        else if (!bP2 && !bP3 && freqOfPeak1 < 295)
```

```
          bC = "f";//freq of the only peak is too low
        else if (weightedRatio > 10) bC = "a";//vocalization was detected
        else if (weightedRatio > 9) bC = "k";//'k' for candidate
        else if (weightedRatio > 6)
          bC = "e";//'e' for candidate who almost did it., i.e. 70% of min_ratio.
        else bC = "~"; //~ for ratio too small to become a candidate

        //7. if the buffer contains loud noise that crosses threshold AND has some reasonable
harmonics=>mark as 'w'=buffer with vocalization.
        if (nBuffers >= NUMBUFFHIST && activity.dtRMSFromFT > (4 * ampThreshold) &&
(bC == "k" || bC == "e"))
          bC = "t";//Threshold crossing //classIndicator="Crossed Threshold"; else
classIndicator="Not crossed threshold";

        //8. Find the beginning of the vocalization as the fist buffer that crossed the threshold -
this approach is NOT USED. We simply always grabe 600ms before the data chunk with
vocalization
        boolean bReportAnalysis = false;
        if (bC == "a" || bC == "t")
        {
          bReportAnalysis = true;//report all info only for these buffers
          if (!bRecordChildOnly || peak3Ratio3Chunks > THRESHOLD_PEAK3_RATIO_DB)
          {//this buffer is definitely part of a vocalization => wait for
DATACHUNKS_TOSAVE_AFTERDETECTION buffers (1sec), then dump the
tempAudioSample into a file
            if (nDataChunksToWait == 0) {
              nDataChunksToWait = DATACHUNKS_TOSAVE_AFTERDETECTION;
              Log.i(TAG, "I have detected a word. I will wait for 1sec
(DATACHUNKS_TOSAVE_AFTERDETECTION) and save a wav file");}
            }//this is the marker for vocalization. It is also a counter of buffers to wait
            else {
              Log.i(TAG, "I have detected a vocalization in this data chunk, but I am saving
this data chink as part of a previous word and therefore I will NOT restart buffering.");
            }
          /*classIndicator.insert(0,"w");//this buffer is part of vocalization //report each data
chunk classification with classIndicator that will look like this: wwwwww~~~~~mmfee~wwwwww
          if (rmsHistory[0] >= ampThreshold) {//if there was noise before this buffer count that
noise as beginning of a word
            if      (nBuffers >= 4 && rmsHistory[3] > ampThreshold)
{classIndicator.setCharAt(4, 'w');classIndicator.setCharAt(3, 'w');classIndicator.setCharAt(2,
'w');classIndicator.setCharAt(1, 'w');}//look as far away as four buffers=> this is part of the same
word
```

```
                 else if (nBuffers >= 3 && rmsHistory[2] > ampThreshold)
{classIndicator.setCharAt(3, 'w');classIndicator.setCharAt(2, 'w');classIndicator.setCharAt(1,
'w');}//look as far away as three buffers=> this is part of the same word
                 else if (nBuffers >= 2 && rmsHistory[1] > ampThreshold)
{classIndicator.setCharAt(2, 'w');classIndicator.setCharAt(1, 'w');}
                 else if (nBuffers >= 1) {classIndicator.setCharAt(1, 'w');}//add one buffer on the
edge
              }*/
              }
              //else classIndicator.insert(0,"-");;

              //9. Report the classification of the current buffer //Consider reporting peak
thickness
              if (classIndicator.length() > 200) classIndicator.deleteCharAt(200);
              classIndicator.insert(0, bC);
              if (bReportAnalysis) {
                 Log.i(TAG, classIndicator.toString());
              }
              if (bReportAnalysis) {
                 Log.i(TAG, bC + "; wR=" + String.valueOf(Math.round(weightedRatio)) +
                       " |  f1=" + String.valueOf(freqOfPeak1) + ", m1=" +
String.valueOf(Math.round(max1)) + "dB, r1=" + String.valueOf(Math.round(ratio1)) +
                       " |  f2=" + String.valueOf(freqOfPeak2) + ", m2=" +
String.valueOf(Math.round(max2)) + "dB, r2=" + String.valueOf(Math.round(ratio2)) +
                       " |  f3=" + String.valueOf(freqOfPeak3) + ", m3=" +
String.valueOf(Math.round(max3)) + "dB, r3=" + String.valueOf(Math.round(ratio3)) +
                       " |  r3_1=" + String.valueOf(Math.round(peak3RatioHistory[1])) + ", r3_2=" +
String.valueOf(Math.round(peak3RatioHistory[2])) + ", r3_3=" +
String.valueOf(Math.round(peak3RatioHistory[3])) +
                       " |  peak3Ratio3Chunks=" +
String.valueOf(Math.round(peak3Ratio3Chunks)) + "  |  mean=" +
String.valueOf(Math.round(mean)) + "dB"
                 );

              }
              //10. Push the last 1.5 sec into a file
              //See above: if(nDataChunksToWait>0)

              /*************AV Playground End****************************************************/
           }
        }
        Log.i(TAG, "SamplingLoop::Run(): Actual sample rate: " +
recorderMonitor.getSampleRate());
        Log.i(TAG, "SamplingLoop::Run(): Stopping and releasing recorder.");
```

```
            record.stop();
            record.release();
        }

    void setAWeighting(boolean isAWeighting) {
        if (stft != null) {
            stft.setAWeighting(isAWeighting);
        }
    }

    void setPause(boolean pause) {
        this.isPaused = pause;
    }

    boolean getPause() {
        return this.isPaused;
    }

    void finish() {
        isRunning = false;
        interrupt();
    }
}
```

SBNumFormat

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.util.Log;
```

```
class SBNumFormat {
  private static final char charDigits[] = {'0','1','2','3','4','5','6','7','8','9'};

  // Invent wheel... so we can eliminate GC
  static void fillInNumFixedWidthPositive(StringBuilder sb, double d, int nInt, int nFrac, char
padChar) {
    if (d<0) {
      for (int i = 0; i < nInt+nFrac+(nFrac>0?1:0); i++) {
        sb.append(padChar);
      }
      Log.w("SBNumFormat", "fillInNumFixedWidthPositive: negative number");
      return;
    }
    if (d >= Math.pow(10,nInt)) {
      sb.append("OFL");
      for (int i = 3; i < nInt+nFrac+(nFrac>0?1:0); i++) {
        sb.append(' ');
      }
      return;
    }
    if (Double.isNaN(d)) {
      sb.append("NaN");
      for (int i = 3; i < nInt+nFrac+(nFrac>0?1:0); i++) {
        sb.append(' ');
      }
      return;
    }
    while (nInt > 0) {
      nInt--;
      if (d < Math.pow(10,nInt) && nInt>0) {
        if (padChar != '\0') {
          sb.append(padChar);
        }
      } else {
        sb.append(charDigits[(int)(d / Math.pow(10,nInt) % 10.0)]);
      }
    }
    if (nFrac > 0) {
      sb.append('.');
      for (int i = 1; i <= nFrac; i++) {
        sb.append(charDigits[(int)(d * Math.pow(10,i) % 10.0)]);
      }
    }
}

static void fillInNumFixedWidthPositive(StringBuilder sb, double d, int nInt, int nFrac) {
  fillInNumFixedWidthPositive(sb, d, nInt, nFrac, ' ');
}

static void fillInNumFixedFrac(StringBuilder sb, double d, int nInt, int nFrac) {
  if (d < 0) {
    sb.append('-');
    d = -d;
  }
  fillInNumFixedWidthPositive(sb, d, nInt, nFrac, '\0');
}

static void fillInNumFixedWidth(StringBuilder sb, double d, int nInt, int nFrac) {
  int it = sb.length();
  sb.append(' ');
  if (d < 0) {
    fillInNumFixedWidthPositive(sb, -d, nInt, nFrac);
    for (int i = it; i < sb.length(); i++) {
      if (sb.charAt(i+1) != ' ') {
        sb.setCharAt(i, '-');
        return;
      }
    }
  }
  fillInNumFixedWidthPositive(sb, d, nInt, nFrac);
}

static void fillInNumFixedWidthSigned(StringBuilder sb, double d, int nInt, int nFrac) {
  int it = sb.length();
  sb.append(' ');
  fillInNumFixedWidthPositive(sb, Math.abs(d), nInt, nFrac);
  for (int i = it; i < sb.length(); i++) {
    if (sb.charAt(i+1) != ' ') {
      if (d < 0) {
        sb.setCharAt(i, '-');
      } else {
        sb.setCharAt(i, '+');
      }
      return;
    }
  }
}
```

```
static void fillInNumFixedWidthSignedFirst(StringBuilder sb, double d, int nInt, int nFrac) {
  if (d < 0) {
    sb.append('-');
  } else {
    sb.append('+');
  }
  fillInNumFixedWidthPositive(sb, Math.abs(d), nInt, nFrac);
}

static void fillInInt(StringBuilder sb, int in) {
  if (in == 0) {
    sb.append('0');
    return;
  }
  if (in<0) {
    sb.append('-');
    in = -in;
  }
  int it = sb.length();
  while (in > 0) {
    sb.insert(it, in%10);
    in /= 10;
  }
}

static void fillTime(StringBuilder sb, double t, int nFrac) {
  // in format x0:00:00.x
  if (t<0) {
    t = -t;
    sb.append('-');
  }
  double u;
  // hours
  u = Math.floor(t/3600.0);
  fillInInt(sb, (int)u);
  sb.append(':');
  // minutes
  t -= u * 3600;
  u = Math.floor(t/60.0);
  fillInNumFixedWidthPositive(sb, u, 2, 0, '0');
  sb.append(':');
  // seconds
  t -= u * 60;
```

```
    fillInNumFixedWidthPositive(sb, t, 2, nFrac, '0');
  }
}
```

ScreenPhysicalMapping

```
/* Copyright 2017 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.util.Log;

import static java.lang.Math.exp;
import static java.lang.Math.log;

/**
 * Mapping between physical value and screen pixel position
 * Use double or float ?
 */

//    | lower bound  ...  higher bound |  physical unit
//    | 0            ...             1 |  "unit 1" (Mapping can be linear or logarithmic)

// In LINEAR mode (default):
//    |lower value  ...    higher value|  physical unit
//    | shift       ... shift + 1/zoom |  "unit 1", 0=vLowerBound, 1=vUpperBound
//    | 0 | 1 |     ...          | n-1 |  pixel

// In LINEAR_ON mode (not implemented):
//      |lower value ...    higher value|     physical unit
//      | shift      ... shift + 1/zoom |     "unit 1" window
//    | 0 | 1 |      ...             | n-1 |  pixel
```

```
class ScreenPhysicalMapping {
   private final static String TAG = "ScreenPhysicalMapping";

   enum Type {  // java's enum type is inconvenient
      LINEAR(0), LINEAR_ON(1), LOG(2);

      private final int value;
      Type(int value) { this.value = value; }
      public int getValue() { return value; }
   }

   Type mapType;                                    // Linear or Log
   double nCanvasPixel;
   double vLowerBound, vUpperBound;                  // Physical limits
   private double vLowerViewBound, vUpperViewBound;  // view bounds
   private double zoom = 1, shift = 0;              // zoom==1: no zooming, shift=0: no shift

   ScreenPhysicalMapping(double _nCanvasPixel, double _vLowerBound, double
_vHigherBound, ScreenPhysicalMapping.Type _mapType) {
      nCanvasPixel = _nCanvasPixel;
      vLowerBound = _vLowerBound;
      vUpperBound = _vHigherBound;
      vLowerViewBound = vLowerBound;
      vUpperViewBound = vUpperBound;
      mapType      = _mapType;
   }

   ScreenPhysicalMapping(ScreenPhysicalMapping _axis) {
      mapType      = _axis.mapType;
      nCanvasPixel = _axis.nCanvasPixel;
      vLowerBound  = _axis.vLowerBound;
      vUpperBound  = _axis.vUpperBound;
      vLowerViewBound = _axis.vLowerViewBound;
      vUpperViewBound = _axis.vUpperViewBound;
      zoom         = _axis.zoom;
      shift        = _axis.shift;
   }

   void setNCanvasPixel(double _nCanvasPixel) {
      nCanvasPixel = _nCanvasPixel;
   }

   void setBounds(double _vL, double _vU) {
      vLowerBound = _vL;
      vUpperBound = _vU;
      // Reset view range, preserve zoom and shift
      setZoomShift(zoom, shift);
      if (AnalyzerUtil.isAlmostInteger(vLowerViewBound)) {  // dirty fix...
          vLowerViewBound = Math.round(vLowerViewBound);
      }
      if (AnalyzerUtil.isAlmostInteger(vUpperViewBound)) {
          vUpperViewBound = Math.round(vUpperViewBound);
      }
      setViewBounds(vLowerViewBound, vUpperViewBound);  // refine zoom shift
   }

   double getZoom() { return zoom; }
   double getShift() { return shift; }

   void setZoomShift(double _zoom, double _shift) {
      zoom = _zoom;
      shift = _shift;
      vLowerViewBound = vFromUnitPosition(0, zoom, shift);
      vUpperViewBound = vFromUnitPosition(1, zoom, shift);
   }

   // set zoom and shift from physical value bounds
   void setViewBounds(double vL, double vU) {
      if (vL == vU) {
          return;  // Or throw an exception?
      }
      double p1 = UnitPositionFromV(vL, vLowerBound, vUpperBound);
      double p2 = UnitPositionFromV(vU, vLowerBound, vUpperBound);
      zoom  = 1 / (p2 - p1);
      shift = p1;
      vLowerViewBound = vL;
      vUpperViewBound = vU;
   }

   double UnitPositionFromV(double v, double vL, double vU) {
      if (vL == vU) {
          return 0;
      }
      if (mapType == Type.LINEAR) {
          return (v - vL) / (vU - vL);
      } else {
          return log(v/vL) / log(vU/vL);
      }
   }

   double vFromUnitPosition(double u, double zoom, double shift) {
      if (zoom == 0) {
          return 0;
      }
      if (mapType == Type.LINEAR) {
          return (u / zoom + shift) * (vUpperBound - vLowerBound) + vLowerBound;
      } else {
          return exp((u / zoom + shift) * log(vUpperBound / vLowerBound)) * vLowerBound;
      }
   }

   double pixelFromV(double v) {
      return UnitPositionFromV(v, vLowerViewBound, vUpperViewBound) * nCanvasPixel;
   }

   double vFromPixel(double pixel) {
      if (nCanvasPixel == 0)
          return vLowerViewBound;
      return vFromUnitPosition(pixel / nCanvasPixel, zoom, shift);
   }

   double vMinInView() {
      return vLowerViewBound;
   }

   double vMaxInView() {
      return vUpperViewBound;
   }

   double pixelNoZoomFromV(double v) {
      return UnitPositionFromV(v, vLowerBound, vUpperBound) * nCanvasPixel;
   }

   double diffVBounds() { return vUpperBound - vLowerBound; }

   void reverseBounds() {
      double oldVL = vLowerViewBound;
      double oldVU = vUpperViewBound;
      setBounds(vUpperBound, vLowerBound);
      setViewBounds(oldVU, oldVL);
   }
```

```
    void setMappingType(ScreenPhysicalMapping.Type _mapType, double lower_bound_ref) {
        // Set internal variables if possible
        double vL = vMinInView();
        double vH = vMaxInView();
        if (_mapType == Type.LOG) {
            if (vLowerBound == 0) vLowerBound = lower_bound_ref;
            if (vUpperBound == 0) vUpperBound = lower_bound_ref;
        } else {
            if (vLowerBound == lower_bound_ref) vLowerBound = 0;
            if (vUpperBound == lower_bound_ref) vUpperBound = 0;
        }
        boolean bNeedUpdateZoomShift = mapType != _mapType;
        mapType = _mapType;
        if (!bNeedUpdateZoomShift || nCanvasPixel == 0 || vLowerBound == vUpperBound) {
            return;
        }

        // Update zoom and shift
        // lower_bound_ref is for how to map zero
        // Only support non-negative bounds
        if (_mapType == Type.LOG) {
            if (vL < 0 || vH < 0) {
                Log.e(TAG, "setMappingType(): negative bounds.");
                return;
            }
            if (vL  < lower_bound_ref) vL = lower_bound_ref;
            if (vH  < lower_bound_ref) vH = lower_bound_ref;
        } else {
            if (vL  <= lower_bound_ref) vL = 0;
            if (vH  <= lower_bound_ref) vH = 0;
        }
        setViewBounds(vL, vH);
    }
}
```

SelectorText

```
/* Copyright 2011 Google Inc.
 *
 *Licensed under the Apache License, Version 2.0 (the "License");
 *you may not use this file except in compliance with the License.
 *You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 *Unless required by applicable law or agreed to in writing, software
 *distributed under the License is distributed on an "AS IS" BASIS,
 *WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *See the License for the specific language governing permissions and
 *limitations under the License.
 *
 * @author Stephen Uhler
 */

package github.bewantbe.audio_analyzer_for_android;

import android.annotation.SuppressLint;
import android.content.Context;
import android.content.res.TypedArray;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.RectF;
import android.util.AttributeSet;
import android.util.Log;
import android.view.animation.Animation;
import android.view.animation.Animation.AnimationListener;
import android.view.animation.RotateAnimation;
import android.widget.TextView;

/**
 * Text view that toggles through a set of values.
 * @author suhler@google.com (Stephen Uhler)
 */

public class SelectorText extends TextView {
  static final String TAG = "SelectorText:";
  private static float DPRatio;
  private static final int ANIMATION_DELAY = 70;
  private int value_id = 0;
  private String[] values = new String[0];
  private String[] valuesDisplay = new String[0];
  private Paint paint, bg;
  private RectF rect = new RectF();
  private RectF bgRect = new RectF();
  private float r;

  public SelectorText(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
    setup(context, attrs);
  }
  public SelectorText(Context context, AttributeSet attrs) {
    super(context, attrs);
    setup(context, attrs);
  }
  public SelectorText(Context context) {
    super(context);
    setup(context, null);
  }

  @SuppressLint("ClickableViewAccessibility")
  @Override
  public boolean performClick() {
    setText(getText());  // fix the no-animation bug
    //setText(valuesDisplay[value_id]);
    Animation an = createAnimation(true, ANIMATION_DELAY);
    an.setAnimationListener(new AnimationListener() {
      @Override
      public void onAnimationEnd(Animation animation) {
        nextValue();
        SelectorText.super.performClick();
        createAnimation(false, ANIMATION_DELAY).start();
      }
      @Override public void onAnimationRepeat(Animation animation) {}
      @Override public void onAnimationStart(Animation animation) {}
    });
    an.start();
    return true;
  }

  /**
   * Choose an arbitrary animation for the text view.
   * @param start   If true, animate the old value "out", otherwise animate the old value in
   * @param millis  Animation time for this step, ms
   */

  private Animation createAnimation(boolean start, int millis) {
    RotateAnimation ra = new RotateAnimation(start?0f:180f, start?180f:360f, getWidth()/2,
getHeight()/2);
//    Log.d("SelectorText", "  createAnimation(): ");
    ra.setDuration(millis);
    setAnimation(ra);
    return ra;
  }

  /**
   * Compute the value of our "select" indicator.
   */

  @Override
  protected void onSizeChanged (int w, int h, int oldw, int oldh) {
    rect.set(2f*DPRatio, h/2 - 5f*DPRatio, 12f*DPRatio, h/2 + 7f*DPRatio);
    bgRect.set(1f*DPRatio, 1f*DPRatio, w - 2f*DPRatio, h - 1f*DPRatio);
  }

  /**
   * Draw the selector, then the selected text.
   */
  @Override
  protected void onDraw(Canvas c) {
    super.onDraw(c);
    c.drawRoundRect(rect, r, r, paint);
    c.drawRoundRect(bgRect, r, r, bg);
  }

  /**
   * Initialize our selector.  We could make most of the features customizable via XML.
   */

  private void setup(Context context, AttributeSet attrs) {
    DPRatio = getResources().getDisplayMetrics().density;
    r = 3 * DPRatio;

    bg = new Paint();
    bg.setStrokeWidth(2*DPRatio);
    bg.setColor(Color.GRAY);
    bg.setStyle(Paint.Style.STROKE);
    paint = new Paint(bg);
    paint.setColor(Color.GREEN);

    setClickable(true);
    if (attrs != null) {
      TypedArray a = context.obtainStyledAttributes(attrs, R.styleable.SelectorText);
      String items = a.getString(R.styleable.SelectorText_items);
      String delim = getValue(a, R.styleable.SelectorText_itemDelim, " ");
      String itemsDisplay = a.getString(R.styleable.SelectorText_itemsDisplay);
```

```
      if (items != null) {
//        Log.i(AnalyzerActivity.TAG, "items: " + items);
        if (itemsDisplay != null && itemsDisplay.length() > 0) {
          setValues(items.split(delim), itemsDisplay.split("::"));
        } else {
          setValues(items.split(delim), items.split(delim));
        }
      }
      a.recycle();
    }
    if (valuesDisplay.length > 0) {
      setText(valuesDisplay[0]);
    }
  }

  private static String getValue(TypedArray a, int index, String dflt) {
    String result = a.getString(index);
    return result == null ? dflt : result;
  }

  public void setValues(String[] values, String[] valuesDisplay) {
    this.values = values;
    if (values.length == valuesDisplay.length) {
      this.valuesDisplay = valuesDisplay;
    } else {
      Log.w(TAG, "values.length != valuesDisplay.length");
      this.valuesDisplay = values;
    }
    adjustWidth();
    invalidate();
  }

  public String getValue() { return values[value_id]; }

  public String[] getValues() {
    return values;
  }

  public void setValue(String v) {
    for (int i = 0; i < values.length; i++) {
      if (! v.equals(values[value_id])) {
        nextValue();
      }
    }
  }

  public String nextValue() {
    if (values.length != 0) {
      value_id++;
      if (value_id >= values.length) {
        value_id = 0;
      }
      setText(valuesDisplay[value_id]);
      return valuesDisplay[value_id];
    }
    return getText().toString();
  }

  private void adjustWidth() {
    Paint p = getPaint();
    int adj = getPaddingLeft() + getPaddingRight();
    int width = 0;
    for (String s : valuesDisplay) {
      width = Math.max(width, Math.round(p.measureText(s)));
    }
    setMinWidth(width + adj + (int)(4*DPRatio));
  }
}
```

SineGenerator

```
/* Copyright 2011 Google Inc.
 *
 *Licensed under the Apache License, Version 2.0 (the "License");
 *you may not use this file except in compliance with the License.
 *You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 *Unless required by applicable law or agreed to in writing, software
 *distributed under the License is distributed on an "AS IS" BASIS,
 *WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 *See the License for the specific language governing permissions and
 *limitations under the License.
 *
 * @author Stephen Uhler
 */

package github.bewantbe.audio_analyzer_for_android;
```

```
/**
 * Recursive sine wave generator
 * - compute parameters using double()
 *
 * y[n] = 2cos(w)y[n-1] - y[n-2]
 * w = 2 pi f / fs
 */

class SineGenerator {
  private double fs;      // sampling frequency
  private double k;       // recursion constant
  private double n0, n1;          // first (next) 2 samples

  /**
   * Create a sine wave generator:
   * @param f             frequency of the sine wave (hz)
   * @param fs            Sampling rate (hz)
   * @param a       Amplitude
   */

  SineGenerator(double f, double fs, double a) {
    this.fs = fs;
    double w = 2.0 * Math.PI * f / fs;
    this.n0 = 0d;
    this.n1 = a * Math.cos(w + Math.PI/2.0);
    this.k  = 2.0 * Math.cos(w);
  }

  /**
   * Set the new frequency, maintaining the phase
   * @param f       New frequency, hz
   */

  public void setF(double f) {
    double w = 2.0 * Math.PI * f / fs;
    k =  getK(f);

    double theta = Math.acos(n0);
    if (n1 > n0) theta = 2 * Math.PI - theta;
    n0 = Math.cos(theta);
    n1 = Math.cos(w + theta);
  }
```

```java
/**
 * Compute the recursion coefficient "k"
 */
private double getK(double f) {
  double w = 2.0 * Math.PI * f / fs;
  return  2.0 * Math.cos(w);
}

/**
 * Generate the next batch of samples.
 * @param samples            Where to put the samples
 * @param start              Start sample
 * @param count              # of samples (must be even)
 */

private void getSamples(double[] samples, int start, int count) {
  for(int cnt = start; cnt < count; cnt += 2) {
    samples[cnt] = n0 = (k * n1) - n0;
    samples[cnt + 1] = n1 = (k * n0) - n1;
  }
}

/**
 * Fill the supplied (even length) array with samples.
 */

void getSamples(double[] samples) {
  getSamples(samples, 0, samples.length);
}

/**
 * Add samples to an existing buffer
 * @param samples            Where to put the samples
 * @param start              Start sample
 * @param count              # of samples (must be even)
 */
private void addSamples(double[] samples, int start, int count) {
  for(int cnt=start; cnt<count; cnt+=2) {
    samples[cnt] += n0 = (k * n1) - n0;
    samples[cnt + 1] += n1 = (k * n0) - n1;
  }
}

/**
```

```
   * Add samples to the supplied (even length) array.
   */
  void addSamples(double[] samples) {
    addSamples(samples, 0, samples.length);
  }

  /**
   * Get the current sampling frequency.
   */

  public double getFs() {
    return fs;
  }
}
```

SpetrogramBMP

```
package github.bewantbe.audio_analyzer_for_android;

import static java.lang.Math.log10;
import static java.lang.Math.pow;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.Log;

import java.util.Arrays;

/**
 * Plot the raw spectrogram BMP.
 */

class SpectrogramBMP {
    final static String TAG = "SpectrogramBMP";

    private int[] cma = ColorMapArray.hot;
    double dBLowerBound = -120;
    double dBUpperBound = 0.0;

    SpectrumCompressStore spectrumStore = new SpectrumCompressStore();
    private PlainLinearSpamBMP linBmp = new PlainLinearSpamBMP();
    private LogFreqSpectrogramBMP logBmp = new LogFreqSpectrogramBMP();
    private LogSegFreqSpectrogramBMP logSegBmp = new LogSegFreqSpectrogramBMP();

    private int bmpWidthDefault = 1000;
    private int bmpWidthMax = 2000;
    private int bmpWidth = bmpWidthDefault;
    private boolean bNeedRebuildLogBmp = false;

    enum LogAxisPlotMode { REPLOT, SEGMENT }
    LogAxisPlotMode logAxisMode = LogAxisPlotMode.REPLOT;

    private ScreenPhysicalMapping axisF = null;

    void init(int _nFreq, int _nTime, ScreenPhysicalMapping _axis) {
        bmpWidth = calBmpWidth(_axis);
        synchronized (this) {
            spectrumStore.init(_nFreq, _nTime);
        }
        synchronized (this) {
            linBmp.init(_nFreq, _nTime);
        }
        if (logAxisMode == LogAxisPlotMode.REPLOT) {
            synchronized (this) {
                logBmp.init(_nFreq, _nTime, _axis, bmpWidth);
            }
        } else {
            synchronized (this) {
                logSegBmp.init(_nFreq, _nTime, _axis);
            }
        }
        axisF = _axis;
    }

    void rebuildLinearBMP() {  // For state restore
        linBmp.rebuild(spectrumStore);
    }

    void rebuildAllBMP() {
        if (spectrumStore.dbShortArray.length == 0) return;
        rebuildLinearBMP();
        if (logAxisMode == LogAxisPlotMode.REPLOT) {
            logBmp.rebuild(spectrumStore, logBmp.axis);
            bNeedRebuildLogBmp = false;
        } else {
            logSegBmp.rebuild(spectrumStore, axisF);
        }
    }

    void setLogAxisMode(LogAxisPlotMode _mode) {
        if (logAxisMode != _mode) {
            if (_mode == LogAxisPlotMode.REPLOT) {
                int nFreq = logSegBmp.nFreq;
                int nTime = logSegBmp.nTime;
                logSegBmp = new LogSegFreqSpectrogramBMP();  // Release
                logBmp.init(nFreq, nTime, axisF, bmpWidth);
                logBmp.rebuild(spectrumStore, logBmp.axis);
                bNeedRebuildLogBmp = false;
            } else {
                int nFreq = logBmp.nFreq;
                int nTime = logBmp.nTime;
                logBmp = new LogFreqSpectrogramBMP();  // Release
                logSegBmp.init(nFreq, nTime, axisF);
                logSegBmp.rebuild(spectrumStore, axisF);
            }
            logAxisMode = _mode;
        }
    }

    private int calBmpWidth(ScreenPhysicalMapping _axisFreq) {
        int tmpBmpWidth = (int) _axisFreq.nCanvasPixel;
        if (tmpBmpWidth <= 1) tmpBmpWidth = bmpWidthDefault;
        if (tmpBmpWidth > 2000) tmpBmpWidth = bmpWidthMax;
        return tmpBmpWidth;
    }

    void updateAxis(ScreenPhysicalMapping _axisFreq) {
        if (_axisFreq.mapType == ScreenPhysicalMapping.Type.LINEAR) {
            return;  // a linear axis, do not update
        }
        if (logAxisMode == LogAxisPlotMode.REPLOT) {
            synchronized (this) {
                bmpWidth = calBmpWidth(_axisFreq);
                logBmp.init(logBmp.nFreq, logBmp.nTime, _axisFreq, bmpWidth);
                bNeedRebuildLogBmp = true;
            }
        } else {
            synchronized (this) {
                logSegBmp.init(logSegBmp.nFreq, logSegBmp.nTime, _axisFreq);
            }
        }
        axisF = _axisFreq;
    }
```

```
    void updateZoom() {
        if (logAxisMode == LogAxisPlotMode.REPLOT) {
            bNeedRebuildLogBmp = true;
        }
    }

    void setColorMap(String colorMapName) {
        cma = ColorMapArray.selectColorMap(colorMapName);
        // Refresh if we have spectrogram data
        rebuildAllBMP();
    }

    // return value between 0 .. nLevel - 1
    private static int levelFromDB(double d, double lowerBound, double upperBound, int nLevel) {
        if (d >= upperBound) {
            return 0;
        }
        if (d <= lowerBound || Double.isInfinite(d) || Double.isNaN(d)) {
            return nLevel - 1;
        }
        return (int)(nLevel * (upperBound - d) / (upperBound - lowerBound));
    }

    private int colorFromDBLevel(short d) {  // 0 <= d <= 32767
        return colorFromDB(AnalyzerGraphic.maxDB - (AnalyzerGraphic.maxDB -
AnalyzerGraphic.minDB) / 32768.0 * d);
//        return cma[(int)(cma.length / 32768.0 * d)];
    }

    private int colorFromDB(double d) {
        // Assume in ARGB format. Always set alpha=0xff for drawBitmap to work correctly.
        return cma[levelFromDB(d, dBLowerBound, dBUpperBound, cma.length)] + 0xff000000;
    }

    void fill(double[] db) {
        synchronized (this) {
            spectrumStore.fill(db);
            linBmp.fill(db);
            if (logAxisMode == LogAxisPlotMode.REPLOT) {
                logBmp.fill(db);
            } else {
                logSegBmp.fill(db);
            }
        }
    }

    void draw(Canvas c, ScreenPhysicalMapping.Type freqAxisType,
SpectrogramPlot.TimeAxisMode showModeSpectrogram,
              Paint smoothBmpPaint, Paint cursorTimePaint) {
        // drawBitmap(int[] ...) was deprecated in API level 21.
        // public void drawBitmap (int[] colors, int offset, int stride, float x, float y,
        //                         int width, int height, boolean hasAlpha, Paint paint)
        // http://developer.android.com/reference/android/graphics/Canvas.html#drawBitmap(int[],
int, int, float, float, int, int, boolean, android.graphics.Paint)
        // Consider use Bitmap
        // http://developer.android.com/reference/android/graphics/Bitmap.html#setPixels(int[], int,
int, int, int, int, int)

        int pt;
        int lineLen;

        if (freqAxisType == ScreenPhysicalMapping.Type.LOG) {
            // Reference answer
//            c.save();
//            c.scale(1, 0.5f);
//            logBmp.draw(c);
//            if (showModeSpectrogram == TimeAxisMode.OVERWRITE) {
//                c.drawLine(0, logBmp.bmPt, logBmp.nFreq, logBmp.bmPt, cursorTimePaint);
//            }
//            c.restore();

//            c.save();
//            c.translate(0, nTimePoints/2);
//            c.scale((float)nFreqPoints / logSegBmp.bmpWidth, 0.5f);
//            logSegBmp.draw(c);
//            if (showModeSpectrogram == TimeAxisMode.OVERWRITE) {
//                c.drawLine(0, logSegBmp.bmPt, logSegBmp.bmpWidth, logSegBmp.bmPt,
cursorTimePaint);
//            }
//            c.restore();

            synchronized (this) {
                if (logAxisMode == LogAxisPlotMode.REPLOT) {
                    // Draw in log, method: draw by axis
                    if (bNeedRebuildLogBmp) {
                        logBmp.rebuild(spectrumStore, axisF);
                        bNeedRebuildLogBmp = false;
                    }
                    logBmp.draw(c, showModeSpectrogram, smoothBmpPaint);
                    pt = logBmp.bmPt;
                    lineLen = logBmp.bmpWidth;
                } else {
                    // Draw in log, method: segmentation log freq BMP
                    c.scale((float) logSegBmp.nFreq / logSegBmp.bmpWidth, 1.0f);
                    logSegBmp.draw(c, showModeSpectrogram, axisF, smoothBmpPaint);
                    pt = logSegBmp.bmPt;
                    lineLen = logSegBmp.bmpWidth;
                }
            }
        } else {
            synchronized (this) {
                linBmp.draw(c, showModeSpectrogram, smoothBmpPaint);
            }
            pt = linBmp.iTimePointer;
            lineLen = linBmp.nFreq;
        }

        // new data line
        if (showModeSpectrogram == SpectrogramPlot.TimeAxisMode.OVERWRITE) {
            c.drawLine(0, pt, lineLen, pt, cursorTimePaint);
        }
    }

    // Save spectrum in a lower resolution short[] (0~32767) instead of double[]
    class SpectrumCompressStore {
        private final static String TAG = "SpectrumCompressStore:";
        int nFreq;
        int nTime;
        int iTimePointer;
        short[] dbShortArray = new short[0];  // java don't have unsigned short

        void init(int _nFreq, int _nTime) {
            // _nFreq == 2^n
            if (dbShortArray.length != (_nFreq + 1) * _nTime) {
                dbShortArray = new short[(_nFreq + 1) * _nTime];
            }
            if (nFreq != _nFreq || nTime != _nTime) {
                clear();
            }
            nFreq = _nFreq;
            nTime = _nTime;
        }
```

```
        void clear() {
            Arrays.fill(dbShortArray, (short) 32767);
            iTimePointer = 0;
        }

        void fill(double[] db) {
            if (db.length - 1 != nFreq) {
                Log.e(TAG, "fill(): WTF");
                return;
            }
            int p0 = (nFreq + 1) * iTimePointer;
            for (int i = 0; i <= nFreq; i++) {
                dbShortArray[p0 + i] = (short) levelFromDB(db[i], AnalyzerGraphic.minDB,
AnalyzerGraphic.maxDB, 32768);
            }
            iTimePointer++;
            if (iTimePointer >= nTime) iTimePointer = 0;
        }
    }

    private class PlainLinearSpamBMP {
        private final static String TAG = "PlainLinearSpamBMP:";
        private int nFreq;
        private int nTime;

        int[] spectrogramColors = new int[0];  // int:ARGB, nFreqPoints columns, nTimePoints rows
        int[] spectrogramColorsShifting;       // temporarily of spectrogramColors for shifting mode
        int iTimePointer;          // pointer to the row to be filled (row major)

        void init(int _nFreq, int _nTime) {
            boolean bNeedClean = nFreq != _nFreq;
            if (spectrogramColors.length != _nFreq * _nTime) {
                spectrogramColors = new int[_nFreq * _nTime];
                spectrogramColorsShifting = new int[_nFreq * _nTime];
                bNeedClean = true;
            }
            if (!bNeedClean && iTimePointer >= _nTime) {
                Log.w(TAG, "setupSpectrogram(): Should not happen!!");
                Log.i(TAG, "setupSpectrogram(): iTimePointer=" + iTimePointer + "  nFreqPoints=" +
_nFreq + "  nTimePoints=" + _nTime);
                bNeedClean = true;
            }
            if (bNeedClean) {
                clear();
            }
            nFreq = _nFreq;
            nTime = _nTime;
        }

        void clear() {
            Arrays.fill(spectrogramColors, 0);
            iTimePointer = 0;
        }

        void fill(double[] db) {
            if (db.length - 1 != nFreq) {
                Log.e(TAG, "fill(): WTF");
                return;
            }
            int pRef = iTimePointer * nFreq - 1;
            for (int i = 1; i < db.length; i++) {  // no DC term
                spectrogramColors[pRef + i] = colorFromDB(db[i]);
            }
            iTimePointer++;
            if (iTimePointer >= nTime) iTimePointer = 0;
        }

        double[] dbTmp = new double[0];

        void rebuild(SpectrumCompressStore dbLevelPic) {
            nFreq = dbLevelPic.nFreq;
            nTime = dbLevelPic.nTime;
            init(nFreq, nTime);  // reallocate

            if (dbTmp.length != nFreq + 1) {
                dbTmp = new double[nFreq + 1];
            }
            iTimePointer = 0;
            for (int k = 0; k < nTime; k++) {
                int p0 = (nFreq + 1) * k;
                for (int i = 0; i <= nFreq; i++) {  // See colorFromDBLevel
                    dbTmp[i] = AnalyzerGraphic.maxDB - (AnalyzerGraphic.maxDB -
AnalyzerGraphic.minDB) / 32768.0 * dbLevelPic.dbShortArray[p0 + i];
                }
                fill(dbTmp);
            }
            iTimePointer = dbLevelPic.iTimePointer;
        }

        void draw(Canvas c, SpectrogramPlot.TimeAxisMode showModeSpectrogram, Paint
smoothBmpPaint) {
            if (spectrogramColors.length == 0) return;
            if (showModeSpectrogram == SpectrogramPlot.TimeAxisMode.SHIFT) {
                System.arraycopy(spectrogramColors, 0, spectrogramColorsShifting,
                        (nTime - iTimePointer) * nFreq, iTimePointer * nFreq);
                System.arraycopy(spectrogramColors, iTimePointer * nFreq,
spectrogramColorsShifting,
                        0, (nTime - iTimePointer) * nFreq);
                c.drawBitmap(spectrogramColorsShifting, 0, nFreq, 0, 0,
                        nFreq, nTime, false, smoothBmpPaint);
            } else {
                c.drawBitmap(spectrogramColors, 0, nFreq, 0, 0,
                        nFreq, nTime, false, smoothBmpPaint);
            }
        }
    }

    // Actually capable to show both Linear and Log spectrogram
    private class LogFreqSpectrogramBMP {
        final static String TAG = "LogFreqSpectrogramBMP:";
        int nFreq = 0;
        int nTime = 0;
        int[] bm = new int[0];   // elements are in "time major" order.
        int[] bmShiftCache = new int[0];
        int bmPt = 0;
        int bmpWidth = 1000;     // width in Frequency direction
        int[] mapFreqToPixL = new int[0];  // map that a frequency point should map to bm[]
        int[] mapFreqToPixH = new int[0];
        ScreenPhysicalMapping axis = null;

        LogFreqSpectrogramBMP() {
        }

        // like setupSpectrogram()
        void init(int _nFreq, int _nTime, ScreenPhysicalMapping _axis, int _bmpWidth) {
            // _nFreq == 2^n
            if (bm.length != _bmpWidth * _nTime) {
                bm = new int[_bmpWidth * _nTime];
                bmShiftCache = new int[bm.length];
            }
            if (mapFreqToPixL.length != _nFreq + 1) {
                Log.d(TAG, "init(): New");
                mapFreqToPixL = new int[_nFreq + 1];
                mapFreqToPixH = new int[_nFreq + 1];
            }
            if (bmpWidth != _bmpWidth || nTime != _nTime) {
                clear();
            }  // else only update axis
            bmpWidth = _bmpWidth;
            nFreq = _nFreq;
            nTime = _nTime;
            if (_axis == null) {
                Log.e(TAG, "init(): damn: axis == null");
                return;
            }
            if (axis != _axis) {  // not itself
                axis = new ScreenPhysicalMapping(_axis);
            }
            if (_axis.vLowerBound > _axis.vUpperBound) {  // ensure axis.vLowerBound <
axis.vUpperBound
                axis.reverseBounds();
            }
            double dFreq = axis.vUpperBound / nFreq;
//            Log.v(TAG, "init(): axis.vL=" + axis.vLowerBound + "  axis.vU=" + axis.vUpperBound +
"  axis.nC=" + axis.nCanvasPixel);
            for (int i = 0; i <= nFreq; i++) {  // freq = i * dFreq
                // do not show DC component (xxx - 1).
                mapFreqToPixL[i] = (int) Math.floor(axis.pixelFromV((i - 0.5) * dFreq) /
axis.nCanvasPixel * bmpWidth);
                mapFreqToPixH[i] = (int) Math.floor(axis.pixelFromV((i + 0.5) * dFreq) /
axis.nCanvasPixel * bmpWidth);
                if (mapFreqToPixH[i] >= bmpWidth) mapFreqToPixH[i] = bmpWidth - 1;
                if (mapFreqToPixH[i] < 0) mapFreqToPixH[i] = 0;
                if (mapFreqToPixL[i] >= bmpWidth) mapFreqToPixL[i] = bmpWidth - 1;
                if (mapFreqToPixL[i] < 0) mapFreqToPixL[i] = 0;
//                Log.i(TAG, "init(): [" + i + "]  L = " + axis.pixelNoZoomFromV((i-0.5f)*dFreq) + "  H = "
+ axis.pixelNoZoomFromV((i+0.5f)*dFreq));
            }
        }

        void clear() {
            Arrays.fill(bm, 0);
            bmPt = 0;
        }
```

```
void fill(double[] db) {
    if (db.length - 1 != nFreq) {
        Log.e(TAG, "fill(): WTF");
        return;
    }
    int bmP0 = bmPt * bmpWidth;
    double maxDB;
    int i = 1;  // skip DC component(i = 0).
    while (i <= nFreq) {
        maxDB = db[i];
        int j = i + 1;
        while (j <= nFreq && mapFreqToPixL[i] + 1 == mapFreqToPixH[j]) {
            // If multiple frequency points map to one pixel, show only the maximum.
            if (db[j] > maxDB) maxDB = db[j];
            j++;
        }
        int c = colorFromDB(maxDB);
        for (int k = mapFreqToPixL[i]; k < mapFreqToPixH[i]; k++) {
            bm[bmP0 + k] = c;
        }
        i = j;
    }
    bmPt++;
    if (bmPt >= nTime) bmPt = 0;
}

private void fill(short[] dbLevel) {
    if (dbLevel.length - 1 != nFreq) {
        Log.e(TAG, "fill(): WTF");
        return;
    }
    int bmP0 = bmPt * bmpWidth;
    short maxDBLevel;
    int i = 1;  // skip DC component(i = 0).
    while (i <= nFreq) {
        maxDBLevel = dbLevel[i];
        int j = i + 1;
        while (j <= nFreq && mapFreqToPixL[i] + 1 == mapFreqToPixH[j]) {
            // If multiple frequency points map to one pixel, show only the maximum.
            if (dbLevel[j] < maxDBLevel) maxDBLevel = dbLevel[j];
            j++;
        }
        int c = colorFromDBLevel(maxDBLevel);
        for (int k = mapFreqToPixL[i]; k < mapFreqToPixH[i]; k++) {
```

```
                    bm[bmP0 + k] = c;
                }
                i = j;
            }
            bmPt++;
            if (bmPt >= nTime) bmPt = 0;
        }

        short[] dbLTmp = new short[0];

        // re-calculate whole spectrogram, according to dbLevelPic under axis _axis
        void rebuild(SpectrumCompressStore dbLevelPic, ScreenPhysicalMapping _axisF) {
            nFreq = dbLevelPic.nFreq;
            nTime = dbLevelPic.nTime;
            init(nFreq, nTime, _axisF, bmpWidth);  // reallocate and rebuild index

            if (dbLTmp.length != nFreq + 1) {
                dbLTmp = new short[nFreq + 1];
            }
            bmPt = 0;
            for (int k = 0; k < nTime; k++) {
                System.arraycopy(dbLevelPic.dbShortArray, (nFreq + 1) * k, dbLTmp, 0, (nFreq + 1));
                fill(dbLTmp);
            }
            bmPt = dbLevelPic.iTimePointer;
        }

        // draw to a regime size nFreq * nTime
        void draw(Canvas c, SpectrogramPlot.TimeAxisMode showModeSpectrogram, Paint
smoothBmpPaint) {
            if (bm.length == 0) return;
            c.scale((float) nFreq / bmpWidth, 1f);
            if (showModeSpectrogram == SpectrogramPlot.TimeAxisMode.SHIFT) {
                System.arraycopy(bm, 0, bmShiftCache, (nTime - bmPt) * bmpWidth, bmPt *
bmpWidth);
                System.arraycopy(bm, bmPt * bmpWidth, bmShiftCache, 0, (nTime - bmPt) *
bmpWidth);
                c.drawBitmap(bmShiftCache, 0, bmpWidth, 0.0f, 0.0f,
                        bmpWidth, nTime, false, smoothBmpPaint);
            } else {
                c.drawBitmap(bm, 0, bmpWidth, 0.0f, 0.0f,
                        bmpWidth, nTime, false, smoothBmpPaint);
            }
        }
```

```
    }

    private class LogSegFreqSpectrogramBMP {
        final static String TAG = "LogSeg..:";
        int nFreq = 0;
        int nTime = 0;
        int[] bm = new int[0];   // elements are in "time major" order.
        int[] bmShiftCache = new int[0];
        int bmPt = 0;
        double[] iFreqToPix = new double[0];
        double[] pixelAbscissa = new double[0];
        double[] freqAbscissa = new double[0];
        int bmpWidth = 0;
        final double incFactor = 2;
        double interpolationFactor = 2.0; // the extra factor (1.0~2.0) here is for sharper image.

        void init(int _nFreq, int _nTime, ScreenPhysicalMapping _axis) {
            if (_nFreq == 0 || _nTime == 0 || Math.max(_axis.vLowerBound, _axis.vUpperBound) ==
0) {
                return;
            }
            // Note that there is limit for bmpWidth, i.e. Canvas.getMaximumBitmapHeight
            //
https://developer.android.com/reference/android/graphics/Canvas.html#getMaximumBitmapHei
ght%28%29
            // Seems that this limit is at about 4096.
            // In case that this limit is reached, we might break down pixelAbscissa[] to smaller
pieces.
            bmpWidth = (int) (_nFreq * incFactor * interpolationFactor);
            if (bm.length != bmpWidth * _nTime) {
                bm = new int[bmpWidth * _nTime];
                bmShiftCache = new int[bm.length];
            }
            if (nFreq != _nFreq || nTime != _nTime) {
                clear();
            }  // else only update axis and mapping
            nFreq = _nFreq;
            nTime = _nTime;

            double maxFreq = Math.max(_axis.vLowerBound, _axis.vUpperBound);
            double minFreq = maxFreq / nFreq;
            double dFreq = maxFreq / nFreq;

            int nSegment = (int) (Math.log((maxFreq + 0.1) / minFreq) / Math.log(incFactor)) + 1;
            Log.d(TAG, "nFreq = " + nFreq + "  dFreq = " + dFreq + "  nSegment = " + nSegment + "
bmpWidth = " + bmpWidth);

            pixelAbscissa = new double[nSegment + 1];
            freqAbscissa = new double[nSegment + 1];
            pixelAbscissa[0] = 0;
            freqAbscissa[0] = minFreq;  // should be minFreq
            //Log.v(TAG, "pixelAbscissa[" + 0 + "] = " + pixelAbscissa[0] + "  freqAbscissa[i] = " +
freqAbscissa[0]);
            Log.v(TAG, "pixelAbscissa[" + 0 + "] = " + pixelAbscissa[0]);
            for (int i = 1; i <= nSegment; i++) {
                /**  Mapping [0, 1] -> [fmin, fmax]
                 *   /  fmax  \ x
                 *   | ------ |   * fmin
                 *   \  fmin  /
                 *   This makes the "pixels"(x) more uniformly map to frequency points in logarithmic
scale.
                 */
                pixelAbscissa[i] = (pow(maxFreq / minFreq, (double) i / nSegment) * minFreq -
minFreq) / (maxFreq - minFreq);
                pixelAbscissa[i] = Math.floor(pixelAbscissa[i] * bmpWidth);   // align to pixel boundary
                freqAbscissa[i] = pixelAbscissa[i] / bmpWidth * (maxFreq - minFreq) + minFreq;
                Log.v(TAG, "pixelAbscissa[" + i + "] = " + pixelAbscissa[i] + "  freqAbscissa[i] = " +
freqAbscissa[i]);
            }

            // Map between [pixelAbscissa[i-1]..pixelAbscissa[i]] and [freqAbscissa[i-
1]..freqAbscissa[i]]
            iFreqToPix = new double[nFreq + 1];
            iFreqToPix[0] = 0;
            double eps = 1e-7;  // 7 + log10(8192) < 15
            int iF = 1;
            ScreenPhysicalMapping axisSeg = new ScreenPhysicalMapping(1.0, minFreq,
maxFreq, ScreenPhysicalMapping.Type.LOG);
            for (int i = 1; i <= nSegment; i++) {
                axisSeg.setNCanvasPixel(Math.round(pixelAbscissa[i] - pixelAbscissa[i - 1]));  //
should work without round()
                axisSeg.setBounds(freqAbscissa[i - 1], freqAbscissa[i]);
                Log.v(TAG, "axisSeg[" + i + "] .nC = " + axisSeg.nCanvasPixel + "  .vL = " +
axisSeg.vLowerBound + "  .vU = " + axisSeg.vUpperBound);
                while ((iF + 0.5) * dFreq <= freqAbscissa[i] + eps) {
                   // upper bound of the pixel position of frequency point iF
                   iFreqToPix[iF] = axisSeg.pixelFromV((iF + 0.5) * dFreq) + pixelAbscissa[i - 1];
//                Log.d(TAG, "seg = " + i + "  iFreqToPix[" + iF + "] = " + iFreqToPix[iF]);
```

```
                    iF++;
                }
            }
            if (iF < nFreq) {  // last point
                iFreqToPix[nFreq] = pixelAbscissa[nSegment];
            }
        }

        void clear() {
            Arrays.fill(bm, 0);
            bmPt = 0;
        }

        double[] dbPixelMix = new double[0];

        void fill(double[] db) {
            if (db.length - 1 != nFreq) {
                Log.e(TAG, "full(): WTF");
                return;
            }
            if (dbPixelMix.length != bmpWidth) {
                dbPixelMix = new double[bmpWidth];
            }
            Arrays.fill(dbPixelMix, 0.0);
            double b0 = iFreqToPix[0];
            for (int i = 1; i <= nFreq; i++) {
                // assign color to pixel iFreqToPix[i-1] .. iFreqToPix[i]
                double b1 = iFreqToPix[i];
                if ((int) b0 == (int) b1) {
                    dbPixelMix[(int) b0] += db[i] * (b1 - b0);
                    continue;
                }
                if (b0 % 1 != 0) {  // mix color
                    //dbPixelMix[(int)b0] += db[i] * (1 - b0 % 1);  // dB mean
                    double db0 = db[i - 1];  // i should > 1
                    double db1 = db[i];
                    dbPixelMix[(int) b0] = 10 * log10(pow(10, db0 / 10) * (b0 % 1) + pow(10, db1 / 10) *
(1 - b0 % 1));  // energy mean
                }
                for (int j = (int) Math.ceil(b0); j < (int) b1; j++) {
                    dbPixelMix[j] = db[i];
                }
//                if (b1 % 1 > 0) {  // avoid out of bound (b1 == bmpWidth)
//                    dbPixelMix[(int) b1] += db[i] * (b1 % 1);
//                  }
                b0 = b1;
            }
            int bmP0 = bmPt * bmpWidth;
            for (int i = 0; i < bmpWidth; i++) {
                bm[bmP0 + i] = colorFromDB(dbPixelMix[i]);
            }
            bmPt++;
            if (bmPt >= nTime) bmPt = 0;
        }

        double[] dbTmp = new double[0];

        void rebuild(SpectrumCompressStore dbLevelPic, ScreenPhysicalMapping _axisF) {
            nFreq = dbLevelPic.nFreq;
            nTime = dbLevelPic.nTime;
            init(nFreq, nTime, _axisF);  // reallocate and rebuild index

            if (dbTmp.length != nFreq + 1) {
                dbTmp = new double[nFreq + 1];
            }
            bmPt = 0;
            for (int k = 0; k < nTime; k++) {
                int p0 = (nFreq + 1) * k;
                for (int i = 0; i <= nFreq; i++) {  // See colorFromDBLevel
                    dbTmp[i] = AnalyzerGraphic.maxDB - (AnalyzerGraphic.maxDB -
AnalyzerGraphic.minDB) / 32768.0 * dbLevelPic.dbShortArray[p0 + i];
                }
                fill(dbTmp);
            }
            bmPt = dbLevelPic.iTimePointer;
        }

        String st1old;  // for debug
        String st2old;  // for debug

        void draw(Canvas c, SpectrogramPlot.TimeAxisMode showModeSpectrogram,
ScreenPhysicalMapping axisFreq, Paint smoothBmpPaint) {
            if (bm.length == 0 || axisFreq.nCanvasPixel == 0) {
                Log.d(TAG, "draw(): what.....");
                return;
            }
            int i1 = pixelAbscissa.length - 1;
            String st1 = "draw():  pixelAbscissa[" + (i1 - 1) + "]=" + pixelAbscissa[i1 - 1] + "  
pixelAbscissa[" + i1 + "]=" + pixelAbscissa[i1] + "  bmpWidth=" + bmpWidth;
            String st2 = "draw():  axis.vL=" + axisFreq.vLowerBound + "  axis.vU=" + 
axisFreq.vUpperBound + "  axisFreq.nC=" + axisFreq.nCanvasPixel + "  nTime=" + nTime;
            if (!st1.equals(st1old)) {
                Log.v(TAG, st1);
                Log.v(TAG, st2);
                st1old = st1;
                st2old = st2;
            }
            int[] bmTmp = bm;
            if (showModeSpectrogram == SpectrogramPlot.TimeAxisMode.SHIFT) {
                System.arraycopy(bm, 0, bmShiftCache, (nTime - bmPt) * bmpWidth, bmPt * 
bmpWidth);
                System.arraycopy(bm, bmPt * bmpWidth, bmShiftCache, 0, (nTime - bmPt) * 
bmpWidth);
                bmTmp = bmShiftCache;
            }
            for (int i = 1; i < pixelAbscissa.length; i++) {  // draw each segmentation
                c.save();
                double f1 = freqAbscissa[i - 1];
                double f2 = freqAbscissa[i];
                double p1 = axisFreq.pixelNoZoomFromV(f1);
                double p2 = axisFreq.pixelNoZoomFromV(f2);
                if (axisFreq.vLowerBound > axisFreq.vUpperBound) {
                    p1 = axisFreq.nCanvasPixel - p1;
                    p2 = axisFreq.nCanvasPixel - p2;
                }
                double widthFactor = (p2 - p1) / (pixelAbscissa[i] - pixelAbscissa[i - 1]) * (bmpWidth / 
axisFreq.nCanvasPixel);
                // Log.v(TAG, "draw():  f1=" + f1 + "  f2=" + f2 + "  p1=" + p1 + "  p2=" + p2 + "  
widthFactor=" + widthFactor + "  modeInt=" + axisFreq.mapType);
                c.scale((float) widthFactor, 1);
                c.drawBitmap(bmTmp, (int) pixelAbscissa[i - 1], bmpWidth, (float)(p1 / 
axisFreq.nCanvasPixel * bmpWidth / widthFactor), 0.0f,
                        (int) (pixelAbscissa[i] - pixelAbscissa[i - 1]), nTime, false, smoothBmpPaint);
                c.restore();
            }
        }
    }
}
```

SpectrogramPlot

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.util.Log;

/**
 * The spectrogram plot part of AnalyzerGraphic
 */

class SpectrogramPlot {
    private static final String TAG = "SpectrogramPlot:";
    boolean showFreqAlongX = false;

    enum TimeAxisMode {  // java's enum type is inconvenient
        SHIFT(0), OVERWRITE(1);       // 0: moving (shifting) spectrogram, 1: overwriting in loop

        private final int value;
        TimeAxisMode(int value) { this.value = value; }
        public int getValue() { return value; }
    }

    private TimeAxisMode showModeSpectrogram = TimeAxisMode.OVERWRITE;
    private boolean bShowTimeAxis = true;
```

```
  private double timeWatch = 4.0;  // TODO: a bit duplicated, use axisTime
  private volatile int timeMultiplier = 1;  // should be accorded with nFFTAverage in
AnalyzerActivity
  int nFreqPoints;  // TODO: a bit duplicated, use BMP.nFreq
  int nTimePoints;  // TODO: a bit duplicated, use BMP.nTime
  double timeInc;

  private Matrix matrixSpectrogram = new Matrix();
  private Paint smoothBmpPaint;
  private Paint backgroundPaint;
  private Paint cursorPaint;
  private Paint gridPaint, rulerBrightPaint;
  private Paint labelPaint;
  private Paint cursorTimePaint;

  ScreenPhysicalMapping axisFreq;
  ScreenPhysicalMapping axisTime;
  private GridLabel fqGridLabel;
  private GridLabel tmGridLabel;
  double cursorFreq;

  private float DPRatio;
  private float gridDensity = 1/85f;  // every 85 pixel one grid line, on average
  private int canvasHeight=0, canvasWidth=0;
  float labelBeginX, labelBeginY;

  SpectrogramPlot(Context _context) {
    DPRatio = _context.getResources().getDisplayMetrics().density;

    gridPaint = new Paint();
    gridPaint.setColor(Color.DKGRAY);
    gridPaint.setStyle(Paint.Style.STROKE);
    gridPaint.setStrokeWidth(0.6f * DPRatio);

    cursorPaint = new Paint(gridPaint);
    cursorPaint.setColor(Color.parseColor("#00CD00"));

    cursorTimePaint = new Paint(cursorPaint);
    cursorTimePaint.setStyle(Paint.Style.STROKE);
    cursorTimePaint.setStrokeWidth(0);

    rulerBrightPaint = new Paint();
```

```
        rulerBrightPaint.setColor(Color.rgb(99, 99, 99));  // 99: between Color.DKGRAY and
Color.GRAY
        rulerBrightPaint.setStyle(Paint.Style.STROKE);
        rulerBrightPaint.setStrokeWidth(1);

        labelPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
        labelPaint.setColor(Color.GRAY);
        labelPaint.setTextSize(14.0f * DPRatio);
        labelPaint.setTypeface(Typeface.MONOSPACE);  // or Typeface.SANS_SERIF

        backgroundPaint = new Paint();
        backgroundPaint.setColor(Color.BLACK);

        cursorFreq = 0f;

        fqGridLabel = new GridLabel(GridLabel.Type.FREQ, canvasWidth  * gridDensity /
DPRatio);
        tmGridLabel = new GridLabel(GridLabel.Type.TIME, canvasHeight * gridDensity /
DPRatio);

        axisFreq = new ScreenPhysicalMapping(0, 0, 0, ScreenPhysicalMapping.Type.LINEAR);
        axisTime = new ScreenPhysicalMapping(0, 0, 0, ScreenPhysicalMapping.Type.LINEAR);

        Log.i(TAG, "SpectrogramPlot() initialized");
    }

    // Before calling this, axes should be initialized.
    void setupSpectrogram(AnalyzerParameters analyzerParam) {
        int sampleRate       = analyzerParam.sampleRate;
        int fftLen           = analyzerParam.fftLen;
        int hopLen           = analyzerParam.hopLen;
        int nAve             = analyzerParam.nFFTAverage;
        double timeDurationE = analyzerParam.spectrogramDuration;

        timeWatch = timeDurationE;
        timeMultiplier = nAve;
        timeInc     = (double)hopLen / sampleRate;  // time of each slice
        nFreqPoints = fftLen / 2;           // no direct current term
        nTimePoints = (int)Math.ceil(timeWatch / timeInc);
        spectrogramBMP.init(nFreqPoints, nTimePoints, axisFreq);
        Log.i(TAG, "setupSpectrogram() done" +
                "\n  sampleRate    = " + sampleRate +
                "\n  fftLen        = " + fftLen +
                "\n  timeDurationE = " + timeDurationE + " * " + nAve + "  (" + nTimePoints + " points)"
+
                "\n  canvas size freq= " + axisFreq.nCanvasPixel + " time=" +
axisTime.nCanvasPixel);
    }

    void setCanvas(int _canvasWidth, int _canvasHeight, double[] axisBounds) {
        Log.i(TAG, "setCanvas(): " + _canvasWidth + " x " + _canvasHeight);
        canvasWidth  = _canvasWidth;
        canvasHeight = _canvasHeight;
        if (canvasHeight > 1 && canvasWidth > 1) {
            updateDrawingWindowSize();
        }
        if (axisBounds != null) {
            if (showFreqAlongX) {
                axisFreq.setBounds(axisBounds[0], axisBounds[2]);
                axisTime.setBounds(axisBounds[1], axisBounds[3]);
            } else {
                axisTime.setBounds(axisBounds[0], axisBounds[2]);
                axisFreq.setBounds(axisBounds[1], axisBounds[3]);
            }
            if (showModeSpectrogram == TimeAxisMode.SHIFT) {
                double b1 = axisTime.vLowerBound;
                double b2 = axisTime.vUpperBound;
                axisTime.setBounds(b2, b1);
            }
        }
        fqGridLabel.setDensity(axisFreq.nCanvasPixel * gridDensity / DPRatio);
        tmGridLabel.setDensity(axisTime.nCanvasPixel * gridDensity / DPRatio);

        spectrogramBMP.updateAxis(axisFreq);
    }

    void setZooms(double xZoom, double xShift, double yZoom, double yShift) {
        //Log.i(TAG, "setZooms():  xZ=" + xZoom + "  xS=" + xShift + "  yZ=" + yZoom + "  yS=" +
yShift);
        if (showFreqAlongX) {
            axisFreq.setZoomShift(xZoom, xShift);
            axisTime.setZoomShift(yZoom, yShift);
        } else {
            axisFreq.setZoomShift(yZoom, yShift);
            axisTime.setZoomShift(xZoom, xShift);
        }
        spectrogramBMP.updateZoom();
    }

    // Linear or Logarithmic frequency axis
    void setFreqAxisMode(ScreenPhysicalMapping.Type mapType, double
freq_lower_bound_for_log, GridLabel.Type gridType) {
        Log.i(TAG, "setFreqAxisMode(): set to mode " + mapType);
        axisFreq.setMappingType(mapType, freq_lower_bound_for_log);
        fqGridLabel.setGridType(gridType);
        spectrogramBMP.updateAxis(axisFreq);
    }

    void setColorMap(String colorMapName) { spectrogramBMP.setColorMap(colorMapName); }

    void setSpectrogramDBLowerBound(double b) { spectrogramBMP.dBLowerBound = b; }

    void setCursor(double x, double y) {
        if (showFreqAlongX) {
            //cursorFreq = axisBounds.width() * (xShift + (x-labelBeginX)/(canvasWidth-
labelBeginX)/xZoom);  // frequency
            cursorFreq = axisFreq.vFromPixel(x - labelBeginX);
        } else {
            //cursorFreq = axisBounds.width() * (1 - yShift - y/labelBeginY/yZoom);  // frequency
            cursorFreq = axisFreq.vFromPixel(y);
        }
        if (cursorFreq < 0) {
            cursorFreq = 0;
        }
    }

    TimeAxisMode getSpectrogramMode() {
        return showModeSpectrogram;
    }

    double getCursorFreq() {
        return canvasWidth == 0 ? 0 : cursorFreq;
    }

    void hideCursor() {
        cursorFreq = 0;
    }

    void setTimeMultiplier(int nAve) {
        timeMultiplier = nAve;
        if (showModeSpectrogram == TimeAxisMode.SHIFT) {
```

```
        axisTime.vLowerBound = timeWatch * timeMultiplier;
    } else {
        axisTime.vUpperBound = timeWatch * timeMultiplier;
    }
    // keep zoom shift
    axisTime.setZoomShift(axisTime.getZoom(), axisTime.getShift());
}

void setShowTimeAxis(boolean bSTA) {
    bShowTimeAxis = bSTA;
}

void setSpectrogramModeShifting(boolean b) {
    if ((showModeSpectrogram == TimeAxisMode.SHIFT) != b) {
        // mode change, swap time bounds.
        double b1 = axisTime.vLowerBound;
        double b2 = axisTime.vUpperBound;
        axisTime.setBounds(b2, b1);
    }
    if (b) {
        showModeSpectrogram = TimeAxisMode.SHIFT;
        setPause(isPaused);  // update time estimation
    } else {
        showModeSpectrogram = TimeAxisMode.OVERWRITE;
    }
}

void setShowFreqAlongX(boolean b) {
    if (showFreqAlongX != b) {
        // Set (swap) canvas size
        double t = axisFreq.nCanvasPixel;
        axisFreq.setNCanvasPixel(axisTime.nCanvasPixel);
        axisTime.setNCanvasPixel(t);
        // swap bounds of freq axis
        axisFreq.reverseBounds();

        fqGridLabel.setDensity(axisFreq.nCanvasPixel * gridDensity / DPRatio);
        tmGridLabel.setDensity(axisTime.nCanvasPixel * gridDensity / DPRatio);
    }
    showFreqAlongX = b;
}

void setSmoothRender(boolean b) {
    if (b) {
        smoothBmpPaint = new Paint(Paint.FILTER_BITMAP_FLAG);
    } else {
        smoothBmpPaint = null;
    }
}

private double timeLastSample = 0;
private boolean updateTimeDiff = false;

void prepare() {
    if (showModeSpectrogram == TimeAxisMode.SHIFT)
        setPause(isPaused);
}

void setPause(boolean p) {
    if (! p) {
        timeLastSample = System.currentTimeMillis()/1000.0;
    }
    isPaused = p;
}

// Will be called in another thread (SamplingLoop)
// db.length == 2^n + 1
void saveRowSpectrumAsColor(final double[] db) {
    // For time compensate in shifting mode
    double tNow = System.currentTimeMillis()/1000.0;
    updateTimeDiff = true;
    if (Math.abs(timeLastSample - tNow) > 0.5) {
        timeLastSample = tNow;
    } else {
        timeLastSample += timeInc * timeMultiplier;
        timeLastSample += (tNow - timeLastSample) * 1e-2;  // track current time
    }

    spectrogramBMP.fill(db);
}

private float getLabelBeginY() {
    float textHeigh     = labelPaint.getFontMetrics(null);
    float labelLargeLen = 0.5f * textHeigh;
    if (!showFreqAlongX && !bShowTimeAxis) {
        return canvasHeight;
    } else {
        return canvasHeight - 0.6f*labelLargeLen - textHeigh;
    }
}

// Left margin for ruler
private float getLabelBeginX() {
    float textHeight = labelPaint.getFontMetrics(null);
    float labelLaegeLen = 0.5f * textHeight;
    if (showFreqAlongX) {
        if (bShowTimeAxis) {
            int j = 3;
            for (int i = 0; i < tmGridLabel.strings.length; i++) {
                if (j < tmGridLabel.strings[i].length()) {
                    j = tmGridLabel.strings[i].length();
                }
            }
            return 0.6f*labelLaegeLen + j*0.5f*textHeight;
        } else {
            return 0;
        }
    } else {
        return 0.6f*labelLaegeLen + 2.5f*textHeight;
    }
}

private float labelBeginXOld = 0;
private float labelBeginYOld = 0;

private void updateDrawingWindowSize() {
    labelBeginX = getLabelBeginX();  // this seems will make the scaling gesture inaccurate
    labelBeginY = getLabelBeginY();
    if (labelBeginX != labelBeginXOld || labelBeginY != labelBeginYOld) {
        if (showFreqAlongX) {
            axisFreq.setNCanvasPixel(canvasWidth - labelBeginX);
            axisTime.setNCanvasPixel(labelBeginY);
        } else {
            axisTime.setNCanvasPixel(canvasWidth - labelBeginX);
            axisFreq.setNCanvasPixel(labelBeginY);
        }
        labelBeginXOld = labelBeginX;
        labelBeginYOld = labelBeginY;
    }
}

private void drawFreqCursor(Canvas c) {
        if (cursorFreq == 0) return;
        float cX, cY;
        // Show only the frequency cursor
        if (showFreqAlongX) {
            cX = (float)axisFreq.pixelFromV(cursorFreq) + labelBeginX;
            c.drawLine(cX, 0, cX, labelBeginY, cursorPaint);
        } else {
            cY = (float)axisFreq.pixelFromV(cursorFreq);
            c.drawLine(labelBeginX, cY, canvasWidth, cY, cursorPaint);
        }
    }

    // Draw time axis for spectrogram
    // Working in the original canvas frame
    private void drawTimeAxis(Canvas c, float labelBeginX, float labelBeginY, boolean
drawOnXAxis) {
        if (drawOnXAxis) {
            AxisTickLabels.draw(c, axisTime, tmGridLabel,
                    labelBeginX, labelBeginY, 0, 1,
                    labelPaint, gridPaint, rulerBrightPaint);
        } else {
            AxisTickLabels.draw(c, axisTime, tmGridLabel,
                    labelBeginX, 0, 1, -1,
                    labelPaint, gridPaint, rulerBrightPaint);
        }
    }

    // Draw frequency axis for spectrogram
    // Working in the original canvas frame
    private void drawFreqAxis(Canvas c, float labelBeginX, float labelBeginY, boolean
drawOnXAxis) {
        if (drawOnXAxis) {
            AxisTickLabels.draw(c, axisFreq, fqGridLabel,
                    labelBeginX, labelBeginY, 0, 1,
                    labelPaint, gridPaint, rulerBrightPaint);
        } else {
            AxisTickLabels.draw(c, axisFreq, fqGridLabel,
                    labelBeginX, 0, 1, -1,
                    labelPaint, gridPaint, rulerBrightPaint);
        }
    }

    private double pixelTimeCompensate = 0;
    private volatile boolean isPaused = false;
    // Plot spectrogram with axis and ticks on the whole canvas c
    void drawSpectrogramPlot(Canvas c) {
        if (canvasWidth == 0 || canvasHeight == 0) {
            return;
        }
        updateDrawingWindowSize();
        fqGridLabel.setDensity(axisFreq.nCanvasPixel * gridDensity / DPRatio);
        tmGridLabel.setDensity(axisTime.nCanvasPixel * gridDensity / DPRatio);
        fqGridLabel.updateGridLabels(axisFreq.vMinInView(), axisFreq.vMaxInView());
        tmGridLabel.updateGridLabels(axisTime.vMinInView(), axisTime.vMaxInView());

        // show Spectrogram
        double halfFreqResolutionShift;  // move the color patch to match the center frequency
        if (axisFreq.mapType == ScreenPhysicalMapping.Type.LINEAR) {
            halfFreqResolutionShift = axisFreq.getZoom() * axisFreq.nCanvasPixel / nFreqPoints /
2;
        } else {
            halfFreqResolutionShift = 0;  // the correction is included in log axis render algo.
        }
        matrixSpectrogram.reset();
        if (showFreqAlongX) {
            // when xZoom== 1: nFreqPoints -> canvasWidth; 0 -> labelBeginX
            matrixSpectrogram.postScale((float)(axisFreq.getZoom() * axisFreq.nCanvasPixel /
nFreqPoints),
                    (float)(axisTime.getZoom() * axisTime.nCanvasPixel / nTimePoints));
            matrixSpectrogram.postTranslate((float)(labelBeginX - axisFreq.getShift() *
axisFreq.getZoom() * axisFreq.nCanvasPixel + halfFreqResolutionShift),
                    (float)(-axisTime.getShift() * axisTime.getZoom() * axisTime.nCanvasPixel));
        } else {
            // postRotate() will make c.drawBitmap about 20% slower, don't know why
            matrixSpectrogram.postRotate(-90);
            matrixSpectrogram.postScale((float)(axisTime.getZoom() * axisTime.nCanvasPixel /
nTimePoints),
                    (float)(axisFreq.getZoom() * axisFreq.nCanvasPixel / nFreqPoints));
            // (1-yShift) is relative position of shift (after rotation)
            // yZoom*labelBeginY is canvas length in frequency direction in pixel unit
            matrixSpectrogram.postTranslate((float)(labelBeginX - axisTime.getShift() *
axisTime.getZoom() * axisTime.nCanvasPixel),
                    (float)((1-axisFreq.getShift()) * axisFreq.getZoom() * axisFreq.nCanvasPixel -
halfFreqResolutionShift));
        }
        c.save();
        c.concat(matrixSpectrogram);
        // Time compensate to make it smoother shifting.
        // But if user pressed pause, stop compensate.
        if (!isPaused && updateTimeDiff) {
            double timeCurrent = System.currentTimeMillis() / 1000.0;
            pixelTimeCompensate = (timeLastSample - timeCurrent) / (timeInc * timeMultiplier *
nTimePoints) * nTimePoints;
            updateTimeDiff = false;
//            Log.i(TAG, " time diff = " + (timeLastSample - timeCurrent));
        }
        if (showModeSpectrogram == TimeAxisMode.SHIFT) {
            c.translate(0.0f, (float)pixelTimeCompensate);
        }

        if (axisFreq.mapType == ScreenPhysicalMapping.Type.LOG &&
                spectrogramBMP.logAxisMode == SpectrogramBMP.LogAxisPlotMode.REPLOT) {
            // Revert the effect of axisFreq.getZoom() axisFreq.getShift() for the mode REPLOT
            c.scale((float)(1 / axisFreq.getZoom()), 1f);
            if (showFreqAlongX) {
                c.translate((float)(nFreqPoints * axisFreq.getShift() * axisFreq.getZoom()), 0.0f);
            } else {
                c.translate((float)(nFreqPoints * (1f - axisFreq.getShift() - 1f / axisFreq.getZoom()) *
axisFreq.getZoom()), 0.0f);
            }
        }

        spectrogramBMP.draw(c, axisFreq.mapType, showModeSpectrogram, smoothBmpPaint,
cursorTimePaint);
        c.restore();

        drawFreqCursor(c);

        if (showFreqAlongX) {
            c.drawRect(0, labelBeginY, canvasWidth, canvasHeight, backgroundPaint);
            drawFreqAxis(c, labelBeginX, labelBeginY, showFreqAlongX);
            if (labelBeginX > 0) {
                c.drawRect(0, 0, labelBeginX, labelBeginY, backgroundPaint);
                drawTimeAxis(c, labelBeginX, labelBeginY, !showFreqAlongX);
            }
        } else {
            c.drawRect(0, 0, labelBeginX, labelBeginY, backgroundPaint);
            drawFreqAxis(c, labelBeginX, labelBeginY, showFreqAlongX);
            if (labelBeginY != canvasHeight) {
                c.drawRect(0, labelBeginY, canvasWidth, canvasHeight, backgroundPaint);
                drawTimeAxis(c, labelBeginX, labelBeginY, !showFreqAlongX);
            }
        }
    }

    SpectrogramBMP spectrogramBMP = new SpectrogramBMP();

}
```

SpectrumPlot

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.os.SystemClock;
import android.util.Log;

import static java.lang.Math.abs;
import static java.lang.Math.ceil;
import static java.lang.Math.floor;
import static java.lang.Math.round;

/**
 * The spectrum plot part of AnalyzerGraphic
 */

class SpectrumPlot {
    private static final String TAG = "SpectrumPlot:";
    boolean showLines;
    private Paint linePaint, linePaintLight, linePeakPaint;
    private Paint cursorPaint;
    private Paint gridPaint;
    private Paint labelPaint;
    private Paint calibNamePaint;
    private Paint calibLinePaint;
    private int canvasHeight=0, canvasWidth=0;

    private GridLabel fqGridLabel;
    private GridLabel dbGridLabel;
    private float DPRatio;
    private double gridDensity = 1/85.0;  // every 85 pixel one grid line, on average

    double cursorFreq, cursorDB;  // cursor location
    ScreenPhysicalMapping axisX;  // For frequency axis
    ScreenPhysicalMapping axisY;  // For dB axis

    SpectrumPlot(Context _context) {
        DPRatio = _context.getResources().getDisplayMetrics().density;

        linePaint = new Paint();
        linePaint.setColor(Color.parseColor("#0D2C6D"));
        linePaint.setStyle(Paint.Style.STROKE);
        linePaint.setStrokeWidth(1);

        linePaintLight = new Paint(linePaint);
        linePaintLight.setColor(Color.parseColor("#3AB3E2"));

        linePeakPaint = new Paint(linePaint);
        linePeakPaint.setColor(0xFF00A0A0);

        gridPaint = new Paint();
        gridPaint.setColor(Color.DKGRAY);
        gridPaint.setStyle(Paint.Style.STROKE);
        gridPaint.setStrokeWidth(0.6f * DPRatio);

        cursorPaint = new Paint(gridPaint);
        cursorPaint.setColor(Color.parseColor("#00CD00"));
        labelPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
        labelPaint.setColor(Color.GRAY);
        labelPaint.setTextSize(14.0f * DPRatio);
        labelPaint.setTypeface(Typeface.MONOSPACE);  // or Typeface.SANS_SERIF

        calibNamePaint = new Paint(labelPaint);
        calibNamePaint.setColor(Color.YELLOW & 0x66ffffff);
        calibLinePaint = new Paint(linePaint);
        calibLinePaint.setColor(Color.YELLOW);

        cursorFreq = cursorDB = 0f;

        fqGridLabel = new GridLabel(GridLabel.Type.FREQ, canvasWidth * gridDensity /
DPRatio);
        dbGridLabel = new GridLabel(GridLabel.Type.DB,   canvasHeight * gridDensity / DPRatio);

        axisX = new ScreenPhysicalMapping(0, 0, 0, ScreenPhysicalMapping.Type.LINEAR);
        axisY = new ScreenPhysicalMapping(0, 0, 0, ScreenPhysicalMapping.Type.LINEAR);
    }

    void setCanvas(int _canvasWidth, int _canvasHeight, double[] axisBounds) {
//        Log.i("SpectrumPlot", "setCanvas: W="+_canvasWidth+"  H="+_canvasHeight);
        canvasWidth  = _canvasWidth;
        canvasHeight = _canvasHeight;
        fqGridLabel.setDensity(canvasWidth * gridDensity / DPRatio);
        dbGridLabel.setDensity(canvasHeight * gridDensity / DPRatio);
        axisX.setNCanvasPixel(canvasWidth);
        axisY.setNCanvasPixel(canvasHeight);
        if (axisBounds != null) {
            axisX.setBounds(axisBounds[0], axisBounds[2]);
            axisY.setBounds(axisBounds[1], axisBounds[3]);
        }
    }

    void setZooms(double xZoom, double xShift, double yZoom, double yShift) {
        axisX.setZoomShift(xZoom, xShift);
        axisY.setZoomShift(yZoom, yShift);
    }

    // Linear or Logarithmic frequency axis
    void setFreqAxisMode(ScreenPhysicalMapping.Type mapType, double
freq_lower_bound_for_log, GridLabel.Type gridType) {
        axisX.setMappingType(mapType, freq_lower_bound_for_log);
        fqGridLabel.setGridType(gridType);
        Log.i(TAG, "setFreqAxisMode(): set to mode " + mapType + " axisX.vL=" +
axisX.vLowerBound + "  freq_lower_bound_for_log = " + freq_lower_bound_for_log);
    }

    private void drawGridLines(Canvas c) {
        for(int i = 0; i < fqGridLabel.values.length; i++) {
            float xPos = (float)axisX.pixelFromV(fqGridLabel.values[i]);
            c.drawLine(xPos, 0, xPos, canvasHeight, gridPaint);
        }
        for(int i = 0; i < dbGridLabel.values.length; i++) {
            float yPos = (float)axisY.pixelFromV(dbGridLabel.values[i]);
            c.drawLine(0, yPos, canvasWidth, yPos, gridPaint);
        }
    }

    private double clampDB(double value) {
        if (value < AnalyzerGraphic.minDB || Double.isNaN(value)) {
            value = AnalyzerGraphic.minDB;
        }
        return value;
    }

    private Matrix matrix = new Matrix();
    private float[] tmpLineXY = new float[0];  // cache line data for drawing
    private double[] db_cache = null;
    int[] index_range_cache = new int[2];
    // index_range_cache[0] == beginFreqPt
    // index_range_cache[1] == endFreqPt
    AnalyzerUtil.PeakHoldAndFall peakHold = new AnalyzerUtil.PeakHoldAndFall();
    long timeLastCall;

    // Find the index range that the plot will appear inside view.
    // index_range[1] must never be reached.
    private void indexRangeFinder(double[] x_values, double x_step, int[] index_range) {
        if (index_range == null || index_range.length != 2) {
            // You are joking
            return ;
        }
        double viewMinX = axisX.vMinInView();
        double viewMaxX = axisX.vMaxInView();
        if (x_values == null) {
            index_range[0] = (int)floor(viewMinX / x_step);    // pointer to tmpLineXY
            index_range[1] = (int)ceil (viewMaxX / x_step)+1;
        } else {
```

```
      if (x_values.length == 0) {
        // mada joking
        return ;
      }
      // Assume x_values is in ascending order
      index_range[0] = AnalyzerUtil.binarySearchElem(x_values, viewMinX, true);
      index_range[1] = AnalyzerUtil.binarySearchElem(x_values, viewMaxX, false) + 1;
    }
    // Avoid log(0)
    if (((x_values == null && index_range[0] == 0)
        || (x_values != null && x_values[index_range[0]] <= 0))
        && axisX.mapType == ScreenPhysicalMapping.Type.LOG) {
      index_range[0]++;
    }
    // Avoid out-of-range access
    if (index_range[1] > db_cache.length) {
      // just in case canvasMaxFreq / freqDelta > nFreqPointsTotal
      index_range[1] = db_cache.length;
    }
  }

  private void plotBarThin(Canvas c, double[] y_value, double[] x_values, int i_begin, int i_end,
double x_inc, Paint linePainter) {
    if (tmpLineXY.length < 4*(y_value.length)) {
      Log.d(TAG, "plotBar(): new tmpLineXY");
      tmpLineXY = new float[4*(y_value.length)];
    }

    final double minYCanvas = axisY.pixelNoZoomFromV(AnalyzerGraphic.minDB);
    c.save();
    matrix.reset();
    matrix.setTranslate(0, (float)(-axisY.getShift() * canvasHeight));
    matrix.postScale(1, (float)axisY.getZoom());
    c.concat(matrix);
    //     float barWidthInPixel = 0.5f * freqDelta / (canvasMaxFreq - canvasMinFreq) *
canvasWidth;
    //     if (barWidthInPixel > 2) {
    //       linePaint.setStrokeWidth(barWidthInPixel);
    //     } else {
    //       linePaint.setStrokeWidth(0);
    //     }
    // plot directly to the canvas
    for (int i = i_begin; i < i_end; i++) {
      float x = (float)axisX.pixelFromV(x_values == null ? i * x_inc : x_values[i]);
```

```
            float y = (float)axisY.pixelNoZoomFromV(clampDB(y_value[i]));
            tmpLineXY[4 * i] = x;
            tmpLineXY[4 * i + 1] = (float)minYCanvas;
            tmpLineXY[4 * i + 2] = x;
            tmpLineXY[4 * i + 3] = y;
        }
        c.drawLines(tmpLineXY, 4*i_begin, 4*(i_end-i_begin), linePainter);
        c.restore();
    }

    private void plotBarThick(Canvas c, double[] y_value, double[] x_values, int i_begin, int i_end,
double x_inc, Paint linePainter) {
        if (tmpLineXY.length < 4*(y_value.length)) {
            Log.d(TAG, "plotBar(): new tmpLineXY");
            tmpLineXY = new float[4*(y_value.length)];
        }

        final double minYCanvas = axisY.pixelNoZoomFromV(AnalyzerGraphic.minDB);
        int pixelStep = 2;  // each bar occupy this virtual pixel
        c.save();
        matrix.reset();
        double extraPixelAlignOffset = 0.0f;
//        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.HONEYCOMB) {
//            // There is an shift for Android 4.4, while no shift for Android 2.3
//            // I guess that is relate to GL ES acceleration
//            if (c.isHardwareAccelerated()) {
//                extraPixelAlignOffset = 0.5f;
//            }
//        }
        matrix.setTranslate((float)(-axisX.getShift() * (y_value.length - 1) * pixelStep -
extraPixelAlignOffset),
                (float)(-axisY.getShift() * canvasHeight));
        matrix.postScale((float)(canvasWidth / ((axisX.vMaxInView() - axisX.vMinInView()) / x_inc *
pixelStep)), (float)axisY.getZoom());
        c.concat(matrix);
        // fill interval same as canvas pixel width.
        for (int i = i_begin; i < i_end; i++) {
            float x = x_values == null ? i * pixelStep : (float)(x_values[i] / x_inc * pixelStep);
            float y = (float)axisY.pixelNoZoomFromV(clampDB(y_value[i]));
            tmpLineXY[4*i  ] = x;
            tmpLineXY[4*i+1] = (float)minYCanvas;
            tmpLineXY[4*i+2] = x;
            tmpLineXY[4*i+3] = y;
        }
        c.drawLines(tmpLineXY, 4*i_begin, 4*(i_end-i_begin), linePainter);
        c.restore();
    }

    private void plotBar(Canvas c, double[] y_value, double[] x_values, int i_begin, int i_end,
double x_inc, Paint linePainter) {
        // If bars are very close to each other, draw bars as lines
        // Otherwise, zoom in so that lines look like bars.
        if (i_end - i_begin >= axisX.nCanvasPixel / 2
                || axisX.mapType != ScreenPhysicalMapping.Type.LINEAR) {
            plotBarThin(c, y_value, x_values, i_begin, i_end, x_inc, linePainter);
        } else {
            // for zoomed linear scale
            plotBarThick(c, y_value, x_values, i_begin, i_end, x_inc, linePainter);
        }
    }

    private void plotLine(Canvas c, double[] y_value, double[] x_values, int i_begin, int i_end,
double x_inc, Paint linePainter) {
        if (tmpLineXY.length < 4*(y_value.length)) {
            Log.d(TAG, "plotLine(): new tmpLineXY");
            tmpLineXY = new float[4*(y_value.length)];
        }
        c.save();
        matrix.reset();
        matrix.setTranslate(0, (float)(-axisY.getShift()*canvasHeight));
        matrix.postScale(1, (float)axisY.getZoom());
        c.concat(matrix);
        if (x_values == null) {
            float o_x = (float)axisX.pixelFromV(i_begin * x_inc);
            float o_y = (float)axisY.pixelNoZoomFromV(clampDB(y_value[i_begin]));
            for (int i = i_begin+1; i < i_end; i++) {
                float x = (float)axisX.pixelFromV(i * x_inc);
                float y = (float)axisY.pixelNoZoomFromV(clampDB(y_value[i]));
                tmpLineXY[4*i  ] = o_x;
                tmpLineXY[4*i+1] = o_y;
                tmpLineXY[4*i+2] = x;
                tmpLineXY[4*i+3] = y;
                o_x = x;
                o_y = y;
            }
        } else {
            float o_x = (float)axisX.pixelFromV(x_values[i_begin]);
            float o_y = (float)axisY.pixelNoZoomFromV(clampDB(y_value[i_begin]));
            for (int i = i_begin+1; i < i_end; i++) {
                float x = (float)axisX.pixelFromV(x_values[i]);
                float y = (float)axisY.pixelNoZoomFromV(clampDB(y_value[i]));
                tmpLineXY[4*i  ] = o_x;
                tmpLineXY[4*i+1] = o_y;
                tmpLineXY[4*i+2] = x;
                tmpLineXY[4*i+3] = y;
                o_x = x;
                o_y = y;
            }
        }
        c.drawLines(tmpLineXY, 4*(i_begin+1), 4*(i_end-i_begin-1), linePainter);
        c.restore();
    }

    private void plotLineBar(Canvas c, double[] db_cache, double[] x_values, boolean drawBar,
Paint linePaint, Paint barPaint) {
        if (db_cache == null || db_cache.length == 0) return;

        // There are db.length frequency points, including DC component
        int nFreqPointsTotal = db_cache.length - 1;
        double freqDelta = axisX.vUpperBound / nFreqPointsTotal;

        indexRangeFinder(x_values, freqDelta, index_range_cache);

        if (drawBar) {
            plotBar(c, db_cache, x_values, index_range_cache[0], index_range_cache[1], freqDelta,
barPaint);
        }
        plotLine(c, db_cache, x_values, index_range_cache[0], index_range_cache[1], freqDelta,
linePaint);
    }

    // Plot the spectrum into the Canvas c
    private void drawSpectrumOnCanvas(Canvas c, final double[] _db) {
        if (canvasHeight < 1 || _db == null || _db.length == 0) {
            return;
        }
        AnalyzerGraphic.setIsBusy(true);

        synchronized (_db) {  // TODO: need lock on savedDBSpectrum, but how?
            if (db_cache == null || db_cache.length != _db.length) {
                Log.d(TAG, "drawSpectrumOnCanvas(): new db_cache");
                db_cache = new double[_db.length];
            }
            System.arraycopy(_db, 0, db_cache, 0, _db.length);
        }

        long timeNow = SystemClock.uptimeMillis();
        peakHold.addCurrentValue(db_cache, (timeNow - timeLastCall)/1000.0);
        timeLastCall = timeNow;

        // Spectrum peak hold
        plotLineBar(c, peakHold.v_peak, null, false, linePeakPaint, null);

        // Spectrum line and bar
        plotLineBar(c, db_cache, null, !showLines, linePaintLight, linePaint);

        if (name_calib != null) {
            c.save();
            c.translate(30*DPRatio, (float)axisY.pixelFromV(0));
            c.drawText(name_calib, 0, 0, calibNamePaint);
            c.restore();
        }
        plotLineBar(c, y_calib, x_calib, false, calibLinePaint, null);

        AnalyzerGraphic.setIsBusy(false);
    }

    double[] y_calib = null;
    double[] x_calib = null;
    String name_calib = null;

    void addCalibCurve(double[] y, double[] x, String name) {
        y_calib = y;
        x_calib = x;
        name_calib = name;
    }

    // x, y is in pixel unit
    void setCursor(double x, double y) {
        cursorFreq = axisX.vFromPixel(x);  // frequency
        cursorDB   = axisY.vFromPixel(y);  // decibel
    }

    double getCursorFreq() {
        return  canvasWidth == 0 ? 0 : cursorFreq;
    }
```

```
    double getCursorDB() {
        return canvasHeight == 0 ?   0 : cursorDB;
    }

    void hideCursor() {
        cursorFreq = 0;
        cursorDB = 0;
    }

    private void drawCursor(Canvas c) {
        if (cursorFreq == 0) {
            return;
        }
        float cX, cY;
        cX = (float)axisX.pixelFromV(cursorFreq);
        cY = (float)axisY.pixelFromV(cursorDB);
        c.drawLine(cX, 0, cX, canvasHeight, cursorPaint);
        c.drawLine(0, cY, canvasWidth, cY, cursorPaint);
    }

    // Plot spectrum with axis and ticks on the whole canvas c
    void drawSpectrumPlot(Canvas c, double[] savedDBSpectrum) {
        fqGridLabel.updateGridLabels(axisX.vMinInView(), axisX.vMaxInView());
        dbGridLabel.updateGridLabels(axisY.vMinInView(), axisY.vMaxInView());
        drawGridLines(c);
        drawSpectrumOnCanvas(c, savedDBSpectrum);
        drawCursor(c);
        AxisTickLabels.draw(c, axisX, fqGridLabel,
                0f, 0f, 0, 1,
                labelPaint, gridPaint, gridPaint);
        AxisTickLabels.draw(c, axisY, dbGridLabel,
                0f, 0f, 1, 1,
                labelPaint, gridPaint, gridPaint);
    }
}
```

STFT

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import java.util.Arrays;

import android.nfc.Tag;
import android.util.Log;

import com.google.corp.productivity.specialprojects.android.fft.RealDoubleFFT;

import static java.lang.Math.PI;
import static java.lang.Math.abs;
import static java.lang.Math.asin;
import static java.lang.Math.cos;
import static java.lang.Math.exp;
import static java.lang.Math.log10;
import static java.lang.Math.pow;
import static java.lang.Math.round;
import static java.lang.Math.sin;
import static java.lang.Math.sqrt;

// Short Time Fourier Transform
class STFT {
   // data for frequency Analysis
   private double[] spectrumAmpOutCum;
   private double[] spectrumAmpOutTmp;
   private double[] spectrumAmpOut;
   private double[] spectrumAmpOutDB;
   private double[] spectrumAmpIn;
   private double[] spectrumAmpInTmp;
   private double[] wnd;
   private double wndEnergyFactor = 1;           // used to keep energy invariant under different
window
   private int sampleRate;
   private int fftLen;
```

```
    private int hopLen;                     // control overlap of FFTs = (1 - lopLen/fftLen)*100%
    private int spectrumAmpPt;
//    private double[][] spectrumAmpOutArray;
//    private int spectrumAmpOutArrayPt = 0;        // Pointer for spectrumAmpOutArray
    private int nAnalysed = 0;
    private RealDoubleFFT spectrumAmpFFT;
    private boolean boolAWeighting = false;
    private double cumRMS = 0;
    private int    cntRMS = 0;
    private double outRMS = 0;

    private double[] dBAFactor;    // multiply to power spectrum to get A-weighting
    private double[] micGain;

    private double sqr(double x) { return x*x; }

    // Generate multiplier for A-weighting
    private void initDBAFactor(int fftlen, double sampleRate) {
        dBAFactor = new double[fftlen/2+1];
        for (int i = 0; i < fftlen/2+1; i++) {
            double f = (double)i/fftlen * sampleRate;
            double r = sqr(12200)*sqr(sqr(f)) / ((f*f+sqr(20.6)) * sqrt((f*f+sqr(107.7)) *
(f*f+sqr(737.9))) * (f*f+sqr(12200)));
            dBAFactor[i] = r*r*1.58489319246111;  // 1.58489319246111 = 10^(1/5)
        }
    }

    private void initWindowFunction(int fftlen, String wndName) {
        wnd = new double[fftlen];
        switch (wndName) {
            case "Bartlett":
                for (int i=0; i<wnd.length; i++) {  // Bartlett
                    wnd[i] = asin(sin(PI*i/wnd.length))/PI*2;
                }
                break;
            case "Hanning":
                for (int i=0; i<wnd.length; i++) {  // Hanning, hw=1
                    wnd[i] = 0.5*(1-cos(2*PI*i/(wnd.length-1.))) *2;
                }
                break;
            case "Blackman":
                for (int i=0; i<wnd.length; i++) {  // Blackman, hw=2
                    wnd[i] = 0.42-0.5*cos(2*PI*i/(wnd.length-1))+0.08*cos(4*PI*i/(wnd.length-1));
                }
                break;
            case "Blackman Harris":
                for (int i=0; i<wnd.length; i++) {  // Blackman_Harris, hw=3
                    wnd[i] = (0.35875-0.48829*cos(2*PI*i/(wnd.length-
1))+0.14128*cos(4*PI*i/(wnd.length-1))-0.01168*cos(6*PI*i/(wnd.length-1))) *2;
                }
                break;
            case "Kaiser, a=2.0": {
                double a = 2.0;
                double dn = besselCal.i0(PI * a);
                for (int i=0; i<wnd.length; i++) {  // Kaiser, a=2.0
                    wnd[i] = besselCal.i0(PI*a*sqrt(1-(2.0*i/(wnd.length-1)-1.0)*(2.0*i/(wnd.length-1)-
1.0))) / dn;
                }
                break;
            }
            case "Kaiser, a=3.0": {
                double a = 3.0;
                double dn = besselCal.i0(PI * a);
                for (int i=0; i<wnd.length; i++) {  // Kaiser, a=3.0
                    wnd[i] = besselCal.i0(PI*a*sqrt(1-(2.0*i/(wnd.length-1)-1.0)*(2.0*i/(wnd.length-1)-
1.0))) / dn;
                }
                break;
            }
            case "Kaiser, a=4.0": {
                double a = 4.0;
                double dn = besselCal.i0(PI * a);
                for (int i=0; i<wnd.length; i++) {  // Kaiser, a=4.0
                    wnd[i] = besselCal.i0(PI*a*sqrt(1-(2.0*i/(wnd.length-1)-1.0)*(2.0*i/(wnd.length-1)-
1.0))) / dn;
                }
                break;
            }
            // 7 more window functions (by james34602, https://github.com/bewantbe/audio-
analyzer-for-android/issues/14 )
            case "Flat-top": {
                for (int i=0; i<wnd.length; i++) {
                    double f = 2 * PI * i / (wnd.length - 1);
                    wnd[i] = 1 - 1.93 * cos(f) + 1.29 * cos(2 * f) - 0.388 * cos(3 * f) + 0.028 * cos(4 * f);
                }
                break;
            }
            case "Nuttall": {
```

```
    double a0 = 0.355768;
    double a1 = 0.487396;
    double a2 = 0.144232;
    double a3 = 0.012604;
    for (int i=0; i<wnd.length; i++)
    {
        double scale = PI * i / (wnd.length - 1);
        wnd[i] = a0 - a1 * cos(2.0 * scale) + a2 * cos(4.0 * scale) - a3 * cos(6.0 * scale);
    }
    break;
}
case "Gaussian, b=3.0": {
    double Beta = 3.0;
    double Arg;
    for (int i=0; i<wnd.length; i++) {
        Arg = (Beta * (1.0 - ((double)i / (double)wnd.length) * 2.0));
        wnd[i] = exp(-0.5 * (Arg * Arg));
    }
    break;
}
case "Gaussian, b=5.0": {
    double Beta = 5.0;
    double Arg;
    for (int i=0; i<wnd.length; i++) {
        Arg = (Beta * (1.0 - ((double)i / (double)wnd.length) * 2.0));
        wnd[i] = exp(-0.5 * (Arg * Arg));
    }
    break;
}
case "Gaussian, b=6.0": {
    double Beta = 6.0;
    double Arg;
    for (int i=0; i<wnd.length; i++) {
        Arg = (Beta * (1.0 - ((double)i / (double)wnd.length) * 2.0));
        wnd[i] = exp(-0.5 * (Arg * Arg));
    }
    break;
}
case "Gaussian, b=7.0": {
    double Beta = 7.0;
    double Arg;
    for (int i=0; i<wnd.length; i++) {
        Arg = (Beta * (1.0 - ((double)i / (double)wnd.length) * 2.0));
        wnd[i] = exp(-0.5 * (Arg * Arg));
```

```
            }
            break;
        }
        case "Gaussian, b=8.0": {
            double Beta = 8.0;
            double Arg;
            for (int i=0; i<wnd.length; i++) {
                Arg = (Beta * (1.0 - ((double)i / (double)wnd.length) * 2.0));
                wnd[i] = exp(-0.5 * (Arg * Arg));
            }
            break;
        }
        default:
            for (int i=0; i<wnd.length; i++) {
                wnd[i] = 1;
            }
            break;
    }
    double normalizeFactor = 0;
    for (int i=0; i<wnd.length; i++) {
        normalizeFactor += wnd[i];
    }
    normalizeFactor = wnd.length / normalizeFactor;
    wndEnergyFactor = 0;
    for (int i=0; i<wnd.length; i++) {
        wnd[i] *= normalizeFactor;
        wndEnergyFactor += wnd[i]*wnd[i];
    }
    wndEnergyFactor = wnd.length / wndEnergyFactor;
}

void setAWeighting(boolean e_isAWeighting) {
    boolAWeighting = e_isAWeighting;
}

boolean getAWeighting() {
    return boolAWeighting;
}

private void init(int fftlen, int _hopLen, int sampleRate, int minFeedSize, String wndName) {
    if (minFeedSize <= 0) {
        throw new IllegalArgumentException("STFT::init(): should minFeedSize >= 1.");
    }
    if (((-fftlen)&fftlen) != fftlen) {
            // error: fftlen should be power of 2
            throw new IllegalArgumentException("STFT::init(): Currently, only power of 2 are
supported in fftlen");
        }
        this.sampleRate = sampleRate;
        fftLen = fftlen;
        hopLen = _hopLen;                          // 50% overlap by default
        spectrumAmpOutCum = new double[fftlen/2+1];
        spectrumAmpOutTmp= new double[fftlen/2+1];
        spectrumAmpOut   = new double[fftlen/2+1];//holds PSD buffer
        spectrumAmpOutDB = new double[fftlen/2+1];//in dB; spectrumAmpOutDB is used to
calculate RMS
        spectrumAmpIn    = new double[fftlen];
        spectrumAmpInTmp = new double[fftlen];
        spectrumAmpFFT   = new RealDoubleFFT(spectrumAmpIn.length);
//        spectrumAmpOutArray = new double[(int)ceil((double)minFeedSize / (fftlen/2))][]; // /2
since half overlap
//        for (int i = 0; i < spectrumAmpOutArray.length; i++) {
//            spectrumAmpOutArray[i] = new double[fftlen/2+1];
//        }

        initWindowFunction(fftlen, wndName);
        initDBAFactor(fftlen, sampleRate);
        clear();
        boolAWeighting = false;
    }

    STFT(AnalyzerParameters analyzerParam) {
        init(analyzerParam.fftLen, analyzerParam.hopLen, analyzerParam.sampleRate,
analyzerParam.nFFTAverage, analyzerParam.wndFuncName);
        if (analyzerParam.micGainDB != null) {
            if (micGain == null || micGain.length != analyzerParam.micGainDB.length) {
                micGain = new double[analyzerParam.micGainDB.length];
            }
            Log.w("STFT:", "calib loaded. micGain.length = " + micGain.length);
            for (int i = 0; i < micGain.length; i++) {
                micGain[i] = pow(10, analyzerParam.micGainDB[i] / 10.0);
            }
        } else {
            Log.w("STFT:", "no calib");
        }
    }

    public void feedData(short[] ds) {
      feedData(ds, ds.length);
    }

    void feedData(short[] ds, int dsLen) {
        if (dsLen > ds.length) {
            Log.e("STFT", "dsLen > ds.length !");
            dsLen = ds.length;
        }
        int inLen = spectrumAmpIn.length;
        int outLen = spectrumAmpOut.length;
        int dsPt = 0;           // input data point to be read
        while (dsPt < dsLen) {
            while (spectrumAmpPt < 0 && dsPt < dsLen) {  // skip data when hopLen > fftLen
                double s = ds[dsPt++] / 32768.0;
                spectrumAmpPt++;
                cumRMS += s*s;
                cntRMS++;
            }
            while (spectrumAmpPt < inLen && dsPt < dsLen) {
                double s = ds[dsPt++] / 32768.0;
                spectrumAmpIn[spectrumAmpPt++] = s;
                cumRMS += s*s;
                cntRMS++;
            }
            if (spectrumAmpPt == inLen) {    // enough data for one FFT
                for (int i = 0; i < inLen; i++) {
                    spectrumAmpInTmp[i] = spectrumAmpIn[i] * wnd[i];
                }
                spectrumAmpFFT.ft(spectrumAmpInTmp);
                fftToAmp(spectrumAmpOutTmp, spectrumAmpInTmp);//spectrumAmpOutTmp  ==
holds PSD buffer
//                System.arraycopy(spectrumAmpOutTmp, 0,
spectrumAmpOutArray[spectrumAmpOutArrayPt], 0,
//                                 spectrumAmpOutTmp.length);
//                spectrumAmpOutArrayPt = (spectrumAmpOutArrayPt+1) %
spectrumAmpOutArray.length;
                for (int i = 0; i < outLen; i++) {
                    spectrumAmpOutCum[i] += spectrumAmpOutTmp[i];
                }
                nAnalysed++;
                if (hopLen < fftLen) {
                    System.arraycopy(spectrumAmpIn, hopLen, spectrumAmpIn, 0, fftLen - hopLen);
                }
                spectrumAmpPt = fftLen - hopLen;  // can be positive and negative
            }
        }
    }

    // Convert complex amplitudes to absolute amplitudes.
    private void fftToAmp(double[] dataOut, double[] data) {
        // data.length should be a even number
        double scaler = 2.0*2.0 / (data.length * data.length);  // *2 since there are positive and
negative frequency part
        dataOut[0] = data[0]*data[0] * scaler / 4.0;
        int j = 1;
        for (int i = 1; i < data.length - 1; i += 2, j++) {
            dataOut[j] = (data[i]*data[i] + data[i+1]*data[i+1]) * scaler;
        }
        dataOut[j] = data[data.length-1]*data[data.length-1] * scaler / 4.0;
    }

    final double[] getSpectrumAmp()
    {//this function corrects automatic gain control. I must run this function before I do any
measurement in spectrumAmpOut
        if (nAnalysed != 0) {    // no new result
            int outLen = spectrumAmpOut.length;
            double[] sAOC = spectrumAmpOutCum;
            for (int j = 0; j < outLen; j++) {
                sAOC[j] /= nAnalysed;
            }
            if (micGain != null && micGain.length == sAOC.length) {
                // No correction to phase.
                // Correction to DC is fake.
                for (int j = 0; j < outLen; j++) {
                    sAOC[j] /= micGain[j];
                }
            }
            if (boolAWeighting) {
                for (int j = 0; j < outLen; j++) {
                    sAOC[j] *= dBAFactor[j];
                }
            }
            System.arraycopy(sAOC, 0, spectrumAmpOut, 0, outLen);//now spectrumAmpOut is
normalized by mic gain

            Arrays.fill(sAOC, 0.0);
            nAnalysed = 0;
            for (int i = 0; i < outLen; i++) {
```

```
        spectrumAmpOutDB[i] = 10.0 * log10(spectrumAmpOut[i]);
      }
    }
    return spectrumAmpOut;
  }

  final double[] getSpectrumAmpDB() {
    getSpectrumAmp();
    return spectrumAmpOutDB;
  }

  double getRMS() {
    if (cntRMS > 8000/30) {
      outRMS = sqrt(cumRMS / cntRMS * 2.0);  // "* 2.0" normalize to sine wave.
      cumRMS = 0;
      cntRMS = 0;
    }
    return outRMS;
  }



  int nElemSpectrumAmp() {
   return nAnalysed;
  }

  double maxAmpFreq = Double.NaN, maxAmpDB = Double.NaN;

  void calculatePeak() {
    //getSpectrumAmpDB();commented out by AV since it is already called inside Sampling
Loop>run(). It can only be called once. It was a bug to call it again.
    // Find and show peak amplitude
    maxAmpDB  = 20 * log10(0.125/32768);
    maxAmpFreq = 0;
    for (int i = 1; i < spectrumAmpOutDB.length; i++) {  // skip the direct current term
      if (spectrumAmpOutDB[i] > maxAmpDB) {
        maxAmpDB  = spectrumAmpOutDB[i];
        maxAmpFreq = i;
      }
    }
    maxAmpFreq = maxAmpFreq * sampleRate / fftLen;

    // Slightly better peak amplitude finder
```

```
        // The peak amplitude around spectrumDB should look like quadratic curve after good window function
        // a*x^2 + b*x + c = y
        // a - b + c = x1
        //         c = x2
        // a + b + c = x3
        if (sampleRate / fftLen < maxAmpFreq && maxAmpFreq < sampleRate/2 - sampleRate / fftLen) {
            int id = (int)(round(maxAmpFreq/sampleRate*fftLen));
            double x1 = spectrumAmpOutDB[id-1];
            double x2 = spectrumAmpOutDB[id];
            double x3 = spectrumAmpOutDB[id+1];
            double c = x2;
            double a = (x3+x1)/2 - x2;
            double b = (x3-x1)/2;
            if (a < 0) {
                double xPeak = -b/(2*a);
                if (abs(xPeak) < 1) {
                    maxAmpFreq += xPeak * sampleRate / fftLen;
                    maxAmpDB = (4*a*c - b*b)/(4*a);
                }
            }
        }
    }


    double getRMSFromFT(int freq) {
        //AV changed definition: calculate RMS in dB above the frequency freq;
        //getSpectrumAmpDB();commented out by AV since it is already called inside Sampling Loop>run(). It can only be called once. It was a bug to call it again.
        double s = 0;
        int indexFreq=1;
        if(freq>0) indexFreq=(freq*fftLen) / sampleRate;//convert freq to indexFreq in freq domain: freq=sampleRate/2 corresponds to spectrumAmpOut.Length;
        for (int i = indexFreq; i < spectrumAmpOut.length; i++) {
            s += spectrumAmpOut[i];
        }
        return sqrt(s * wndEnergyFactor);
    }

    double getMeanPSD_dB(int freqLeft, int freqRight)
    {//AV freqLeft is absolute value of freq
        double meanAmp=0; int ni=0;
        int indexFreqLeft=1, indexFreqRight=1;
        if(freqLeft>=0)  indexFreqLeft =(freqLeft* fftLen) / sampleRate;//convert freq to indexFreq in freq domain: freq=sampleRate/2 corresponds to spectrumAmpOut.Length;
        if(freqRight>=0) indexFreqRight=(freqRight*fftLen) / sampleRate;
        if(freqRight>spectrumAmpOutDB.length) freqRight=spectrumAmpOutDB.length;
        for (int i=indexFreqLeft; i<indexFreqRight; i++) {
            meanAmp += spectrumAmpOutDB[i];
            ni++;
        }
        return( meanAmp/ni );
    }

    double getPeakAmpl_dB(int freqOfPeak) {
        int indexFreq=1;
        if(freqOfPeak>=0)  indexFreq =(freqOfPeak* fftLen) / sampleRate;//convert freq to indexFreq in freq domain: freq=sampleRate/2 corresponds to spectrumAmpOut.Length;
        double x1 = spectrumAmpOutDB[indexFreq];
        double x2 = spectrumAmpOutDB[indexFreq];
        double x3 = spectrumAmpOutDB[indexFreq+1];
        return (x1+x2+x3)/3;
    }

    int getIndexOfPeakFreq(int freqLeft, int freqRight)
    {//AV freqLeft is absolute value of freq
        double maxAmpl=0; int indexMax=0;
        int indexFreqLeft=1, indexFreqRight=1;
        if(freqLeft>=0)  indexFreqLeft =(freqLeft* fftLen) / sampleRate;//convert freq to indexFreq in freq domain: freq=sampleRate/2 corresponds to spectrumAmpOut.Length;
        if(freqRight>=0) indexFreqRight=(freqRight*fftLen) / sampleRate;
        if(freqRight>spectrumAmpOut.length) freqRight=spectrumAmpOut.length;
        for (int i=indexFreqLeft; i<indexFreqRight; i++) {
            if(maxAmpl < spectrumAmpOut[i]){ maxAmpl=spectrumAmpOut[i]; indexMax=i;}
        }
        return( (indexMax*sampleRate) / fftLen );
    }

    double ArrayAvgFreq(int freqCenter, int deltaLeft, int deltaRight)
    {//deltaLeft is delta, relative to freqCenter.
        int ni=0, i=0;
        double result=0;
        int iFreqLeft=1, iFreqRight=1;
        iFreqLeft =((deltaLeft +freqCenter)*fftLen) / sampleRate;//convert freq to indexFreq in freq domain: freq=sampleRate/2 corresponds to spectrumAmpOut.Length;
        iFreqRight=((deltaRight+freqCenter)*fftLen) / sampleRate;
        for(i=iFreqLeft; i<=iFreqRight; i++)
        {
            if(i>=0 && i<spectrumAmpOut.length)
            {//watch to avoid going beyond the array range
                result += spectrumAmpOut[i];
                ni++;
            }
        }
        if(ni<1) return 999.9f;
        else return(result/ni);//index 104 in the PSD corresponds to 809Hz
    }


    double getPeakToMeanRatio(int freqOfPeak, int maxDeltaFreq)
    {//
        double ratio=0, maxRatio=0, max=0, mean=0;
        for(int freq=10; freq<=maxDeltaFreq; freq+=5)
        {//scan for peak thickness that generates the highest ratio against background
            max =  ArrayAvgFreq(freqOfPeak, -freq, freq);
            mean=( ArrayAvgFreq(freqOfPeak, -round(freq*2.4f), -round(freq*1.1f)) +
ArrayAvgFreq(freqOfPeak, round(freq*1.1f), round(freq*2.4f)) )/2;//this must be away from
harmonics
            if( mean!=0 ) {
                ratio = max / mean;
                if (ratio > (maxRatio * 1.2f)) {
                    maxRatio = ratio;
                }//by multiplying by 1.2 we favor narrow peaks over broad peaks.
            }
        }
        return maxRatio;
    }


    void clear() {
        spectrumAmpPt = 0;
        Arrays.fill(spectrumAmpOut, 0.0);
        Arrays.fill(spectrumAmpOutDB, log10(0));
        Arrays.fill(spectrumAmpOutCum, 0.0);
//        for (int i = 0; i < spectrumAmpOutArray.length; i++) {
//            Arrays.fill(spectrumAmpOutArray[i], 0.0);
//        }
    }

}
```

toDogActivity

```
package github.bewantbe.audio_analyzer_for_android;

import android.content.SharedPreferences;
import android.app.Activity;
import android.app.AlertDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.ActivityNotFoundException;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.location.Criteria;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.TextView;
import android.widget.ListView;
import android.widget.Switch;
import android.Manifest;


import com.wowwee.bluetoothrobotcontrollib.chip.ChipRobot;
import com.wowwee.bluetoothrobotcontrollib.chip.ChipRobotFinder;


import java.util.ArrayList;
```

```java
import java.util.List;


public class toDogActivity extends Activity implements ChipRobot.ChipRobotInterface {

    static String prefDirect = "MyPrefs";
    static SharedPreferences dogPref;
    static SharedPreferences.Editor dogEdit;
    ChipBaseFragment baseFragmentReady;

    private boolean isFirst = false;
    private static final int REQUEST_ENABLE_BT = 1;
    public static final int MY_PERMISSIONS_REQUEST_LOCATION = 99;
    private BluetoothAdapter mBluetoothAdapter;
    private TextView textView;
    private ListView listView;
    public Switch attemptAutoBool;
    List<String> robotNameList;
    LocationManager locationManager;
    String provider;
    String dogChangeVar;// Will probably be changed to a class, for now string placeholder

    @Override
    public void onCreate(Bundle savedInstances) {

        super.onCreate(savedInstances);
        setContentView(R.layout.todog);

        final int flags = View.SYSTEM_UI_FLAG_LAYOUT_STABLE
                | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
                | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
                | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
                | View.SYSTEM_UI_FLAG_FULLSCREEN
                | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY;

        dogPref = getApplicationContext().getSharedPreferences(prefDirect, 0);
        dogEdit = dogPref.edit();


        //getActivity().getWindow().getDecorView().setSystemUiVisibility(flags);


        //View view = inflater.inflate(R.layout.fragment_connect, container, false);
        textView = (TextView) findViewById(R.id.topTV);
        listView = (ListView) findViewById(R.id.connectionTable);
        attemptAutoBool = (Switch) findViewById(R.id.AttemptAuto);
        String[] robotNameArr = {"Please turn on CHIP"};
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(toDogActivity.this,
android.R.layout.simple_list_item_1, android.R.id.text1, robotNameArr);
        listView.setAdapter(adapter);
        checkLocationPermission();


        final Button refreshBtn = (Button) findViewById(R.id.refreshBtn);
        refreshBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                ChipRobotFinder.getInstance().clearFoundChipList();
                scanLeDevice(false);
                updateChipList();
                scanLeDevice(true);
            }
        });

        final Button listConnectToDog = (Button) findViewById(R.id.toConnectionList);
        listConnectToDog.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                refreshConnection();
            }
        });

        locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        // listConnectToDog.setVisibility(View.INVISIBLE);

        provider = locationManager.getBestProvider(new Criteria(), false);

        this.initBluetooth();


        listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
                                    long id) {
                if (ChipRobotFinder.getInstance().getChipFoundList().size() > 0) { // If there exists a
dog in the list
```

```
                    String targetRobotName = robotNameList.get(position); // clicked robot becomes
the robot name
                    dogChangeVar = targetRobotName;
                    // Save typical robot ID and submit the check to false.
                    for (ChipRobot robot : (List<ChipRobot>)
ChipRobotFinder.getInstance().getChipFoundList()) {
                        if (robot.getName().contentEquals(targetRobotName)) {
                            final ChipRobot connectChipRobot = robot;
                            toDogActivity.this.runOnUiThread(new Runnable() {
                                public void run() {
                                    connect(connectChipRobot);
                                    // Stop scan Chip
                                    scanLeDevice(false);
                                }
                            });
                            dogEdit.putString("dogID", targetRobotName);
                            dogEdit.putBoolean("firstConnect", false);
                            dogEdit.commit();
                            openAnalyzerActivity();
                            break;
                        }
                    }
                }
            }
        });
        // We want to set a listview listener for when a field is added or changed. When that
happens, attempt to autoconnect if it
        // is the prefered dog

        if (!dogPref.getBoolean("firstConnect",true)){
            attemptAutoBool.setShowText(true);
        }
        // Conditional to check if this is the first time the program has run. If yes, we ignore it, if not,
we just chill and wait for the onclick listeners for connecting
        // Make this just attempt auto correct, have an if otherwise

    }



    void connect(ChipRobot robot) {
        robot.setCallbackInterface(this);
        robot.connect(toDogActivity.this);
    }
    @Override
    public void onResume() {
        super.onResume();

        // Register ChipRobotFinder broadcast receiver
        this.registerReceiver(mChipFinderBroadcastReceiver,
ChipRobotFinder.getChipRobotFinderIntentFilter());

        // Ensures Bluetooth is enabled on the device.  If Bluetooth is not currently enabled,
        // fire an intent to display a dialog asking the user to grant permission to enable it.
        if (mBluetoothAdapter != null && !mBluetoothAdapter.isEnabled()) {
            if (!mBluetoothAdapter.isEnabled()) {
                try {
                    Intent enableBtIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
                    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);

                } catch (ActivityNotFoundException ex) {
                }
            }
        }

        // Start scan
        ChipRobotFinder.getInstance().clearFoundChipList();
        scanLeDevice(false);
        updateChipList();
        scanLeDevice(true);
    }

    @Override
    public void onPause() {
        super.onPause();
        this.unregisterReceiver(mChipFinderBroadcastReceiver);
        scanLeDevice(false);
    }

    private void initBluetooth(){
        final BluetoothManager bluetoothManager = (BluetoothManager)
this.getSystemService(Context.BLUETOOTH_SERVICE);
        mBluetoothAdapter = bluetoothManager.getAdapter();
        ChipRobotFinder.getInstance().setBluetoothAdapter(mBluetoothAdapter);
        ChipRobotFinder.getInstance().setApplicationContext(this);
    }
    public void scanLeDevice(final boolean enable) {
        if (enable) {
            Log.d("ChipScan", "Scan Le device start");
            // Stops scanning after a pre-defined scan period.
            ChipRobotFinder.getInstance().scanForChipContinuous();
        }else{
            Log.d("ChipScan", "Scan Le device stop");
            ChipRobotFinder.getInstance().stopScanForChipContinuous();
        }
    }

    public void updateChipList(){
        robotNameList = new ArrayList();
        for (ChipRobot robot :
(List<ChipRobot>)ChipRobotFinder.getInstance().getChipFoundList()) {
            robotNameList.add(robot.getName());
            if (robot.getName() == dogPref.getString("dogID", " ") &&
attemptAutoBool.getShowText()) {
                final ChipRobot connectChipRobot = robot;
                toDogActivity.this.runOnUiThread(new Runnable() {
                    public void run() {
                        connect(connectChipRobot);
                        // Stop scan Chip
                        scanLeDevice(false);
                    }
                });
                openAnalyzerActivity();
                break;
                }
            }



        String[] robotNameArr;
        if (robotNameList.size() > 0){
            robotNameArr = robotNameList.toArray(new String[0]);

        }
        else {
            robotNameArr = new String[1];
            robotNameArr[0] = "Please turn on CHIP";
        }
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, android.R.id.text1, robotNameArr);
```

```
        listView.setAdapter(adapter);
    }

    private final BroadcastReceiver mChipFinderBroadcastReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (ChipRobotFinder.ChipRobotFinder_ChipFound.equals(action)){
                BluetoothDevice device =
(BluetoothDevice)(intent.getExtras().get("BluetoothDevice"));
                Log.d("BLE", "ChipScanFragment broadcast receiver found Chip: " +
device.getName());
                updateChipList();
            } else if (ChipRobotFinder.ChipRobotFinder_ChipListCleared.equals(action)) {
                updateChipList();
            }
        }
    };




    public void openAnalyzerActivity(){
            Intent intent = new Intent(this,AnalyzerActivity.class);
            startActivity(intent);
    }

    @Override
    public void chipDeviceReady(ChipRobot chipRobot) {

    }

    @Override
    public void chipDeviceDisconnected(ChipRobot chipRobot) {

    }

    @Override
    public void chipDidReceiveVolumeLevel(ChipRobot chipRobot, byte b) {

    }

    @Override
    public void chipDidReceivedBatteryLevel(ChipRobot chipRobot, float v, byte b, byte b1) {
    }

    @Override
    public void chipDidReceiveDogVersionWithBodyHardware(int i, int i1, int i2, int i3, int i4, int i5,
int i6, int i7, int i8, int i9) {

    }

    @Override
    public void chipDidSendAdultOrKidSpeed() {

    }

    @Override
    public void chipDidReceiveAdultOrKidSpeed(byte b) {

    }

    @Override
    public void chipDidReceiveEyeRGBBrightness(byte b) {

    }

    @Override
    public void chipDidReceiveCurrentClock(int i, int i1, int i2, int i3, int i4, int i5, int i6) {

    }

    @Override
    public void chipDidReceiveCurrentAlarm(int i, int i1, int i2, int i3, int i4) {

    }

    @Override
    public void chipDidReceiveBodyconStatus(int i) {

    }
    public boolean checkLocationPermission() {
        if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.ACCESS_FINE_LOCATION)
                != PackageManager.PERMISSION_GRANTED) {

            // Should we show an explanation?
```

```
                // No explanation needed, we can request the permission.
                ActivityCompat.requestPermissions(this,
                        new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                        MY_PERMISSIONS_REQUEST_LOCATION);

            return false;
        } else {
            return true;
        }
    }
    public void refreshConnection(){
        dogEdit.putBoolean("firstConnect",true);
        dogEdit.putString("dogID"," ");
        dogEdit.commit();
        ChipRobotFinder.getInstance().clearFoundChipList();
        scanLeDevice(false);
        updateChipList();
        scanLeDevice(true);
    }


}
```

wavWriter

```
/* Copyright 2014 Eddy Xiao <bewantbe@gmail.com>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package github.bewantbe.audio_analyzer_for_android;

import android.annotation.SuppressLint;
import android.os.Environment;
import android.os.StatFs;
import android.util.Log;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.RandomAccessFile;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

class WavWriter {
  private static final String TAG = "WavWriter";
  private File outPath;
  private OutputStream out;
  private byte[] header = new byte[44];
  final String relativeDir = "/Recorder";

  private int channels = 1;
  private byte RECORDER_BPP = 16;  // bits per sample
  private int byteRate;            // Average bytes per second
  private int totalDataLen  = 0;   // (file size) - 8
  private int totalAudioLen = 0;   // bytes of audio raw data
  private int framesWritten = 0;

  WavWriter(int sampleRate) {
    byteRate = sampleRate*RECORDER_BPP/8*channels;

    header[0] = 'R';  // RIFF/WAVE header
    header[1] = 'I';
    header[2] = 'F';
    header[3] = 'F';
    header[4] = (byte) (totalDataLen & 0xff);
    header[5] = (byte) ((totalDataLen >> 8) & 0xff);
    header[6] = (byte) ((totalDataLen >> 16) & 0xff);
    header[7] = (byte) ((totalDataLen >> 24) & 0xff);
    header[8] = 'W';
    header[9] = 'A';
    header[10] = 'V';
    header[11] = 'E';
    header[12] = 'f';  // 'fmt ' chunk
    header[13] = 'm';
    header[14] = 't';
    header[15] = ' ';
    header[16] = 16;  // 4 bytes: size of 'fmt ' chunk
    header[17] = 0;
    header[18] = 0;
    header[19] = 0;
    header[20] = 1;  // format = 1, PCM/uncompressed
    header[21] = 0;
    header[22] = (byte) channels;
    header[23] = 0;
    header[24] = (byte) (sampleRate & 0xff);
    header[25] = (byte) ((sampleRate >> 8) & 0xff);
    header[26] = (byte) ((sampleRate >> 16) & 0xff);
    header[27] = (byte) ((sampleRate >> 24) & 0xff);
    header[28] = (byte) (byteRate & 0xff);              // Average bytes per second
    header[29] = (byte) ((byteRate >> 8) & 0xff);
    header[30] = (byte) ((byteRate >> 16) & 0xff);
    header[31] = (byte) ((byteRate >> 24) & 0xff);
    header[32] = (byte) (channels * RECORDER_BPP / 8);  // Block align (number of bytes per
sample slice)
    header[33] = 0;
    header[34] = RECORDER_BPP;                          // bits per sample (Significant bits per
sample)
    header[35] = 0;                                     // Extra format bytes
    header[36] = 'd';
    header[37] = 'a';
    header[38] = 't';
    header[39] = 'a';
    header[40] = (byte) (totalAudioLen & 0xff);
    header[41] = (byte) ((totalAudioLen >> 8) & 0xff);
    header[42] = (byte) ((totalAudioLen >> 16) & 0xff);
    header[43] = (byte) ((totalAudioLen >> 24) & 0xff);
  }

  private static final int version = android.os.Build.VERSION.SDK_INT;

  @SuppressLint("NewApi")
  @SuppressWarnings("deprecation")
  double secondsLeft() {
    long byteLeft;
    if (version >= 9) {
      byteLeft = outPath.getFreeSpace();  // Need API level 9
    } else {
      StatFs statFs = new StatFs(outPath.getAbsolutePath());
      byteLeft = (statFs.getAvailableBlocks() * (long)statFs.getBlockSize());
    }
    if (byteRate == 0 || byteLeft == 0) {
      return 0;
    }
    return (double)byteLeft / byteRate;
  }

  boolean start() {
    if (!isExternalStorageWritable()) {
      return false;
    }
    File path = new File(Environment.getExternalStorageDirectory().getPath() + relativeDir);
    if (!path.exists() && !path.mkdirs()) {
      Log.e(TAG, "Failed to make directory: " + path.toString());
      return false;
    }
    DateFormat df = new SimpleDateFormat("yyyy-MM-dd_HH'h'mm'm'ss.SSS's'", Locale.US);
    String nowStr = df.format(new Date());
    outPath = new File(path, "rec" + nowStr + ".wav");

    try {
      out = new FileOutputStream(outPath);
      out.write(header, 0, 44);
      //
http://developer.android.com/reference/android/os/Environment.html#getExternalStoragePublic
Directory%28java.lang.String%29
    } catch (IOException e) {
      Log.w(TAG, "start(): Error writing " + outPath, e);
      out = null;
    }
    return true;
  }

  void stop() {
    if (out == null) {
      Log.w(TAG, "stop(): Error closing " + outPath + "  null pointer");
      return;
    }
    try {
      out.close();
    } catch (IOException e) {
      Log.w(TAG, "stop(): Error closing " + outPath, e);
    }
    out = null;
    // Modify totalDataLen and totalAudioLen to match data
    RandomAccessFile raf;
    try {
      totalAudioLen = framesWritten * RECORDER_BPP / 8 * channels;
      totalDataLen = header.length + totalAudioLen - 8;
      raf = new RandomAccessFile(outPath, "rw");
      raf.seek(4);
      raf.write((byte) ((totalDataLen      ) & 0xff));
      raf.write((byte) ((totalDataLen >>  8) & 0xff));
      raf.write((byte) ((totalDataLen >> 16) & 0xff));
      raf.write((byte) ((totalDataLen >> 24) & 0xff));
      raf.seek(40);
      raf.write((byte) ((totalAudioLen      ) & 0xff));
      raf.write((byte) ((totalAudioLen >>  8) & 0xff));
      raf.write((byte) ((totalAudioLen >> 16) & 0xff));
      raf.write((byte) ((totalAudioLen >> 24) & 0xff));
      raf.close();
    } catch (IOException e) {
      Log.w(TAG, "stop(): Error modifying " + outPath, e);
    }
  }

  private byte[] byteBuffer;

  // Assume RECORDER_BPP == 16 and channels == 1
  void pushAudioShort(short[] ss, int numOfReadShort) {
    if (out == null) {
      Log.w(TAG, "pushAudioShort(): Error writing " + outPath + "  null pointer");
      return;
    }
    if (byteBuffer == null || byteBuffer.length != ss.length*2) {
      byteBuffer = new byte[ss.length*2];
    }
    for (int i = 0; i < numOfReadShort; i++) {
      byteBuffer[2*i  ] = (byte)(ss[i] & 0xff);
      byteBuffer[2*i+1] = (byte)((ss[i]>>8) & 0xff);
    }
    framesWritten += numOfReadShort;
    try {
      out.write(byteBuffer, 0, numOfReadShort*2);
      // if use out.write(byte), then a lot of GC will generated
    } catch (IOException e) {
      Log.w(TAG, "pushAudioShort(): Error writing " + outPath, e);
      out = null;
    }
  }

  double secondsWritten() {
    return (double) framesWritten /(byteRate*8/RECORDER_BPP/channels);
  }

  /* Checks if external storage is available for read and write */
  boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();  // Need API level 8
    return Environment.MEDIA_MOUNTED.equals(state);
  }

  //String getPath() {
    //return outPath.getPath();
  //}

}
```

res
layout
activity_info_rec.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".InfoRecActivity" >

    <TextView
        android:id="@+id/info_rec_tv"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:textSize="16sp"
        android:typeface="monospace"
        android:text="@string/text_view_info_log_0" />

</RelativeLayout>
```

dialog_view_range.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:paddingLeft="10dp"
    android:paddingRight="10dp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/show_range_tv_fL"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:gravity="center_vertical"
        android:text="@string/show_range_tv_fL" />

    <TextView
        android:id="@+id/show_range_tv_fH"
        android:layout_below="@id/show_range_tv_fL"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:gravity="center_vertical"
        android:text="@string/show_range_tv_fH" />

    <!-- Somehow android:layout_height="wrap_content" not working correctly on low resolution device -->
    <EditText
        android:id="@+id/et_freq_setting_lower_bound"
        android:layout_below="@id/show_range_tv_fH"
        android:layout_width="wrap_content"
        android:layout_height="40sp"
        android:layout_marginTop="5dp"
        android:maxEms="6"
        android:inputType="numberDecimal"
        android:hint="@string/et_freq_setting_lower_bound" />

    <EditText
        android:id="@+id/et_freq_setting_upper_bound"
        android:layout_below="@id/show_range_tv_fH"
        android:layout_toRightOf="@id/et_freq_setting_lower_bound"
        android:layout_toEndOf="@id/et_freq_setting_lower_bound"
        android:layout_width="wrap_content"
        android:layout_height="40sp"
        android:layout_marginTop="5dp"
        android:maxEms="6"
        android:inputType="numberDecimal"
        android:hint="@string/et_freq_setting_upper_bound" />

    <TextView
        android:id="@+id/show_range_tv_dBL"
        android:layout_below="@+id/et_freq_setting_upper_bound"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:gravity="center_vertical"
        android:text="@string/show_range_tv_dBL"/>

    <TextView
        android:id="@+id/show_range_tv_dBH"
        android:layout_below="@+id/show_range_tv_dBL"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:gravity="center_vertical"
        android:text="@string/show_range_tv_dBH"/>

    <EditText
        android:id="@+id/et_db_setting_lower_bound"
        android:layout_below="@+id/show_range_tv_dBH"
        android:layout_width="wrap_content"
        android:layout_height="40sp"
        android:layout_marginTop="5dp"
        android:maxEms="6"
        android:inputType="numberDecimal|numberSigned"
        android:hint="@string/et_db_setting_lower_bound" />

    <EditText
        android:id="@+id/et_db_setting_upper_bound"
        android:layout_below="@+id/show_range_tv_dBH"
        android:layout_toRightOf="@id/et_freq_setting_lower_bound"
        android:layout_toEndOf="@id/et_freq_setting_lower_bound"
        android:layout_width="wrap_content"
        android:layout_height="40sp"
        android:layout_marginTop="5dp"
        android:maxEms="6"
        android:inputType="numberDecimal|numberSigned"
        android:hint="@string/et_db_setting_upper_bound" />

    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_below="@+id/et_db_setting_upper_bound"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingLeft="8dp"
        android:paddingRight="8dp"
        android:layout_marginTop="4dp"
        android:layout_marginBottom="8dp"
        android:orientation="horizontal" >

        <CheckBox android:id="@+id/show_range_lock"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:layout_marginTop="1dp"
            android:layout_marginBottom="1dp"
            android:text="@string/show_range_lock" />

        <Button android:id="@+id/show_range_button_load"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:layout_marginStart="20dp"
            android:layout_marginLeft="20dp"
            android:layout_marginTop="1dp"
            android:layout_marginBottom="1dp"
            android:textSize="16sp"
            android:text="@string/show_range_button_lock" />

    </LinearLayout>

</RelativeLayout>
```

main
main.xml (landscape)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res-auto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

```
<view
    android:id="@+id/plot"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_weight="0.32"
    class="github.bewantbe.audio_analyzer_for_android.AnalyzerGraphic"
    custom:cutoffDb="-25"
    custom:sampleRate="16000" />


<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <Button
        android:id="@+id/toDog"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Go To Dog"></Button>

    <TextView
        android:id="@+id/textview_RMS"
        android:layout_width="60dp"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:lines="2"
        android:visibility="invisible"
        android:text="@string/textview_RMS_text"
        android:typeface="monospace" />

    <TextView
        android:id="@+id/textview_cur"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:visibility="invisible"
        android:layout_toRightOf="@id/textview_RMS"
        android:layout_toEndOf="@id/textview_RMS"
        android:text="@string/textview_cur_text"
        android:typeface="monospace" />

    <TextView
        android:id="@+id/textview_peak"
        android:layout_width="wrap_content"
        android:visibility="invisible"
        android:layout_height="wrap_content"
        android:layout_below="@id/textview_cur"
        android:layout_toRightOf="@id/textview_RMS"
        android:layout_toEndOf="@id/textview_RMS"
        android:text="@string/textview_peak_text"
        android:typeface="monospace" />

    <Button
        android:id="@+id/button_sample_rate"
        android:layout_width="wrap_content"
        android:layout_height="40sp"
        android:visibility="invisible"
        android:layout_alignParentTop="true"
        android:layout_gravity="center"
        android:layout_toRightOf="@+id/textview_cur"
        android:layout_toEndOf="@+id/textview_cur"
        android:onClick="showPopupMenu"
        android:text="@string/sample_s"
        android:textSize="18sp" />

    <Button
        android:id="@+id/button_fftlen"
        android:layout_width="wrap_content"
        android:layout_height="40sp"
        android:layout_alignParentTop="true"
        android:visibility="invisible"
        android:layout_gravity="center"
        android:layout_toRightOf="@+id/button_sample_rate"
        android:layout_toEndOf="@+id/button_sample_rate"
        android:onClick="showPopupMenu"
        android:text="@string/fftlen"
        android:textSize="18sp" />
</RelativeLayout>

<TextView
    android:id="@+id/textview_rec"
    android:layout_width="fill_parent"
    android:visibility="invisible"
    android:layout_height="wrap_content"
    android:text="@string/textview_rec_text"
    android:typeface="monospace" />
```

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center" >

    <github.bewantbe.audio_analyzer_for_android.SelectorText
        android:id="@+id/button_recording"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible"
        android:layout_margin="4dp"
        android:paddingLeft="15dp"
        android:paddingStart="15dp"
        android:tag="select"
        android:text="Mon"
        android:textSize="20sp"
        custom:items="Mon Rec" />

    <github.bewantbe.audio_analyzer_for_android.SelectorText
        android:id="@+id/spectrum_spectrogram_mode"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible"
        android:layout_margin="4dp"
        android:paddingLeft="15dp"
        android:paddingStart="15dp"
        android:tag="select"
        android:text="1D"
        android:textSize="20sp"
        custom:itemsDisplay="@string/button_spectrum_spectrogram_mode"
        custom:items="spum spam" />

    <github.bewantbe.audio_analyzer_for_android.SelectorText
        android:id="@+id/dbA"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible"
        android:layout_margin="4dp"
        android:paddingLeft="15dp"
        android:paddingStart="15dp"
        android:tag="select"
        android:text="dB"
        android:textSize="20sp"
        custom:itemsDisplay="@string/button_dbA"
        custom:items="dB dBA" />

    <!--<github.bewantbe.audio_analyzer_for_android.SelectorText-->
    <!--android:id="@+id/graph_view_mode"-->
    <!--android:layout_width="wrap_content"-->
    <!--android:layout_height="wrap_content"-->
    <!--android:layout_margin="4dp"-->
    <!--android:paddingLeft="15dp"-->
    <!--android:tag="select"-->
    <!--android:text="cursor"-->
    <!--android:textSize="20sp"-->
    <!--custom:items="cursor scale" />-->

    <github.bewantbe.audio_analyzer_for_android.SelectorText
        android:id="@+id/freq_scaling_mode"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible"
        android:layout_margin="4dp"
        android:paddingLeft="15dp"
        android:paddingStart="15dp"
        android:tag="select"
        android:text="linear"
        android:textSize="20sp"
        custom:itemsDisplay="@string/button_freq_scaling_mode"
        custom:items="linear log note" />

    <github.bewantbe.audio_analyzer_for_android.SelectorText
        android:id="@+id/run"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="invisible"
        android:layout_margin="4dp"
        android:paddingLeft="15dp"
        android:paddingStart="15dp"
        android:tag="select"
        android:text="run"
        android:textSize="20sp"
        custom:items="run stop" />

    <Button
        android:id="@+id/button_average"
        android:layout_width="wrap_content"
```

```
            android:visibility="invisible"
            android:layout_height="40sp"
            android:layout_gravity="center"
            android:onClick="showPopupMenu"
            android:text="@string/ave"
            android:textSize="18sp" />

    </LinearLayout>

</LinearLayout>
```

main.xml (portrait)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res-auto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >


    <view
        android:id="@+id/plot"
        android:visibility="invisible"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="0.32"
        class="github.bewantbe.audio_analyzer_for_android.AnalyzerGraphic"
        custom:cutoffDb="-25"
        custom:sampleRate="16000" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textview_RMS"
            android:visibility="visible"
            android:layout_width="@dimen/textview_RMS_layout_width"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:lines="2"
            android:text="@string/textview_RMS_text"
            android:typeface="monospace" />
        <TextView
            android:id="@+id/textview_cur"
            android:visibility="visible"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_toRightOf="@id/textview_RMS"
            android:layout_toEndOf="@id/textview_RMS"
            android:text="@string/textview_cur_text"
            android:typeface="monospace" />

        <TextView
            android:id="@+id/textview_peak"
            android:layout_width="fill_parent"
            android:visibility="visible"
            android:layout_height="wrap_content"
            android:layout_below="@id/textview_cur"
            android:layout_toRightOf="@id/textview_RMS"
            android:layout_toEndOf="@id/textview_RMS"
            android:text="@string/textview_peak_text"
            android:typeface="monospace" />
    </RelativeLayout>

    <TextView
        android:id="@+id/textview_rec"
        android:layout_width="fill_parent"
        android:visibility="visible"
        android:layout_height="wrap_content"
        android:text="@string/textview_rec_text"
        android:typeface="monospace" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:visibility="invisible"
        android:layout_height="50sp"
        android:gravity="center"
        android:orientation="horizontal" >

        <github.bewantbe.audio_analyzer_for_android.SelectorText
            android:id="@+id/button_recording"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="4dp"
            android:visibility="invisible"
            android:paddingLeft="15dp"
            android:paddingStart="15dp"
            android:tag="select"
            android:text="Mon"
            android:textSize="20sp"
            custom:itemsDisplay="@string/button_recording"
            custom:items="Mon Rec" />

        <Button
            android:id="@+id/button_sample_rate"
            android:layout_width="wrap_content"
            android:layout_height="@dimen/button_button_height"
            android:visibility="invisible"
            android:layout_gravity="center"
            android:onClick="showPopupMenu"
            android:text="@string/sample_s"
            android:textSize="@dimen/button_text_fontsize" />

        <Button
            android:id="@+id/button_fftlen"
            android:layout_width="wrap_content"
            android:layout_height="@dimen/button_button_height"
            android:layout_gravity="center"
            android:onClick="showPopupMenu"
            android:visibility="invisible"
            android:text="@string/fftlen"
            android:textSize="@dimen/button_text_fontsize" />

        <Button
            android:id="@+id/button_average"
            android:layout_width="wrap_content"
            android:layout_height="@dimen/button_button_height"
            android:layout_gravity="center"
            android:visibility="invisible"
            android:onClick="showPopupMenu"
            android:text="@string/ave"
            android:textSize="@dimen/button_text_fontsize" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center" >
```

```
<github.bewantbe.audio_analyzer_for_android.SelectorText
   android:id="@+id/spectrum_spectrogram_mode"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_margin="4dp"
   android:visibility="invisible"
   android:paddingLeft="15dp"
   android:paddingStart="15dp"
   android:tag="select"
   android:text="1D"
   android:textSize="20sp"
   custom:itemsDisplay="@string/button_spectrum_spectrogram_mode"
   custom:items="spum spam" />

<github.bewantbe.audio_analyzer_for_android.SelectorText
   android:id="@+id/dbA"
   android:layout_width="fill_parent"
   android:visibility="invisible"
   android:layout_height="wrap_content"
   android:layout_margin="4dp"
   android:paddingLeft="15dp"
   android:paddingStart="15dp"
   android:tag="select"
   android:text="dB"
   android:textSize="20sp"
   custom:itemsDisplay="@string/button_dbA"
   custom:items="dB dBA" />

<!--<github.bewantbe.audio_analyzer_for_android.SelectorText-->
<!--android:id="@+id/graph_view_mode"-->
<!--android:layout_width="0dp"-->
<!--android:layout_height="0dp"-->
<!--android:layout_margin="4dp"-->
<!--android:paddingLeft="15dp"-->
<!--android:tag="select"-->
<!--android:text="scale"-->
<!--android:textSize="20sp"-->
<!--custom:items="cursor scale" />-->

<github.bewantbe.audio_analyzer_for_android.SelectorText
   android:id="@+id/freq_scaling_mode"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
```

```
            android:visibility="invisible"
            android:layout_margin="4dp"
            android:paddingLeft="15dp"
            android:paddingStart="15dp"
            android:tag="select"
            android:text="linear"
            android:textSize="20sp"
            custom:itemsDisplay="@string/button_freq_scaling_mode"
            custom:items="linear log note" />

        <github.bewantbe.audio_analyzer_for_android.SelectorText
            android:id="@+id/run"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:visibility="invisible"
            android:layout_margin="4dp"
            android:paddingLeft="15dp"
            android:paddingStart="15dp"
            android:tag="select"
            android:text="run"
            android:textSize="20sp"
            custom:itemsDisplay="@string/button_run"
            custom:items="run stop" />


    </LinearLayout>

    <TextView
        android:id="@+id/testingLetter"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:text="@string/testingString"
        android:visibility="visible" />

    <Button
        android:id="@+id/toDog"
        android:visibility="invisible"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Go To Dog"></Button>

</LinearLayout>
```

main2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res-auto"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="0.32"
        android:orientation="horizontal" >

        <view
            android:id="@+id/plot"
            android:layout_width="0dp"
            android:layout_height="fill_parent"
            class="github.bewantbe.audio_analyzer_for_android.AnalyzerGraphic" />

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:orientation="vertical" >

            <github.bewantbe.audio_analyzer_for_android.SelectorText
                android:visibility="invisible"
                android:id="@+id/spectrum_spectrogram_mode"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="4dp"
                android:paddingLeft="15dp"
                android:paddingStart="15dp"
                android:tag="select"
                android:text="spum"
                android:textSize="20sp"
                custom:items="spum spam" />
        <github.bewantbe.audio_analyzer_for_android.SelectorText
            android:visibility="invisible"
            android:id="@+id/dbA"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="4dp"
            android:paddingLeft="15dp"
            android:paddingStart="15dp"
            android:tag="select"
            android:text="dB"
            android:textSize="20sp"
            custom:items="dB dBA" />

        <github.bewantbe.audio_analyzer_for_android.SelectorText
            android:id="@+id/graph_view_mode"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:visibility="invisible"
            android:layout_margin="4dp"
            android:paddingLeft="15dp"
            android:paddingStart="15dp"
            android:tag="select"
            android:text="cursor"
            android:textSize="20sp"
            custom:items="cursor scale" />

        <github.bewantbe.audio_analyzer_for_android.SelectorText
            android:visibility="invisible"
            android:id="@+id/run"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="4dp"
            android:paddingLeft="15dp"
            android:paddingStart="15dp"
            android:tag="select"
            android:text="run"
            android:textSize="20sp"
            custom:items="run stop" />

        <github.bewantbe.audio_analyzer_for_android.SelectorText
            android:visibility="invisible"
            android:id="@+id/button_recording"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="4dp"
            android:paddingLeft="15dp"
            android:paddingStart="15dp"
            android:tag="select"
            android:text="Mon"
            android:textSize="20sp"
            custom:items="Mon Rec" />
        <Button
            android:visibility="invisible"
            android:id="@+id/button_average"
            android:layout_width="wrap_content"
            android:layout_height="40sp"
            android:layout_gravity="center"
            android:onClick="showPopupMenu"
            android:text="@string/ave"
            android:textSize="18sp" />
        </LinearLayout>
      </LinearLayout>

      <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/textview_RMS"
            android:visibility="invisible"
            android:layout_width="60dp"
            android:layout_height="wrap_content"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:lines="2"
            android:text="@string/textview_RMS_text"
            android:typeface="monospace" />

        <TextView
            android:id="@+id/textview_cur"
            android:visibility="invisible"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_toRightOf="@id/textview_RMS"
            android:layout_toEndOf="@id/textview_RMS"
            android:text="@string/textview_cur_text"
            android:typeface="monospace" />

        <TextView
            android:id="@+id/textview_peak"
            android:visibility="invisible"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
        android:layout_below="@id/textview_cur"
        android:layout_toRightOf="@id/textview_RMS"
        android:layout_toEndOf="@id/textview_RMS"
        android:text="@string/textview_peak_text"
        android:typeface="monospace" />

    <Button
        android:id="@+id/button_sample_rate"
        android:layout_width="wrap_content"
        android:layout_height="40sp"
        android:visibility="invisible"
        android:layout_alignParentTop="true"
        android:layout_gravity="center"
        android:layout_toLeftOf="@+id/button_fftlen"
        android:layout_toStartOf="@+id/button_fftlen"
        android:onClick="showPopupMenu"
        android:text="@string/sample_s"
        android:textSize="18sp" />

    <Button
        android:id="@+id/button_fftlen"
        android:layout_width="wrap_content"
        android:visibility="invisible"
        android:layout_height="40sp"
        android:layout_alignParentTop="true"
        android:layout_gravity="center"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:onClick="showPopupMenu"
        android:text="@string/fftlen"
        android:textSize="18sp" />
</RelativeLayout>

<TextView
    android:id="@+id/textview_rec"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:visibility="invisible"
    android:text="@string/textview_rec_text"
    android:typeface="monospace" />

<Button
    android:id="@+id/toDog"
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Go To Dog" />

</LinearLayout>
```

todog

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/topTV"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="CHIP Scan Page"
        android:textAlignment="center"
        android:textSize="30dp"
        android:layout_marginTop="20dp"
        android:layout_alignParentTop="true"/>

    <Button
        android:id="@+id/refreshBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Refresh"
        android:layout_marginTop="20dp"
        android:layout_alignParentEnd="true"/>

    <Switch
        android:id="@+id/AttemptAuto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Attempt Autoconnect" />

    <TextView
        android:id="@+id/bottomTV"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Please turn on CHIP and select it on above table."
        android:textAlignment="center"
        android:textSize="10dp"
        android:layout_alignParentBottom="true"/>
    <Button
        android:id="@+id/toConnectionList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Connect to different dog" />

    <ListView
        android:id="@+id/connectionTable"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@id/topTV"
        android:layout_above="@id/bottomTV"></ListView>

</LinearLayout>
```

menu
info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/menu_manual"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_info_details"
        android:menuCategory="container"
        android:title="@string/menu_manual"
        android:titleCondensed="@string/menu_manual_condensed"/>

    <item
        android:id="@+id/menu_settings"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_manage"
        android:menuCategory="container"
        android:title="@string/menu_preferences"
        android:titleCondensed="@string/menu_preferences_condensed"/>

    <item
        android:id="@+id/menu_test_recorder"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_manage"
        android:menuCategory="container"
        android:title="@string/menu_test"
        android:titleCondensed="@string/menu_test_condensed"/>
    <item
        android:id="@+id/menu_view_range"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_manage"
        android:menuCategory="container"
        android:title="@string/menu_set_view"
        android:titleCondensed="@string/menu_set_view_condensed"/>

    <item
        android:id="@+id/menu_calibration"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_manage"
        android:menuCategory="container"
        android:title="Load calibration"
        android:titleCondensed="calib"/>

</menu>
```

info_red.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item
        android:id="@+id/rec_tester_std"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_info_details"
        android:menuCategory="container"
        android:title="@string/rec_tester_std"
        android:titleCondensed="info"/>

    <item
        android:id="@+id/rec_tester_support"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_info_details"
        android:menuCategory="container"
        android:title="@string/rec_tester_support"
        android:titleCondensed="info"/>

    <item
        android:id="@+id/rec_tester_all"
        android:enabled="true"
        android:icon="@android:drawable/ic_menu_info_details"
        android:menuCategory="container"
        android:title="@string/rec_tester_all"
        android:titleCondensed="info"/>

</menu>
```

values
arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string-array name="dbColorMapDescribe" translatable="false">
        <item>Hot</item>
        <item>Black-body</item>
        <item>Jet</item>
        <item>Parula (Matlab)</item>
        <item>Gray</item>
        <item>inferno (Matplotlib)</item>
        <item>magma (Matplotlib)</item>
        <item>plasma (Matplotlib)</item>
        <item>viridis (Matplotlib)</item>
    </string-array>
    <string name="dbColorMap_default" translatable="false">viridis</string>
    <string-array name="dbColorMap" translatable="false">
        <item>hot</item>
        <item>blackbody_uniform</item>
        <item>jet</item>
        <item>parula</item>
        <item>gray</item>
        <item>inferno</item>
        <item>magma</item>
        <item>plasma</item>
        <item>viridis</item>
    </string-array>
    <string name="spectrum_dbRange_default" translatable="false">-120</string>
    <string name="spectrogram_dbRange_default" translatable="false">-120</string>
    <string-array name="dbRangeArray" translatable="false">
        <item>-30</item>
        <item>-60</item>
        <item>-90</item>
        <item>-120</item>
        <item>-150</item>
    </string-array>
    <string name="spectrogram_duration_default" translatable="false">6.0</string>
    <string-array name="spectrogram_duration_array" translatable="false">
        <item>2.0</item>
        <item>4.0</item>
        <item>6.0</item>
        <item>8.0</item>
        <item>10.0</item>
    </string-array>
    <string name="spectrogram_fps_default" translatable="false">8</string>
    <string-array name="spectrogram_fps_array" translatable="false">
        <item>8</item>
        <item>12</item>
        <item>18</item>
        <item>27</item>
        <item>40</item>
        <item>60</item>
    </string-array>
    <string name="wnd_func_default" translatable="false">Hanning</string>
    <string-array name="wnd_func_names" translatable="false">
        <item>Rectangular</item>
        <item>Bartlett</item>
        <item>Hanning</item>
        <item>Blackman</item>
        <item>Blackman Harris</item>
        <item>Kaiser, a=2.0</item>
        <item>Kaiser, a=3.0</item>
        <item>Kaiser, a=4.0</item>
        <item>Flat-top</item>
        <item>Nuttall</item>
        <item>Gaussian, b=3.0</item>
        <item>Gaussian, b=5.0</item>
        <item>Gaussian, b=6.0</item>
        <item>Gaussian, b=7.0</item>
        <item>Gaussian, b=8.0</item>
    </string-array>
    <string-array name="fft_overlap_percent_describe" translatable="false">
        <item>0%</item>
        <item>50%</item>
        <item>75%</item>
        <item>87.5%</item>
    </string-array>
    <string-array name="fft_overlap_percent" translatable="false">
        <item>0.0</item>
        <item>50.0</item>
        <item>75.0</item>
        <item>87.5</item>
    </string-array>
```

```xml
<string name="fft_overlap_percent_default" translatable="false">0.0</string>
<string-array name="audio_source" translatable="false">
   <item>VOICE_RECOGNITION</item>
   <item>DEFAULT</item>
   <item>MIC</item>
   <item>VOICE_UPLINK</item>
   <item>VOICE_DOWNLINK</item>
   <item>VOICE_CALL</item>
   <item>CAMCORDER</item>
   <item>test signal 1\n\t 440Hz @ -6dB</item>
   <item>test signal 2\n\t 625Hz @ -6dB\n +1875Hz @ -12dB</item>
   <item>white noise</item>
</string-array>
<string-array name="audio_source_id" translatable="false">
   <item>6</item>
   <item>0</item>
   <item>1</item>
   <item>2</item>
   <item>3</item>
   <item>4</item>
   <item>5</item>
   <item>1000</item>
   <item>1001</item>
   <item>1002</item>
</string-array>
<string name="audio_source_id_default" translatable="false">6</string>

<string-array name="sample_rates" translatable="false">
   <item>@string/sample_rates_name</item>
   <item>16000::16000</item>
   <item>8000::8000</item>
   <item>11025::11025</item>
   <item>22050::22050</item>
   <item>32000::32000</item>
   <item>44100::44100</item>
   <item>48000::48000</item>
   <item>96000::96000</item>
   <item>192k::192000</item>
</string-array>

<string-array name="fft_len" translatable="false">
   <item>@string/fft_len_name</item>
   <item>512::512</item>
   <item>1024::1024</item>
```

```
        <item>2048::2048</item>
        <item>4096::4096</item>
        <item>8192::8192</item>
        <item>16384::16384</item>
    </string-array>
    <string-array name="fft_ave_num" translatable="false">
        <item>@string/fft_ave_num_name</item>
        <item>1::1</item>
        <item>2::2</item>
        <item>4::4</item>
        <item>8::8</item>
        <item>16::16</item>
        <item>32::32</item>
    </string-array>

</resources>
```

attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <declare-styleable name="SelectorText">
        <attr name="items" format="string" />
        <attr name="itemsDisplay" format="string" />
        <attr name="itemDelim" format="string" />
    </declare-styleable>
    <declare-styleable name="AnalyzerGraphic">
        <attr name="sampleRate" format="integer" />
        <attr name="cutoffDb" format="integer" />
    </declare-styleable>

</resources>
```

audio_source.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!--List of all standard audio source-->
    <!--In the order of ID-->
    <!--
https://developer.android.com/reference/android/media/MediaRecorder.AudioSource.html-->

    <!--                    val API-->
    <!--DEFAULT                                     0       1-->
    <!--MIC                                         1       1-->
```

```
<!--VOICE_UPLINK            2       4       CAPTURE_AUDIO_OUTPUT-->
<!--VOICE_DOWNLINK          3       4       CAPTURE_AUDIO_OUTPUT-->
<!--VOICE_CALL              4       4-->
<!--CAMCORDER               5       7-->
<!--VOICE_RECOGNITION       6       7-->
<!--VOICE_COMMUNICATION     7       11-->
<!--REMOTE_SUBMIX           8       19      CAPTURE_AUDIO_OUTPUT-->
<!--UNPROCESSED                     9       24-->

<integer-array name="std_audio_source_id">
   <item>0</item>
   <item>1</item>
   <item>2</item>
   <item>3</item>
   <item>4</item>
   <item>5</item>
   <item>6</item>
   <item>7</item>
   <item>8</item>
   <item>9</item>
</integer-array>

<string-array name="std_audio_source_name" translatable="false">
   <item>DEFAULT</item>
   <item>MIC</item>
   <item>VOICE_UPLINK</item>
   <item>VOICE_DOWNLINK</item>
   <item>VOICE_CALL</item>
   <item>CAMCORDER</item>
   <item>VOICE_RECOGNITION</item>
   <item>VOICE_COMMUNICATION</item>
   <item>REMOTE_SUBMIX</item>
   <item>UNPROCESSED</item>
</string-array>

<!--API level that this source is available-->
<integer-array name="std_audio_source_api_level">
   <item>1</item>
   <item>1</item>
   <item>4</item>
   <item>4</item>
   <item>4</item>
   <item>7</item>
   <item>7</item>
```

```
    <item>11</item>
    <item>19</item>
    <item>24</item>
  </integer-array>

  <!--Some sources not available for third-party applications.-->
  <string-array name="std_audio_source_permission" translatable="false">
    <item>""</item>
    <item>""</item>
    <item>"CAPTURE_AUDIO_OUTPUT"</item>
    <item>"CAPTURE_AUDIO_OUTPUT"</item>
    <item>""</item>
    <item>""</item>
    <item>""</item>
    <item>""</item>
    <item>"CAPTURE_AUDIO_OUTPUT"</item>
    <item>""</item>
  </string-array>

  <string-array name="std_sampling_rates" translatable="false">
    <item>8000</item>
    <item>11025</item>
    <item>16000</item>
    <item>22050</item>
    <item>32000</item>
    <item>44100</item>
    <item>48000</item>
    <item>96000</item>
    <item>192000</item>
  </string-array>
</resources>
```

dimens.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <!--List of all standard audio source-->
  <!--In the order of ID-->
  <!--
https://developer.android.com/reference/android/media/MediaRecorder.AudioSource.html-->

  <!--              val API-->
  <!--DEFAULT                         0     1-->
  <!--MIC                             1     1-->
  <!--VOICE_UPLINK              2     4     CAPTURE_AUDIO_OUTPUT-->
```

```
<!--VOICE_DOWNLINK          3       4       CAPTURE_AUDIO_OUTPUT-->
<!--VOICE_CALL              4       4-->
<!--CAMCORDER               5       7-->
<!--VOICE_RECOGNITION       6       7-->
<!--VOICE_COMMUNICATION     7       11-->
<!--REMOTE_SUBMIX           8       19      CAPTURE_AUDIO_OUTPUT-->
<!--UNPROCESSED                     9       24-->

<integer-array name="std_audio_source_id">
   <item>0</item>
   <item>1</item>
   <item>2</item>
   <item>3</item>
   <item>4</item>
   <item>5</item>
   <item>6</item>
   <item>7</item>
   <item>8</item>
   <item>9</item>
</integer-array>

<string-array name="std_audio_source_name" translatable="false">
   <item>DEFAULT</item>
   <item>MIC</item>
   <item>VOICE_UPLINK</item>
   <item>VOICE_DOWNLINK</item>
   <item>VOICE_CALL</item>
   <item>CAMCORDER</item>
   <item>VOICE_RECOGNITION</item>
   <item>VOICE_COMMUNICATION</item>
   <item>REMOTE_SUBMIX</item>
   <item>UNPROCESSED</item>
</string-array>

<!--API level that this source is available-->
<integer-array name="std_audio_source_api_level">
   <item>1</item>
   <item>1</item>
   <item>4</item>
   <item>4</item>
   <item>4</item>
   <item>7</item>
   <item>7</item>
   <item>11</item>
```

```
        <item>19</item>
        <item>24</item>
    </integer-array>

    <!--Some sources not available for third-party applications.-->
    <string-array name="std_audio_source_permission" translatable="false">
        <item>""</item>
        <item>""</item>
        <item>"CAPTURE_AUDIO_OUTPUT"</item>
        <item>"CAPTURE_AUDIO_OUTPUT"</item>
        <item>""</item>
        <item>""</item>
        <item>""</item>
        <item>""</item>
        <item>"CAPTURE_AUDIO_OUTPUT"</item>
        <item>""</item>
    </string-array>

    <string-array name="std_sampling_rates" translatable="false">
        <item>8000</item>
        <item>11025</item>
        <item>16000</item>
        <item>22050</item>
        <item>32000</item>
        <item>44100</item>
        <item>48000</item>
        <item>96000</item>
        <item>192000</item>
    </string-array>
</resources>
```

strings

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Audio Spectrum Analyzer</string>
    <string name="preferences">Preferences</string>

    <!-- Main UI -->
    <string name="textview_RMS_text">RMS:dB \n-XXX.X</string>
    <string name="textview_cur_text" >Cur :XXXXX.XHz(AX#+XX) -XXX.XdB</string>
    <string name="text_cur">Cur :</string>
    <string name="textview_peak_text">Peak:XXXXX.XHz(AX#+XX) -XXX.XdB</string>
    <string name="text_peak">Peak:</string>
    <string name="textview_rec_text">Rec: 00:00:00.0, Remain: 0000:00:00</string>
    <string name="text_rec">Rec:</string>
    <string name="text_remain">", Remain: "</string>
    <string name="sample_s">SAMPLE/S</string>
    <string name="fftlen">FFT LEN</string>
    <string name="ave">N AVE</string>
    <string name="sample_rates_name">SAMPLE/S::0</string>
    <string name="fft_len_name">FFT LEN::0</string>
    <string name="fft_ave_num_name">N AVE::0</string>
    <string name="button_freq_scaling_mode">linear::log::note</string>
    <string name="button_run">run::stop</string>
    <string name="button_dbA">dB::dBA</string>
    <string name="button_spectrum_spectrogram_mode">1D::2D</string>
    <string name="button_recording">Mon::Rec</string>

    <!-- permission dialog -->
    <string name="permission_explanation_title">Permission denied</string>
    <string name="permission_explanation_recorder">This app analyze the sound heard by *this* device. A permission of using microphone is required.</string>
    <string name="permission_explanation_write">To record the sound, A permission of using read/write is required.</string>

    <!-- Show range dialog -->
    <string name="ok">OK</string>
    <string name="cancel">Cancel</string>
    <string name="show_range_tv_fL">"Set lower and upper Frequency Bound"</string>
    <string name="show_range_tv_fH">"(%1$.2f&lt;= Hz &lt;=%2$.2f)"</string>
    <string name="show_range_tv_dBL">"Set lower and upper dB Bound"</string>
    <string name="show_range_tv_dBH">"(%1$.2f&lt;= dB &lt;=%2$.2f)"</string>
    <string name="et_freq_setting_lower_bound">frequency lower bound</string>
    <string name="et_freq_setting_upper_bound">frequency upper bound</string>
    <string name="et_db_setting_lower_bound">dB lower bound</string>
    <string name="et_db_setting_upper_bound">dB upper bound</string>
    <string name="show_range_lock">Lock Range</string>
    <string name="show_range_button_lock">Load Previous</string>

    <!-- Pop-up menu -->
    <string name="menu_manual">User Manual</string>
    <string name="menu_preferences">Preferences</string>
    <string name="menu_test">Recorder Sources Test</string>
    <string name="menu_set_view">View Range Setting</string>
    <string name="menu_manual_condensed">Manual</string>
    <string name="menu_preferences_condensed">Preferences</string>
    <string name="menu_test_condensed">Test</string>
    <string name="menu_set_view_condensed">Set View</string>

    <!-- InfoRecActivity (Test Recorder Sources) -->
    <string name="text_view_info_log_0">Testing AudioRecord properties..</string>
    <string name="title_activity_info_rec">Info of Input Audio</string>
    <string name="rec_tester_std">Standard Sources</string>
    <string name="rec_tester_support">Supported Sources</string>
    <string name="rec_tester_all">All Sources</string>
    <string name="rec_tester_hint1">"(Only test MONO, 16BIT format)\n"</string>
    <string name="rec_tester_col1">"\nSampleRate MinBuf    (Time)\n"</string>
    <string name="rec_tester_col2">"%1$5d Hz  %2$4d Bytes (%3$4.1f ms)\n"</string>
    <string name="rec_tester_error1">" Hz  ERROR\n"</string>
    <string name="rec_tester_hint2">"\n--- Audio Source Test ---\n"</string>
    <string name="rec_tester_col3">"\nSource: %1$s\n"</string>
    <string name="rec_tester_col3_row1">"%1$5d Hz  "</string>
    <string name="rec_tester_col3_succeed">"succeed"</string>
    <string name="rec_tester_col3_error1">"Illegal Argument"</string>
    <string name="rec_tester_col3_error2">"read 0 byte"</string>
    <string name="rec_tester_col3_error3">"fail to read sample"</string>
    <string name="rec_tester_col3_error4">"fail to initialize"</string>
    <string name="rec_tester_col3_hint1">"(source alter: %1$s)"></string>
    <string name="rec_tester_col3_end">".\n"</string>
    <string name="True">true</string>

    <!-- preferences -->
    <string name="preference_generalSettings">General Settings</string>
    <string name="preference_keepScreenOn_1">Prevent display from turning off</string>
    <string name="preference_keepScreenOn_2">Keep screen on</string>
    <string name="preference_audioSource_1">Audio source for spectrum</string>
    <string name="preference_audioSource_2">Audio source</string>
    <string name="preference_windowFunction_1">Window function for (local) short-time Fourier transform (STFT)</string>
    <string name="preference_windowFunction_2">Window Function</string>
    <string name="preference_windowOverlap_1">Higher percentage for finer time resolution, lower to save CPU</string>
    <string name="preference_windowOverlap_2">Time window overlap</string>
    <string name="preference_spectrumAppearance">Spectrum appearance</string>
    <string name="preference_showLines_1">Use lines instead of bars (area) for spectrum</string>
    <string name="preference_showLines_2">Spectrum uses lines</string>
    <string name="preference_spectrumRange_1">Show range of spectrum in dB scale</string>
    <string name="preference_spectrumRange_2">Spectrum range scale</string>
    <string name="preference_spectrogramAppearance">Spectrogram appearance</string>
```

```xml
    <string name="preference_spectrogramShifting_1">The whole spectrogram shifts on a rolling basis over time</string>
    <string name="preference_spectrogramShifting_2">Spectrogram shifting</string>
    <string name="preference_spectrogramTimeAxis_1">Turns on/off time axis label</string>
    <string name="preference_spectrogramTimeAxis_2">Show time axis</string>
    <string name="preference_spectrogramShowFreqAlongX_1">So time axis is Y axis</string>
    <string name="preference_spectrogramShowFreqAlongX_2">Frequency as X axis</string>
    <string name="preference_spectrogramSmoothRender_1">Smoothly render the spectrogram</string>
    <string name="preference_spectrogramSmoothRender_2">Smooth render</string>
    <string name="preference_spectrogramColorMap_1">Choose color of dB to map</string>
    <string name="preference_spectrogramColorMap_2">Color Map</string>
    <string name="preference_spectrogramRange_1">Show range of spectrogram in dB scale</string>
    <string name="preference_spectrogramRange_2">Spectrogram range</string>
    <string name="preference_spectrogramDuration_1">Show duration of spectrogram (when average number is set to 1)</string>
    <string name="preference_spectrogramDuration_2">Spectrogram duration</string>
    <string name="preference_spectrogramFPS_1">Higher for smoother graphic, lower to save battery.</string>
    <string name="preference_spectrogramFPS_2">Limit Refresh rate</string>
    <string name="preference_miscellaneous">Miscellaneous</string>
    <string name="preference_warnOverrun_1">Show warning if audio buffer enters overrun state</string>
    <string name="preference_warnOverrun_2">Overrun Warning</string>
    <string name="preference_developerSetting">Developer Settings</string>
    <string name="preference_spectrogramLogPlotMethod_1">Replot spectrogram only while zooming (Saves battery)</string>
    <string name="preference_spectrogramLogPlotMethod_2">Log plot: Replot on zoom</string>
    <string name="testingString">testCar</string>

</resources>
```

II. CODE FOR ASR

Data.py

```
import sugartensor as tf
import numpy as np
import csv
import string



# default data path
_data_path = 'asset/data/'

#
# vocabulary table
#

# index to byte mapping
index2byte = ['<EMP>', ' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
              'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q',
              'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']

# byte to index mapping
byte2index = {}
for i, ch in enumerate(index2byte):
    byte2index[ch] = i

# vocabulary size
voca_size = len(index2byte)


# convert sentence to index list
def str2index(str_):

    # clean white space
    str_ = ' '.join(str_.split())
    # remove punctuation and make lower case
    str_ = str_.translate(None, string.punctuation).lower()

    res = []
    for ch in str_:
        try:
            res.append(byte2index[ch])
        except KeyError:
            # drop OOV
            pass
    return res


# convert index list to string
def index2str(index_list):
    # transform label index to character
    str_ = ''
    for ch in index_list:
        if ch > 0:
            str_ += index2byte[ch]
        elif ch == 0:  # <EOS>
            break
    return str_


# print list of index list
def print_index(indices):
    for index_list in indices:
        #print(index2str(index_list))
        return(index2str(index_list))


# real-time wave to mfcc conversion function
@tf.sg_producer_func
def _load_mfcc(src_list):

    # label, wave_file
    label, mfcc_file = src_list

    # decode string to integer
    label = np.fromstring(label, np.int)

    # load mfcc
    mfcc = np.load(mfcc_file, allow_pickle=False)

    # speed perturbation augmenting
    mfcc = _augment_speech(mfcc)

    return label, mfcc
```

```
def _augment_speech(mfcc):

    # random frequency shift ( == speed perturbation effect on MFCC )
    r = np.random.randint(-2, 2)

    # shifting mfcc
    mfcc = np.roll(mfcc, r, axis=0)

    # zero padding
    if r > 0:
        mfcc[:r, :] = 0
    elif r < 0:
        mfcc[r:, :] = 0

    return mfcc


# Speech Corpus
class SpeechCorpus(object):

    def __init__(self, batch_size=16, set_name='train'):

        # load meta file
        label, mfcc_file = [], []
        with open(_data_path + 'preprocess/meta/%s.csv' % set_name) as csv_file:
            reader = csv.reader(csv_file, delimiter=',')
            for row in reader:
                # mfcc file
                mfcc_file.append(_data_path + 'preprocess/mfcc/' + row[0] + '.npy')
                # label info ( convert to string object for variable-length support )
                label.append(np.asarray(row[1:], dtype=np.int).tostring())

        # to constant tensor
        label_t = tf.convert_to_tensor(label)
        mfcc_file_t = tf.convert_to_tensor(mfcc_file)

        # create queue from constant tensor
        label_q, mfcc_file_q \
            = tf.train.slice_input_producer([label_t, mfcc_file_t], shuffle=True)

        # create label, mfcc queue
        label_q, mfcc_q = _load_mfcc(source=[label_q, mfcc_file_q],
                                     dtypes=[tf.sg_intx, tf.sg_floatx],
                                     capacity=256, num_threads=64)

        # create batch queue with dynamic pad
        batch_queue = tf.train.batch([label_q, mfcc_q], batch_size,
                                     shapes=[(None,), (20, None)],
                                     num_threads=64, capacity=batch_size*32,
                                     dynamic_pad=True)

        # split data
        self.label, self.mfcc = batch_queue
        # batch * time * dim
        self.mfcc = self.mfcc.sg_transpose(perm=[0, 2, 1])
        # calc total batch count
        self.num_batch = len(label) // batch_size

        # print info
        tf.sg_info('%s set loaded.(total data=%d, total batch=%d)'
                   % (set_name.upper(), len(label), self.num_batch))
```

Model.py

```
import sugartensor as tf


num_blocks = 3     # dilated blocks
num_dim = 128      # latent dimension


#
# logit calculating graph using atrous convolution
#
def get_logit(x, voca_size):

    # residual block
    def res_block(tensor, size, rate, block, dim=num_dim):

        with tf.sg_context(name='block_%d_%d' % (block, rate)):

            # filter convolution
            conv_filter = tensor.sg_aconv1d(size=size, rate=rate, act='tanh', bn=True,
name='conv_filter')
```

```
            # gate convolution
            conv_gate = tensor.sg_aconv1d(size=size, rate=rate,  act='sigmoid', bn=True,
name='conv_gate')

            # output by gate multiplying
            out = conv_filter * conv_gate

            # final output
            out = out.sg_conv1d(size=1, dim=dim, act='tanh', bn=True, name='conv_out')

            # residual and skip output
            return out + tensor, out

    # expand dimension
    with tf.sg_context(name='front'):
        z = x.sg_conv1d(size=1, dim=num_dim, act='tanh', bn=True, name='conv_in')

    # dilated conv block loop
    skip = 0  # skip connections
    for i in range(num_blocks):
        for r in [1, 2, 4, 8, 16]:
            z, s = res_block(z, size=7, rate=r, block=i)
            skip += s

    # final logit layers
    with tf.sg_context(name='logit'):
        logit = (skip
                 .sg_conv1d(size=1, act='tanh', bn=True, name='conv_1')
                 .sg_conv1d(size=1, dim=voca_size, name='conv_2'))

    return logit
```

Preprocess.py

```
import numpy as np
import pandas as pd
import glob
import csv
import librosa
import scikits.audiolab
import data
import os
import subprocess



# data path
_data_path = "asset/data/"


#
# process VCTK corpus
#

def process_vctk(csv_file):

    # create csv writer
    writer = csv.writer(csv_file, delimiter=',')

    # read label-info
    df = pd.read_table(_data_path + 'VCTK-Corpus/speaker-info.txt', usecols=['ID'],
                       index_col=False, delim_whitespace=True)

    # read file IDs
    file_ids = []
    for d in [_data_path + 'VCTK-Corpus/txt/p%d/' % uid for uid in df.ID.values]:
        file_ids.extend([f[-12:-4] for f in sorted(glob.glob(d + '*.txt'))])

    for i, f in enumerate(file_ids):

        # wave file name
        wave_file = _data_path + 'VCTK-Corpus/wav48/%s/' % f[:4] + f + '.wav'
        fn = wave_file.split('/')[-1]
        target_filename = 'asset/data/preprocess/mfcc/' + fn + '.npy'
        if os.path.exists( target_filename ):
            continue
        # print info
        print("VCTK corpus preprocessing (%d / %d) - '%s']" % (i, len(file_ids), wave_file))

        # load wave file
        wave, sr = librosa.load(wave_file, mono=True, sr=None)

        # re-sample ( 48K -> 16K )
        wave = wave[::3]

        # get mfcc feature
        mfcc = librosa.feature.mfcc(wave, sr=16000)

        # get label index
        label = data.str2index(open(_data_path + 'VCTK-Corpus/txt/%s/' % f[:4] + f + '.txt').read())

        # save result ( exclude small mfcc data to prevent ctc loss )
        if len(label) < mfcc.shape[1]:
            # save meta info
            writer.writerow([fn] + label)
            # save mfcc
            np.save(target_filename, mfcc, allow_pickle=False)


#
# process LibriSpeech corpus
#

def process_libri(csv_file, category):

    parent_path = _data_path + 'LibriSpeech/' + category + '/'
    labels, wave_files = [], []

    # create csv writer
    writer = csv.writer(csv_file, delimiter=',')

    # read directory list by speaker
    speaker_list = glob.glob(parent_path + '*')
    for spk in speaker_list:

        # read directory list by chapter
        chapter_list = glob.glob(spk + '/*/')
        for chap in chapter_list:

            # read label text file list
            txt_list = glob.glob(chap + '/*.txt')
            for txt in txt_list:
                with open(txt, 'rt') as f:
                    records = f.readlines()
                    for record in records:
                        # parsing record
                        field = record.split('-')  # split by '-'
                        speaker = field[0]
                        chapter = field[1]
                        field = field[2].split()  # split field[2] by ' '
                        utterance = field[0]  # first column is utterance id

                        # wave file name
                        wave_file = parent_path + '%s/%s/%s-%s-%s.flac' % \
                                                  (speaker, chapter, speaker, chapter, utterance)
                        wave_files.append(wave_file)

                        # label index
                        labels.append(data.str2index(' '.join(field[1:])))  # last column is text label

    # save results
    for i, (wave_file, label) in enumerate(zip(wave_files, labels)):
        fn = wave_file.split('/')[-1]
        target_filename = 'asset/data/preprocess/mfcc/' + fn + '.npy'
        if os.path.exists( target_filename ):
            continue
        # print info
        print("LibriSpeech corpus preprocessing (%d / %d) - '%s']" % (i, len(wave_files), wave_file))

        # load flac file
        wave, sr, _ = scikits.audiolab.flacread(wave_file)

        # get mfcc feature
        mfcc = librosa.feature.mfcc(wave, sr=16000)

        # save result ( exclude small mfcc data to prevent ctc loss )
        if len(label) < mfcc.shape[1]:
            # filename

            # save meta info
            writer.writerow([fn] + label)

            # save mfcc
            np.save(target_filename, mfcc, allow_pickle=False)


#
# process TEDLIUM corpus
#
```

```
def convert_sph( sph, wav ):
    """Convert an sph file into wav format for further processing"""
    command = [
        'sox','-t','sph', sph, '-b','16','-t','wav', wav
    ]
    subprocess.check_call( command ) # Did you install sox (apt-get install sox)

def process_ted(csv_file, category):

    parent_path = _data_path + 'TEDLIUM_release2/' + category + '/'
    labels, wave_files, offsets, durs = [], [], [], []

    # create csv writer
    writer = csv.writer(csv_file, delimiter=',')

    # read STM file list
    stm_list = glob.glob(parent_path + 'stm/*')
    for stm in stm_list:
        with open(stm, 'rt') as f:
            records = f.readlines()
            for record in records:
                field = record.split()

                # wave file name
                wave_file = parent_path + 'sph/%s.sph.wav' % field[0]
                wave_files.append(wave_file)

                # label index
                labels.append(data.str2index(' '.join(field[6:])))

                # start, end info
                start, end = float(field[3]), float(field[4])
                offsets.append(start)
                durs.append(end - start)

    # save results
    for i, (wave_file, label, offset, dur) in enumerate(zip(wave_files, labels, offsets, durs)):
        fn = "%s-%.2f" % (wave_file.split('/')[-1], offset)
        target_filename = 'asset/data/preprocess/mfcc/' + fn + '.npy'
        if os.path.exists( target_filename ):
            continue
        # print info
        print("TEDLIUM corpus preprocessing (%d / %d) - '%s-%.2f]" % (i, len(wave_files),
wave_file, offset))
        # load wave file
        if not os.path.exists( wave_file ):
            sph_file = wave_file.rsplit('.',1)[0]
            if os.path.exists( sph_file ):
                convert_sph( sph_file, wave_file )
            else:
                raise RuntimeError("Missing sph file from TedLium corpus at %s"%(sph_file))
        wave, sr = librosa.load(wave_file, mono=True, sr=None, offset=offset, duration=dur)

        # get mfcc feature
        mfcc = librosa.feature.mfcc(wave, sr=16000)

        # save result ( exclude small mfcc data to prevent ctc loss )
        if len(label) < mfcc.shape[1]:
            # filename

            # save meta info
            writer.writerow([fn] + label)

            # save mfcc
            np.save(target_filename, mfcc, allow_pickle=False)


#
# Create directories
#
if not os.path.exists('asset/data/preprocess'):
    os.makedirs('asset/data/preprocess')
if not os.path.exists('asset/data/preprocess/meta'):
    os.makedirs('asset/data/preprocess/meta')
if not os.path.exists('asset/data/preprocess/mfcc'):
    os.makedirs('asset/data/preprocess/mfcc')



#
# Run pre-processing for training
#

# VCTK corpus
csv_f = open('asset/data/preprocess/meta/train.csv', 'w')
process_vctk(csv_f)
```

```
csv_f.close()

# LibriSpeech corpus for train
csv_f = open('asset/data/preprocess/meta/train.csv', 'a+')
process_libri(csv_f, 'train-clean-360')
csv_f.close()

# TEDLIUM corpus for train
csv_f = open('asset/data/preprocess/meta/train.csv', 'a+')
process_ted(csv_f, 'train')
csv_f.close()

#
# Run pre-processing for validation
#

# LibriSpeech corpus for valid
csv_f = open('asset/data/preprocess/meta/valid.csv', 'w')
process_libri(csv_f, 'dev-clean')
csv_f.close()

# TEDLIUM corpus for valid
csv_f = open('asset/data/preprocess/meta/valid.csv', 'a+')
process_ted(csv_f, 'dev')
csv_f.close()

#
# Run pre-processing for testing
#

# LibriSpeech corpus for test
csv_f = open('asset/data/preprocess/meta/test.csv', 'w')
process_libri(csv_f, 'test-clean')
csv_f.close()

# TEDLIUM corpus for test
csv_f = open('asset/data/preprocess/meta/test.csv', 'a+')
process_ted(csv_f, 'test')
csv_f.close()
```

Recognize.py

```
# -*- coding: utf-8 -*-
import sugartensor as tf
import numpy as np
import librosa
from model import *
import data
import os
from collections import Counter




# set log level to debug
tf.sg_verbosity(10)

#
# hyper parameters
#

batch_size = 1     # batch size

#
# inputs
#

# vocabulary size
voca_size = data.voca_size

# mfcc feature of audio
x = tf.placeholder(dtype=tf.sg_floatx, shape=(batch_size, None, 20))

# sequence length except zero-padding
seq_len = tf.not_equal(x.sg_sum(axis=2), 0.).sg_int().sg_sum(axis=1)

# encode audio feature
logit = get_logit(x, voca_size=voca_size)

# ctc decoding
decoded, _ = tf.nn.ctc_beam_search_decoder(logit.sg_transpose(perm=[1, 0, 2]), seq_len,
merge_repeated=False)

# to dense tensor
```

```
y = tf.sparse_to_dense(decoded[0].indices, decoded[0].dense_shape, decoded[0].values) + 1

#
# regcognize wave file
#

# command line argument for input wave file path
tf.sg_arg_def(file=('', 'speech wave file to recognize.'))

# load wave file
wav, _ = librosa.load(tf.sg_arg().file, mono=True, sr=16000)
# get mfcc feature
mfcc = np.transpose(np.expand_dims(librosa.feature.mfcc(wav, 16000), axis=0), [0, 2, 1])

# run network
with tf.Session() as sess:

    # init variables
    tf.sg_init(sess)

    basedir = '/media/sf_ec327file/'
    os.chdir(basedir)
    # restore parameters
    saver = tf.train.Saver()
    saver.restore(sess, tf.train.latest_checkpoint('ASR'))
    #saver.restore(sess, tf.train.latest_checkpoint('asset/train'))
    basedir = '/home/eugeneychsiao/Desktop/speech to text/'
    os.chdir(basedir)

    # run session
    label = sess.run(y, feed_dict={x: mfcc})

    # print label
    #data.print_index(label)
    prediction = data.print_index(label).replace(" ", "")
    print(prediction)
    word = input("What word is this?").replace(" ", "")
    strings = []
    strings.append(prediction)
    strings.append(word)
    counter = 0
    #print(strings[0])
    #print(strings[1])
    if len(strings[0]) >= len(strings[1]):
        shortest = len(strings[1])
    else:
        shortest = len(strings[0])

    for i in range(0,shortest):
        if strings[0][i] == strings[1][i]:
            counter += 1

    percentsimilarity = counter/shortest
    print(percentsimilarity)
```

<u>Test.py</u>

```
import sugartensor as tf
from data import SpeechCorpus, voca_size
from model import *
import numpy as np
from tqdm import tqdm





# set log level to debug
tf.sg_verbosity(10)

# command line argument for set_name
tf.sg_arg_def(set=('valid', "'train', 'valid', or 'test'.  The default is 'valid'"))
tf.sg_arg_def(frac=(1.0, "test fraction ratio to whole data set. The default is 1.0(=whole set)"))


#
# hyper parameters
#

# batch size
batch_size = 16

#
# inputs
```

```
#

# corpus input tensor ( with QueueRunner )
data = SpeechCorpus(batch_size=batch_size, set_name=tf.sg_arg().set)

# mfcc feature of audio
x = data.mfcc
# target sentence label
y = data.label

# sequence length except zero-padding
seq_len = tf.not_equal(x.sg_sum(axis=2), 0.).sg_int().sg_sum(axis=1)

#
# Testing Graph
#

# encode audio feature
logit = get_logit(x, voca_size=voca_size)

# CTC loss
loss = logit.sg_ctc(target=y, seq_len=seq_len)

#
# run network
#

with tf.Session(config=tf.ConfigProto(allow_soft_placement=True)) as sess:

    # init variables
    tf.sg_init(sess)

    # restore parameters
    saver = tf.train.Saver()
    saver.restore(sess, tf.train.latest_checkpoint('asset/train'))

    # logging
    tf.sg_info('Testing started on %s set at global step[%08d].' %
               (tf.sg_arg().set.upper(), sess.run(tf.sg_global_step())))

    with tf.sg_queue_context():

        # create progress bar
        iterator = tqdm(range(0, int(data.num_batch * tf.sg_arg().frac)), total=int(data.num_batch *
tf.sg_arg().frac),
                        initial=0, desc='test', ncols=70, unit='b', leave=False)

        # batch loop
        loss_avg = 0.
        for _ in iterator:

            # run session
            batch_loss = sess.run(loss)

            # loss history update
            if batch_loss is not None and \
                    not np.isnan(batch_loss.all()) and not np.isinf(batch_loss.all()):
                loss_avg += np.mean(batch_loss)

        # final average
        loss_avg /= data.num_batch * tf.sg_arg().frac

    # logging
    tf.sg_info('Testing finished on %s.(CTC loss=%f)' % (tf.sg_arg().set.upper(), loss_avg))
```

Train.py

```
import sugartensor as tf
from data import SpeechCorpus, voca_size
from model import *




# set log level to debug
tf.sg_verbosity(10)



#
# hyper parameters
#

batch_size = 16    # total batch size

#
```

```
# inputs
#

# corpus input tensor
data = SpeechCorpus(batch_size=batch_size * tf.sg_gpus())

# mfcc feature of audio
inputs = tf.split(data.mfcc, tf.sg_gpus(), axis=0)
# target sentence label
labels = tf.split(data.label, tf.sg_gpus(), axis=0)

# sequence length except zero-padding
seq_len = []
for input_ in inputs:
    seq_len.append(tf.not_equal(input_.sg_sum(axis=2), 0.).sg_int().sg_sum(axis=1))


# parallel loss tower
@tf.sg_parallel
def get_loss(opt):
    # encode audio feature
    logit = get_logit(opt.input[opt.gpu_index], voca_size=voca_size)
    # CTC loss
    return logit.sg_ctc(target=opt.target[opt.gpu_index], seq_len=opt.seq_len[opt.gpu_index])

#
# train
#
tf.sg_train(lr=0.0001, loss=get_loss(input=inputs, target=labels, seq_len=seq_len),
            ep_size=data.num_batch, max_ep=50)
```

```
%% FmapsChild.m
%% Takes all young child sound .wav files and finds the formant values
%% All older children, age 8 and above, were considered in the ADULT category
close all;
clear all;
clc;
%% absolute paths to the sound directories
Ball='C:\Users\bu\Documents\ELAT\VowelSpace\ChildrenVocalizationsTestModel\Ball - BEST';
Daddy='C:\Users\bu\Documents\ELAT\VowelSpace\ChildrenVocalizationsTestModel\Daddy';
Jeep='C:\Users\bu\Documents\ELAT\VowelSpace\ChildrenVocalizationsTestModel\Jeep';
No='C:\Users\bu\Documents\ELAT\VowelSpace\ChildrenVocalizationsTestModel\No';
%% make a cell - to "vectorize" directory path variables
sound_dirs={Ball, Daddy,Jeep, No};
num_sounds=length(sound_dirs);
words=cell(1,num_sounds);
%% peel off the directory name from the path
for i=1:num_sounds;
    path=sound_dirs{i};
[path, fname, ext] = fileparts(path);
    %This peels off the last folder from path.
    %The ext must be preserved in case the folder name has a dot it it.
opendir = strcat(fname, ext);
words{i}=opendir;
end
%% Each Directory is a cell. --- Within each cell you keep the struct array returned by the dir↙
function.
data=cell(length(num_sounds),3); %preallocate space given # sound types
disp(data);
    % Each row corresponds to a different sound
    % First column = directory name
    % Second column = "files struct" for .wav files
    % Third column = "formant data struct"
%% fill the data_structure cell columns 1 & 2, with directory names and struct of .wav files.
for i=1:num_sounds;
    wavPATH=fullfile(sound_dirs{i}, '*.wav'); % fullfile(Ball, '*.wav'); <-- gives specified path
    wavLIST=dir(wavPATH); %saves .wav files in a struct call wavs_in_dir
    data{i,2}=wavLIST;
    data{i,1}=words{i};
end %all raw data saved into data_structure
%% data_structure{i,3} holds the processed data <-- Formants
for i=1:num_sounds;
     NUMwavs=numel(data{i,2});
     ith_file=data{i,2};
     formants=struct('F1',[],'maxPxx1', [], 'F2',[],'maxPxx2',[], 'F3', [], 'maxPxx3', []);
     for k=1:NUMwavs;
     [y Fs]= audioread(fullfile(sound_dirs{i},data{i,2}(k).name));
     %need to define the window frame to match the size of the signal vector
        w=ones(1, length(y));
        %nfft points in discrete Fourier Transform (DFT)
        nfft=length(y);
        [pxx,f]= periodogram(y, w,nfft,Fs);
        %% now find the range of frequencies
        % L1= 200 - 800 Hz
        % L2= 800 - 1800 Hz
        % L3= 1800 - 3500 Hz
        %% find Level 1 indices for frequency values between 200 - 800 Hz
        fmin1=200; fmax1=800;
        L1_ind= find((f>=fmin1)&(f<=fmax1));%L1_ind are indicies that are within 200-800
        %% use L1_ind to find the relevant vectors
            chunkOpsdFreqs1= f(L1_ind); %chunk_O_psd_Frequencies_1 [=] chunkOpsdFreqs1 gives all↙
the freq values that are [200Hz, 800Hz]
            chunkOpsd1 = pxx(L1_ind); %chunkOpsd1 gives all the pxx values
            peak1=max(chunkOpsd1);
            %want to find the max intensity value this index will give the max intensity which↙
corresponds to formant 1
            %indices the same, because [pxx, f] makes pxx and f one to one element wise
            F_index1=find(chunkOpsd1==peak1);
            Formant1=chunkOpsdFreqs1(F_index1);
                    %% find Level 2 indices for frequency values between 800 - 1800 Hz
        Fmin2=800; Fmax2=1800;
        L2_ind=find((f>=Fmin2) & (f<=Fmax2));
        %% use L2_ind to find the relevant vectors
        chunkOpsd2= pxx(L2_ind);
        chunkOpsdFreqs2= f(L2_ind);
            peak2=max(chunkOpsd2);
            F_index2=find(chunkOpsd2==peak2);
            Formant2=chunkOpsdFreqs2(F_index2);
        %% find Level 3 indices for frequency values between 1800 - 3500 Hz
        Fmin3=1800; Fmax3=3500;
        L3_ind=find((f>=Fmin3) & (f<=Fmax3));
        %% find Level 3 indices for frequency values between 800 - 1800 Hz
        chunkOpsd3= pxx(L3_ind);
        chunkOpsdFreqs3= f(L3_ind);
            peak3=max(chunkOpsd3);
            F_index3=find(chunkOpsd3==peak3);
            Formant3=chunkOpsdFreqs3(F_index3);
        %% Store all values in formants struct
        % each row has its own formants struct
        formants(k).F1= Formant1; %Hz Values
        formants(k).F2= Formant2;
        formants(k).F3= Formant3;
        formants(k).maxPxx1=peak1; %power spectral density (y) values correlated to the formants,↙
which are Hz values.
        formants(k).maxPxx2=peak2;
        formants(k).maxPxx3=peak3;
     end
        data{i,3}=formants;
end

%% Get the F#s from the structure in the cell
Plot3D=figure(1);
Plot2D_12=figure(2);
Plot2D_23=figure(3);
Plot2D_13=figure(4);
colors= ['b', 'k', 'r', 'g', 'm'];
for i=1:num_sounds;
    num_pts=length(data{i,3});
           for n=1:num_pts;
            [F1pts]=zeros(1,num_pts);
            [F2pts]=zeros(1,num_pts);
            [F3pts]=zeros(1,num_pts);
           end
           for n=1:num_pts;
            F1pts(n)=data{i,3}(n).F1;
            F2pts(n)=data{i,3}(n).F2;
            F3pts(n)=data{i,3}(n).F3;
           end
    figure(Plot3D); scatter3(F1pts, F2pts, F3pts, colors(i)); hold on; xlim([200, 900]); ylim↙
([700, 1800]);
    figure(Plot2D_12); scatter(F1pts,F2pts, colors(i), 'o'); hold on; xlim([200,900]); ylim([700,↙
1800]);
    figure(Plot2D_13); scatter(F1pts,F3pts, colors(i), 'o'); hold on; xlim([200,900]); ylim↙
([1700,3600]);
    figure(Plot2D_23); scatter(F2pts,F3pts, colors(i), 'o'); hold on; xlim([700, 1800]); ylim↙
([1700, 3600]);
end

figure(Plot3D);
title('F1 vs F2 vs F3');
grid on; xlabel('F1 (Frequency) in Hz'); ylabel('F2 (Frequency) in Hz'); zlabel('F3 (Frequency)↙
in Hz');
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});

figure(Plot2D_13);
title ('F1 vs F3');
grid on; xlabel('F1 (Frequency) in Hz'); ylabel('F3 (Frequency) in Hz');
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});

figure(Plot2D_12);
title ('F1 vs F2');
grid on; xlabel('F1 (Frequency) in Hz'); ylabel('F2 (Frequency) in Hz');
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});

figure(Plot2D_23);
title('F2 vs F3');
grid on; xlabel('F2 (Frequency) in Hz'); ylabel('F3 (Frequency) in Hz');
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});
```

```
%% FmapsAdults.m
%% Provides 3D plots of F1:F2:F3 Formants
%% and F1:F2, F2:F3, F1:F3 <-- 2D Scatter Plots
%% Approach Exactly the same, but contains Adult Sound Files
%% Children Age 8, considered in "ADULT" sound category
%%  All these sounds were given a 100% scoring in terms of intelligibility

%% absolute paths to the sound directories
Ball='C:\Users\bu\Documents\ELAT\VowelSpace\Adults\Ball - Adult';
Daddy='C:\Users\bu\Documents\ELAT\VowelSpace\Adults\Daddy - Adult';
Jeep='C:\Users\bu\Documents\ELAT\VowelSpace\Adults\Jeep - Adult';
No='C:\Users\bu\Documents\ELAT\VowelSpace\Adults\No - Adult';
%% make a cell - to "vectorize" directory path variables
sound_dirs={Ball, Daddy,Jeep, No};
num_sounds=length(sound_dirs);
words=cell(1,num_sounds);
%% peel off the directory name from the path
for i=1:num_sounds;
    path=sound_dirs{i};
[path, fname, ext] = fileparts(path);
    %This peels off the last folder from path.
    %The ext must be preserved in case the folder name has a dot it it.
opendir = strcat(fname, ext);
words{i}=opendir;
end
%% Each Directory is a cell. --- Within each cell you keep the struct array returned by the dir↙
function.
data=cell(length(num_sounds),3); %preallocate space given # sound types
disp(data);
    % Each row corresponds to a different sound
    % First column = directory name
    % Second column = "files struct" for .wav files
    % Third column = "formant data struct"
%% fill the data_structure cell columns 1 & 2, with directory names and struct of .wav files.
for i=1:num_sounds;
    wavPATH=fullfile(sound_dirs{i}, '*.wav'); % fullfile(Ball, '*.wav'); <-- gives specified path
    wavLIST=dir(wavPATH); %saves .wav files in a struct call wavs_in_dir
    data{i,2}=wavLIST;
    data{i,1}=words{i};
end %all raw data saved into data
%% data{i,3} holds the processed data <-- Formants
for i=1:num_sounds;
     NUMwavs=length(data{i,2});
     ith_file=data{i,2};
     formants=struct('F1',[],'maxPxx1', [], 'F2',[],'maxPxx2',[], 'F3', [], 'maxPxx3', []);
     for k=1:NUMwavs;
     [y Fs]= audioread(fullfile(sound_dirs{i},data{i,2}(k).name));
     %need to define the window frame to match the size of the signal vector
        w=ones(1, length(y));
        %nfft points in discrete Fourier Transform (DFT)
        nfft=length(y);
        [pxx,f]= periodogram(y, w,nfft,Fs);
        %% now find the range of frequencies
        % L1= 200 - 800 Hz
        % L2= 800 - 1800 Hz
        % L3= 1800 - 3500 Hz
        %% find Level 1 indices for frequency values between 200 - 800 Hz
        fmin1=200; fmax1=800;
        L1_ind= find((f>=fmin1)&(f<=fmax1));%L1_ind are indicies that are within 200-800
        %% use L1_ind to find the relevant vectors
            chunkOpsdFreqs1= f(L1_ind); %chunk_O_psd_Frequencies_1 [=] chunkOpsdFreqs1 gives all↙
the freq values that are [200Hz, 800Hz]
            chunkOpsd1 = pxx(L1_ind); %chunkOpsd1 gives all the pxx values
            peak1=max(chunkOpsd1);
            %want to find the max intensity value this index will give the max intensity which↙
corresponds to formant 1
            %indices the same, because [pxx, f] makes pxx and f one to one element wise
            F_index1=find(chunkOpsd1==peak1);
            Formant1=chunkOpsdFreqs1(F_index1);
        %% find Level 2 indices for frequency values between 800 - 1800 Hz
        Fmin2=800; Fmax2=1800;
        L2_ind=find((f>=Fmin2) & (f<=Fmax2));
        %% use L2_ind to find the relevant vectors
        chunkOpsd2= pxx(L2_ind);
        chunkOpsdFreqs2= f(L2_ind);
            peak2=max(chunkOpsd2);
            F_index2=find(chunkOpsd2==peak2);
            Formant2=chunkOpsdFreqs2(F_index2);
        %% find Level 3 indices for frequency values between 1800 - 3500 Hz
        Fmin3=1800; Fmax3=3500;
        L3_ind=find((f>=Fmin3) & (f<=Fmax3));
        %% find Level 3 indices for frequency values between 800 - 1800 Hz
        chunkOpsd3= pxx(L3_ind);
        chunkOpsdFreqs3= f(L3_ind);
            peak3=max(chunkOpsd3);
            F_index3=find(chunkOpsd3==peak3);
            Formant3=chunkOpsdFreqs3(F_index3);
        %% Store all values in formants struct
        % each row has its own formants struct
        formants(k).F1= Formant1; %Hz Values
        formants(k).F2= Formant2;
        formants(k).F3= Formant3;
        formants(k).maxPxx1=peak1; %power spectral density (y) values correlated to the formants,↙
which are Hz values.
        formants(k).maxPxx2=peak2;
        formants(k).maxPxx3=peak3;
     end
        data{i,3}=formants;
end

%% Get the F#s from the structure in the cell
Plot3D=figure(5);
Plot2D_12=figure(6);
Plot2D_23=figure(7);
Plot2D_13=figure(8);
colors= ['b', 'k', 'r', 'g']; %color character vector to be able to distinguish sounds
for i=1:num_sounds;
    num_pts=length(data{i,3});
           %Preallocate Vectors to correct size
           %size of .wav variables variable <-- reason for this approach
           for n=1:num_pts;
            [F1pts]=zeros(1,num_pts);
            [F2pts]=zeros(1,num_pts);
            [F3pts]=zeros(1,num_pts);
           end

           %fill the vectors with all the formant values from all the .wav
           %files
           for n=1:num_pts;
            F1pts(n)=data{i,3}(n).F1;
            F2pts(n)=data{i,3}(n).F2;
            F3pts(n)=data{i,3}(n).F3;
           end
    % plot the points in a scatter plot
    figure(Plot3D); scatter3(F1pts, F2pts, F3pts, colors(i)); hold on; xlim([200, 900]); ylim↙
([700, 1800]);
    figure(Plot2D_12); scatter(F1pts,F2pts, colors(i), 'o'); hold on; xlim([200,900]); ylim([700,↙
1800]);
    figure(Plot2D_13); scatter(F1pts,F3pts, colors(i), 'o'); hold on; xlim([200,900]); ylim↙
([1700,3600]);
    figure(Plot2D_23); scatter(F2pts,F3pts, colors(i), 'o'); hold on; xlim([700, 1800]); ylim↙
([1700, 3600]);
end

figure(Plot3D);
title('F1 vs F2 vs F3');
grid on; xlabel('F1 (Frequency in Hz)'); ylabel('F2 (Frequency in Hz)'); zlabel('F3 (Frequency in↙
Hz)');
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});

figure(Plot2D_13);
title ('F1 vs F3');
grid on; xlabel('F1 (Frequency in Hz)'; ylabel('F3 (Frequency in Hz)';
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});

figure(Plot2D_12);
title ('F1 vs F2');
grid on; xlabel('F1 (Frequency in Hz)'); ylabel('F2 (Frequency) in Hz');
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});

figure(Plot2D_23);
title('F2 vs F3');
grid on; xlabel('F2 (Frequency in Hz)'); ylabel('F3 (Frequency in Hz)');
legend(data{1,1}, data{2,1}, data{3,1}, data{4,1});
```