Audio Engineering Society

# Convention Paper 10037

Presented at the 145th Convention
2018 October 17–20, New York, NY, USA



# The new Dynamics Processing Effect in Android Open Source Project

Ricardo A. Garcia

*Google, 1600 Amphitheatre Pkwy, Mountain View, CA 94043*

Correspondence should be addressed to Ricardo Garcia (attn:rago) (rago @ google.com)

**ABSTRACT**

The Android "P" Audio Framework's new Dynamics Processing Effect (DPE) in Android Open Source Project (AOSP), provides developers with controls to fine-tune the audio experience using several stages of equalization, multi-band compressors and linked limiters. The API allows developers to configure the DPE's multi-channel architecture to exercise real-time control over thousands of audio parameters. This talk additionally discusses the design and use of DPE in the recently announced Sound Amplifier accessibility service for Android, and outlines other uses for acoustic compensation and hearing applications.

## 1 Introduction

The Android Open Source Project (AOSP) includes built-in audio effects for developers and Original Equipment Manufacturers (OEMs) to apply to their applications and devices. These effects include 5-band equalization, reverberation, bass boost, and loudness enhancement. Until Android "P," there were limited opportunities for developers to customize the behavior of these effects. The new Dynamics Processing Effect offers developers thousands of new parameters with which to configure the effect architecture and control its real-time performance [1].

Developed to satisfy the audio post-processing needs of users looking to modify the audio energy distribution over multiple frequency ranges and exercise finer control over dynamic ranges, the DPE's architecture supports 4 stages per audio channel: Pre Equalizer, Multi-band Compressor, Post Equalizer and Limiter. Each stage type can be configured to include an arbitrary number of bands. Band cutoff and additional parameters can also be set by the developer.

The DPE's multichannel and link limiter capability allows developers to synchronously limit multiple related channels.

The DPE's high-resolution floating-point data path allows for high dynamic range processing with high Signal to Noise Ratio (SNR).

This flexible architecture enables applications to quickly create a template of the desired architecture. Developers can access parameters of interest and have them set at initialization time. They can also be actively controlled during app execution. Flexibility in the design enables different Digital Signal Processing (DSP) code styles. For instance, some algorithms make heavier use of equalization as opposed to Multi-Band compression, thus allowing developers to easily adapt their existing logic to the new DPE.

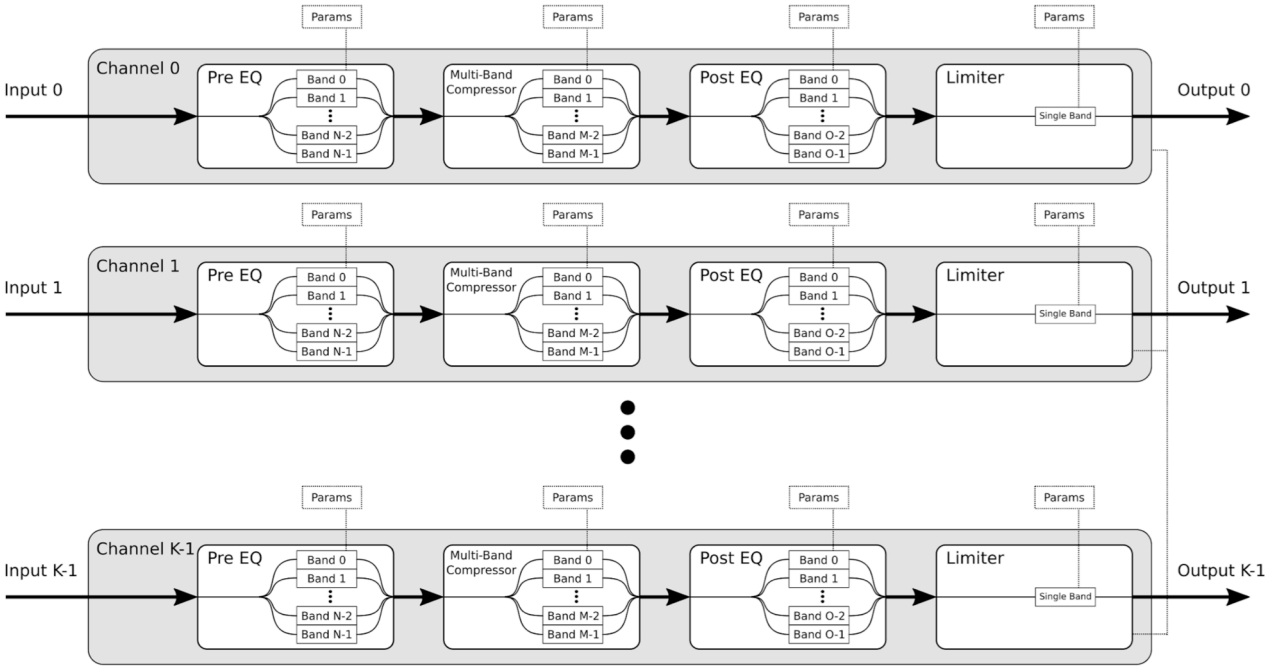



Figure 1. Dynamics Processing Effect architecture.

This paper discusses the new Android sound-processing API, and illustrates its use with a multi-purpose application called *Sound Amplifier*.

## 2 Audio Effects in Android Operating System

Android Open Source Project (AOSP) has defined a suite of audio effects to be available in all systems [2].

Default implementations of these effects can be replaced by the Original Equipment Manufacturers (OEMs) as long as they comply with their Application Program Interface (API) and pass all the compatibility tests required [3].

Developers can create instances of the effects in their applications and use the APIs to configure and control the behavior of the effect [4], thus modifying the audio in their applications.

Although AOSP only distributes a software implementation of the default effects, it is also possible for OEMs to define a hardware-accelerated implementation of the effects. The OS would then decide whether to instantiate hardware or software-based effects according to its own ruleset which could include; stream type, audio routing, or energy consumption.

## 3 Dynamics Processing Effect

### 3.1 History

The development of the Dynamics Processing Effect was heavily influenced by the desire to create a *Sound Amplifier* to enhance a user's listening experience in a variety of conditions. Although this kind of solution could be implemented as a stand-alone application by allowing the app to process the audio itself, the development team considered the idea of implementing the core part of the solution as a built-in effect into the operating system, with enough flexibility to be usable by this application and a whole family of applications that require similar processing needs.

Equalization and Multi-Band Compressors are used routinely in sound processing to change the

frequency response and/or the dynamics of signals, in applications like speaker tuning, hearing enhancement, playback personalization [5].

The development team reached out to developers and talked to many researchers and users to collect and tabulate their needs, to offer a powerful but flexible API to satisfy as many of their needs as possible.

### 3.2 Architecture

The Dynamics Processing Effect is composed of multiple channels, each of them sharing the same stage structure (see Figure 1)

Each channel has 4 stages: Pre Equalizer (PreEQ), Multi-Band Compressor (MBC), Post Equalizer (PostEQ) and Limiter.

Each stage type can be initialized to have a certain number of bands, or to be turned on or off. Stages of different types can have different number of bands, but all stages of the same type in different channels are required to have the same number of bands. The cutoff frequency for each band is also configurable.

The limiter is single band, but developers may link

```
DynamicsProcessing.Config.Builder builder =
        new DynamicsProcessing.Config.Builder(
                DynamicsProcessing.VARIANT_FAVOR_FREQUENCY_RESOLUTION,
                1 /*dummy channels*/,
                true /*Use preEq*/,  8 /*preEq bands*/,
                true /*Use mbc*/,    8 /*mbc bands*/,
                true /*Use postEq*/, 8 /*postEqBands*/,
                true /*Use Limiter*/);
builder.setPreferredFrameDuration(10);  //ms
Config config = builder.build();
```

Figure 2. Configuring the DPE with the builder.

```
Mbc mbc = config.getChannelByChannelIndex(0).getMbc();
for (int i = 0; i < 8; i++) {
    MbcBand mbcBand = mbc.getBand(i);
    mbcBand.setAttackTime(50); //ms
    mbcBand.setReleaseTime(100); //ms
    mbcBand.setRatio(2.0f); //1:2.0
    mbcBand.setThreshold(-50.0f); //dBFS
    mbcBand.setKneeWidth(0); //dB
    mbcBand.setNoiseGateThreshold(-80.0f); //dBFS
    mbcBand.setExpanderRatio(3.0f); //3.0:1
    mbcBand.setPreGain(0.0f);  //no Pre-gain
    mbcBand.setPostGain(0.0f); //no Post-gain
}
DynamicsProcessing myDPE = new DynamicsProcessing(0 /*priority*/,
        sessionId /* sessionId from MediaPlayer, audio track, etc */,
        config);
```

Figure 3. Further customization of MBC bands and DPE instantiation.

limiters to other channels. When one of the channels in the link group is triggered, all the channels will react in the same way.

### 3.3 Android API

The DPE is included in Android SDK 28 and above. To use it, just import `android.media.audiofx.DynamicsProcessing` into your application.

Figure 2 shows the use of a Builder [6] to define a simple DPE, with all four stages enabled, and with 8 bands in each type of stage. An optional parameter allows to specify a preferred frame duration in milliseconds.

Figure 3 shows a more detailed configuration of a DPE by iterating on each one of the MBC bands and setting default values to their parameters. Then, the DPE object is created and attached to an existing `AudioTrack` or `MediaPlayer` via the *sessionId* number.

It is important to notice that thresholds are given in Decibels Full Scale ($dB_{FS}$), where 0 dB is the Root Mean Square (RMS) value of a full-scale square wave, or -3 $dB_{FS}$ for full scale sine wave [7]. See section 5 for a more detailed analysis of digital levels.

An important part of the DPE API is the ability to access and modify the parameters in each stage at will.
A visualization of the API modules and the DPE stages is shown in Figure 4. Each channel has 4 modules: PreEQ, MBC, PostEQ and Limiter. For each module there are global parameters and for some modules there are bands. Each band has parameters that can also be accessed and controlled.

#### *3.3.1 Equalizer*

The equalizer module allows the developer to change the level in one or multiple frequency bands. Figure 5 shows a graphical representation of multiband equalizer and API methods available. Note that bands can be specified to be any width desired.

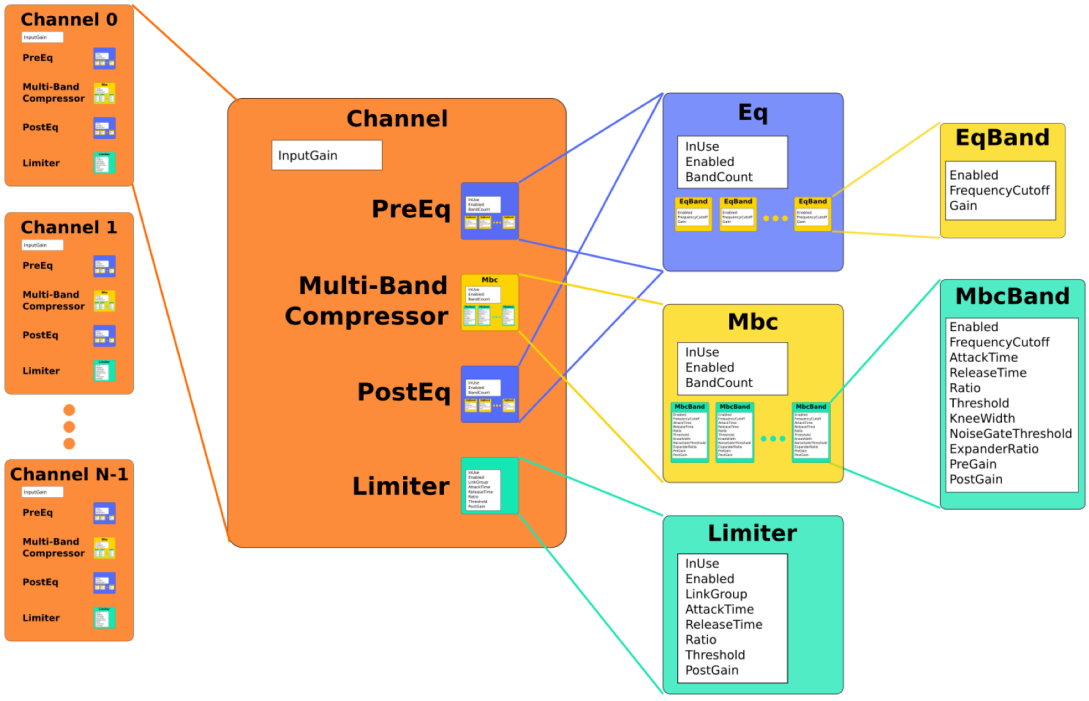
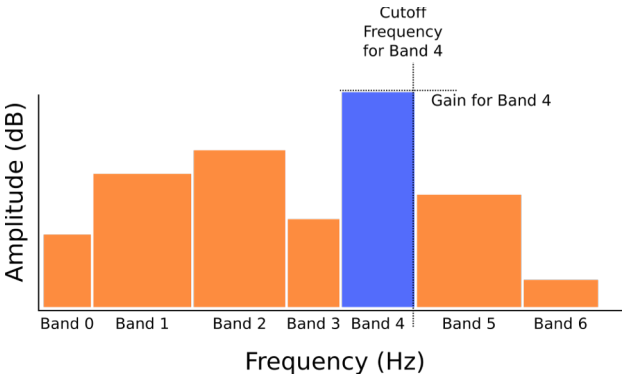



Figure 4. API Modules and relationship with DPE stages, bands and a subset of the available parameters.




```
public float getGain()
public void setGain(float gain)
```

Figure 5. Multi-Band equalizer and access methods. Parameters for Band 4 are highlighted.

### 3.3.2 *Multi-Band Compressor (MBC)*

An audio compressor reduces the level of loud sounds, or increases the level of soft sounds, thus *compressing* the signal's dynamic range. A multi-band compressor does this but allows the user to specify different parameters for different frequency ranges or bands.

The MBC provides controls for a noise gate and expander, which are useful to reduce the level of soft sounds, effectively reducing the noise floor of the signal; and a knee-width parameter for the compressor, to allow for smoother transitions between compress - linear zones.

Figure 6 illustrates the parameters available for the

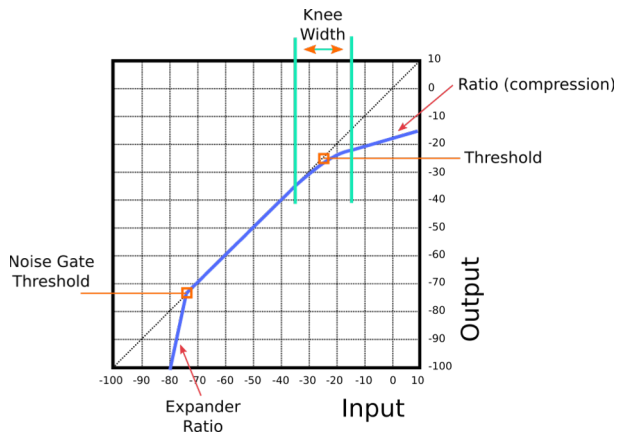



Figure 6. Parameters for a single band in the MBC.

MBC API to be controlled per band. Figure 7 shows some of the API calls available per band.

### 3.3.3 *Limiter*

A limiter module is very similar to a compressor, but usually employed to make sure signals don't get louder than a certain threshold. One example of their use is to ensure signals are lower than the maximum level, otherwise they could damage or distort in the next stages of the signal chain (e.g. to protect an audio amplifier).

An important parameter is the LinkGroup. The Limiter stage is the only stage that can depend on the information from other channels. If multiple limiters belong to the same LinkGroup, once any of the limiters is engaged: the level detected is ABOVE the threshold, ALL the limiters are engaged to the same level. Conversely, if multiple limiters are engaged, the one with the highest attenuation range will be used in all the linked channels. This can be useful to maintain the spatial image of a multichannel signal, if one of the channels engages the limiter protection: all the channels will be attenuated in the same fashion.

Figure 8 shows the parameters and a graphical description for the Limiter stage. Note this stage is always single band.

## 3.4 AOSP Implementation Notes

AOSP defines the Dynamics Processing Effect API for instantiating and controlling its parameters. The API allows to specify a preferred frame duration in milliseconds and to specify a preferred type of implementation: Time domain to favor time resolution or Frequency domain to favor frequency resolution. The availability for the preferred values is not guaranteed and it depends on the actual implementations available in a specific system.

The default software implementations shipped in AOSP have been designed to be robust and with high quality for the majority of applications, leaving room for OEM and third party vendors to create and offer their implementations [1].

```
public float getAttackTime()          public void setAttackTime(float attackTime)    //ms
public float getReleaseTime()         public void setReleaseTime(float releaseTime) //ms
public float getRatio()               public void setRatio(float ratio)
public float getThreshold()           public void setThreshold(float threshold)      //dBFS
public float getKneeWidth()           public void setKneeWidth(float kneeWidth)      //dB
public float getNoiseGateThreshold()  public void setNoiseGateThreshold(float noiseGateThreshold) //dBFS
public float getExpanderRatio()       public void setExpanderRatio(float expanderRatio)
public float getPreGain()             public void setPreGain(float preGain)    //dB
public float getPostGain()            public void setPostGain(float postGain) //dB
```

Figure 7. Access methods per band in the MBC.

```
public int getLinkGroup()
public float getAttackTime()
public float getReleaseTime()
public float getRatio()
public float getThreshold()
public float getPostGain()

public void setLinkGroup(int linkGroup)
public void setAttackTime(float attackTime)
public void setReleaseTime(float releaseTime)
public void setRatio(float ratio)
public void setThreshold(float threshold)
public void setPostGain(float postGain)
```

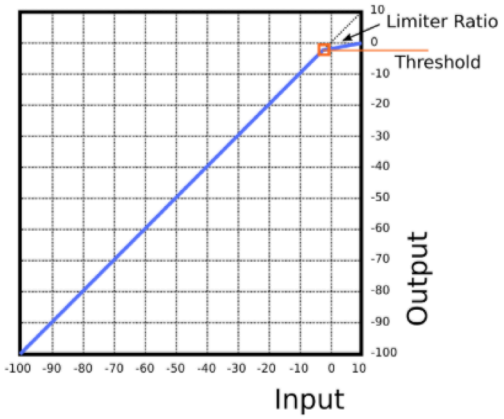



Figure 8. Access methods and Parameters for the Limiter stage.

## 4 Sound Amplifier

The *Sound Amplifier* is an application that improves the listening experience in a variety of situations and makes heavy use of the DPE. The goal is to enable to use an Android device and a set of headphones to amplify the sound around the user and easily tune it to enhance the listening experience.

The average threshold of hearing in quiet has been extensively measured [8][9]. Sounds that are quieter than this threshold are not "heard" or perceived by the listener. The hearing threshold can shift due to many circumstances: illness, environmental conditions, listening equipment, etc.

When a hearing threshold shift occurs, sounds that are softer (or quieter) than the threshold, can't be perceived by the listener, or make the sound very difficult to be understood.

Figure 9 shows a depiction of the threshold in quiet, and how the threshold might shift due to different circumstances. If a sound of interest lies in part below the shifted thresholds, the listening experience can be improved by amplifying the sound to be above the threshold of hearing.
But, doing this just by adding energy in some frequencies might make parts of the sound "too loud", thus, some way of reducing the level of these loud sounds is welcome.

The Sound Amplifier makes use of the DPE and the multiband compressor and equalization to amplify sounds and make sure the level remains in a comfortable range (not too quiet, not too loud).

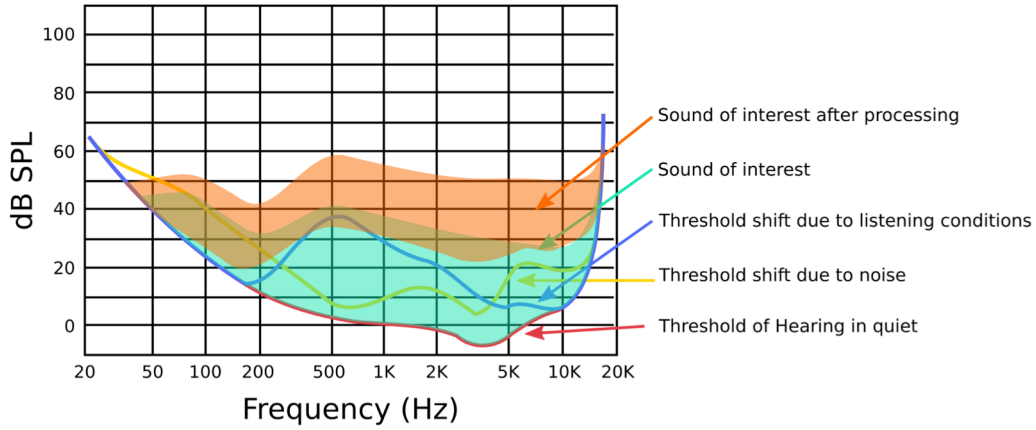



Figure 9. Hearing thresholds and shifts. Sound of interest and suggested processing.

## 5 About Digital Levels and the Analog World

The thresholds in the DPE are given in Decibels Full Scale ($dB_{FS}$), where 0 dB is the RMS value of a full-scale square wave, and will correspond to -3 $dB_{FS}$ for full scale sine wave as seen in Figure 10 [7]. The "Full Scale" in this reference is sometimes misleading. If the format of the digital signal is fixed point (e.g. 16 bits), louder signals will be clipped or distorted. But if the format is floating point, although the max expected amplitude value is 1.0, there is headroom with values greater than 1.0, without any distortion.

Converting from "real world" analog sound pressure levels (SPL) to digital domain is not standardized, and depends heavily on the device and its transducers.

### 5.1 Input/Output level scaling

In a digital system, the sound from the real world is captured by a microphone and converted to digital by an Analog to Digital Converter (ADC). Figures 11.a and 11.b show the input and output of a digital system, and how the real world levels relate to the digital world levels. The scaling factors A and B vary from system to system, and sometimes are different within a system depending on the actual configuration used at the time of capture or reproduction.

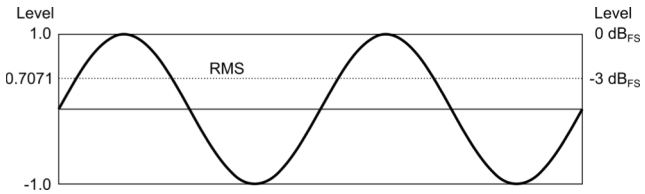



Figure 10. Amplitude levels for sine tone with peak value of 1.0 and its relationship with $dB_{FS}$ scale.

### 5.2 Input/Output level scaling for Android devices

#### *5.2.1 Input Level Conversion*

Android devices are required to pass the Compatibility Definition Document (CDD) [3] to be considered compatible. Many functional aspects of Android are defined in the CDD, including some expected levels for the different microphone sources.

We present here the expected levels and how they were used as anchors to estimate the input level used for the Sound Amplifier in Section 4.

For a 1 KHz tone, the CDD expectations for the microphone input level are shown in Table 1.

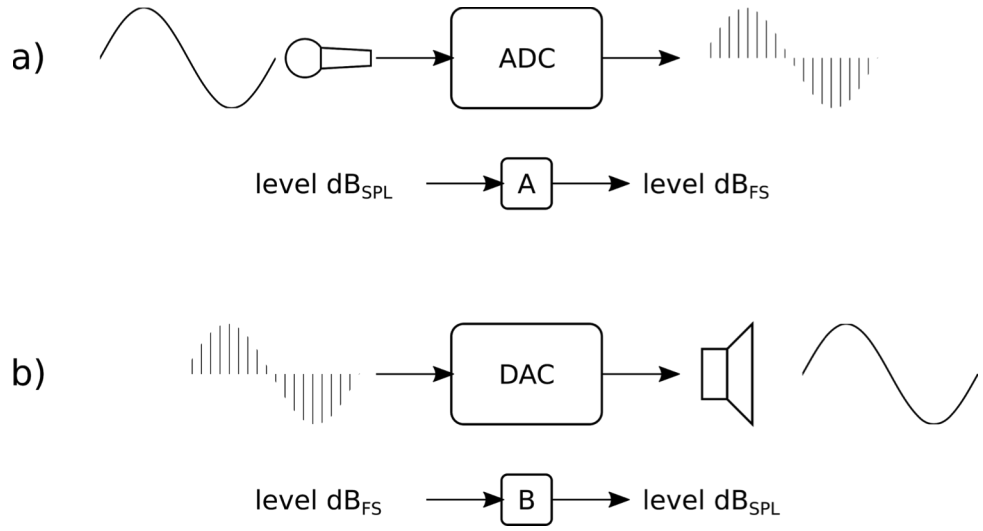



Figure 11. Input (a) and Output (b) digital systems and their level scaling. a) shows the analog signal converted to digital by the Analog to Digital Converter (ADC). The level measured on the analog world is given in $dB_{SPL}$, and can be converted to digital scale by factor A to $dB_{FS}$. Conversely, b) shows the Digital to Analog Converter (DAC). The level measured on the digital device is given in $dB_{FS}$ and can be converted to analog scale by factor B to $dB_{SPL}$.

| **Recording Source** | **Input level Anchor** | **Digital Level** | **Conversion Factor "A"** |
|---|---|---|---|
| VOICE_RECOGNITION | 90 $dB_{SPL}$ | -22.35 $dB_{FS}$ [10] | -112.35 |
| UNPROCESSED | 94 $dB_{SPL}$ | -36 $dB_{FS}$ [11] | -126 |

Table 1. CDD expected levels for a 1 KHz tone.

Figure 12 shows the hearing threshold curve plotted in both scales $dB_{SPL}$ and $dB_{FS}$ for a recording using VOICE_RECOGNITION recording source. Thresholds should be adjusted to this scale if they want to keep they relationship with the analog world.

$$A=\begin{cases}-112.35 & \text{VOICE\_RECOGNITION} \\ -126 & \text{UNPROCESSED}\end{cases} \quad (1)$$

For other recording inputs (e.g. headset), the CDD recommends to follow the same specifications set for built-in microphones and VOICE_RECOGNITION recording source [10].

### *5.2.2 Output Level Conversion*

Output levels from digital devices are heavily dependent on the output device calibration. Headphones output in Android follows the EN 50332-2 ITU Specification [12][13].

The EN 50332-2 test signal "Program Simulation Noise" has an RMS level of -13 $dB_{FS}$, and the specification says this is equivalent to 100 $dB_{SPL}$ (for a simulated load of 32 ohms)

$$B = \{113 \quad 32\ \Omega \text{ load} \quad (2)$$

### *5.2.3 End to end measurements*

These conversion factors were verified using an end-to-end test.

A custom recording and loopback app was created and run in a Pixel Android phone for several iterations. The measurement setup consisted of an Android device reproducing a 1 KHz sine tone and amplified through a low distortion speaker. The

| Tone Frequency | Input [air] | Digital Level | Output [load] | End to End difference |
|---|---|---|---|---|
| 1 KHz | 100 $dB_{SPL}$ | -13.6 $dB_{FS}$ | 97.9 $dB_{SPL}$ | -2.1 dB |

Table 2. End to End average level difference

level of the tone was measured adjacent to the built-in microphone in the Android device using a standard SPL meter (REED R8050, Hi mode, $dB_A$ weighting).

The $dB_{FS}$ level of the RMS was displayed on the application, and the unmodified sound was looped back to the output of the device (unmodified). A 32 ohms simulated headphones load was connected to the headphone output of the device, and the output voltage was measured.

The estimated SPL at the load was computed assuming the EN 50332-2 reference of 150 mV equivalence to 100 $dB_A$, which is equivalent to 100 $dB_{SPL}$ for the 1 KHz tone.

The average result is shown in Table 2.

The differences on the end to end experiments are considered by the authors to be within the acceptable range of variability for consumer electronics audio equipment.

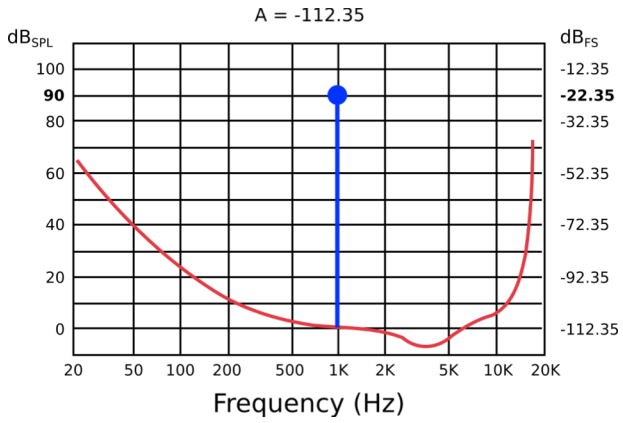



Figure 12. Frequency plot showing the equivalence between $dB_{SPL}$ and $dB_{FS}$ for the Hearing Threshold in quiet and for a 1 KHz tone, with 90 $dB_{SPL}$ <-> -22.35 $dB_{FS}$ (for A = -112.35)

## 6 Conclusions and Next Steps

A discussion about the new Dynamics Processing Effect in Android was conducted, with some level of detail thought useful for developers and scientists looking to work with audio processing algorithms in mobile platforms.

The author wants to thank all the people in cross disciplinary teams that made this project possible in Google: Android Audio Frameworks Team, Sound Understanding Team and Taipei Team; and the external researchers including Harry Levitt and Helen Simon, as well as partners and customers that gave early feedback to our APIs. Special thanks to Dick Lyon and Malcolm Slaney from the Sound Understanding team for their valuable advice, and to Brian Kemler that pushed the envelope to align all the teams.

The author and teams will continue development of applications similar to the *Sound Amplifier*, and make a call to the community to use the DPE in their products and research, and to give feedback.

## References

[1] "DynamicsProcessing | Android Developers." https://developer.android.com/reference/android/media/audiofx/DynamicsProcessing. Accessed 25 Jul. 2018.

[2] "AudioEffect | Android Developers." https://developer.android.com/reference/android/media/audiofx/AudioEffect. Accessed 25 Jul. 2018.

[3] "Compatibility Definition Document - Android Open Source Project." 5 Dec. 2017, https://source.android.com/compatibility/cdd. Accessed 25 Jul. 2018.

[4] J. J. Annuzzi, S. Conder, and L. Darcey, *Advanced Android application development*. Addison-Wesley, 2015.

[5] W. Pirkle, Designing Audio Effect Plug-Ins in C++: with digital audio signal processing theory. Focal Press, 2013.

[6] "Config Builder | DynamicsProcessing | Android Developers. "https://developer.android.com/reference/android/media/audiofx/DynamicsProcessing.Config.Builder. Accessed 25 Jul. 2018.

[7] K. C. Pohlmann, *Principles of digital audio*. New York: McGraw-Hill, 2011

[8] H. Fastl and E. Zwicker, *Psychoacoustics: facts and models*. Berlin: Springer, 2005.

[9] R. F. Lyon, *Human and machine hearing extracting meaning from sound*. Cambridge: Cambridge University Press, 2017.

[10] "Android 8.1 Compatibility Definition | Android Open Source Project | Capture for Voice Recognition", https://source.android.com/compatibility/android-cdd#5_4_2_capture_for_voice_recognition. Accessed 25 Jul. 2018.

[11] "Android 8.1 Compatibility Definition | Android Open Source Project | Capture for Unprocessed", https://source.android.com/compatibility/android-cdd#https://source.android.com/compatibility/android-cdd#5_11_capture_for_unprocessed. Accessed 25 Jul. 2018.

[12] BS EN 50332-2(Headphones and Earphones associated with portable audio equipment,-Maximum sound pressure level methodology and limit considerations, part-2:Matching of sets with headphones if either or both are offers separately ) 2003

[13] "3.5 mm Headset Jack: Device Specification | Android Open Source ...." 27 Jul. 2017, https://source.android.com/devices/accessories/headset/jack-headset-spec. Accessed 25 Jul. 2018.