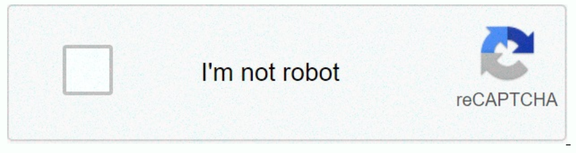

Continue

Audio mixer android library

I'm developing a karaoke app and I've been dealing with sounds from android. I need an opinion on how I can record a single song from a recorded sound and another song to play in the background. Thanks to high-performance audio apps, it often requires more functions than the simple ability to play or record audio. They need real-time behavior that responds. Some common use cases include: digital audio workstations, Synthesizers, drum machines, music, learning, karaoke apps, DJ apps, mixing, audio effects, video/audio conferencing, this section explains the general principles of reducing audio latency. It also provides audio sampling advice to help you choose the best sample rate and consider the pros and cons of using floating point numbers to display your audio data. The rest of the section describes two libraries that are available for writing high-performance audio applications: OpenSL ES. As an Android-specific use of the OpenSL ES™ API requirements from the Khronos group, OpenSL ES is not recommended for new design. App developers and middleware providers should target either Oboe or AAudio as the original audio interface. More resources to learn more Take advantage of the following resources: Codelabs video preview, getting started with Oboe Best Practices for Android Audio (Google I/O '17), the content and code snippets on this page are subject to the license described in the Java content license as a registered trademark of Oracle and/or its affiliates, last update 2020-08-17 UTC. Different depending on the target platform: Linux, MacOS, iOS: OpenAL Supported formats: Windows ogg+vorbis: XAudio2 Supported Formats: ogg + vorbis Android: SLES Supported formats: ogg+vorbis JavaScript, Asm.js: WebAudio Supported formats: mp3 (and ogg + vorbis in some browsers) Usage: Import sound.sound on a given (android): Var activity: jobject # You should get a reference to the activity from somewhere. Activity = androidGetActivity() # for example, if you use sdl initSoundEngineWithActivity (activity) var snd: Audio when assigned (android): # Supported URL layouts: android_asset, snd file = newSoundWithURL(android_asset://testfile.ogg) # Relative path to asset folder elif defined (js): snd=newSoundWithURL (testfile.ogg) # URL may be absolute or audio is loaded asynchronously, others: # Supported URL layouts: snd file = newSoundWithURL(file:// & getAppDir() & /testfile.ogg) snd.play() performance subject It is especially true for The performance you want to sound as smooth as possible. In this article, I'll see the different options you can use in your high-performance audio app on Android, but first we need to understand the motivation behind high-performance audio. Why does it matter? What are we trying to solve? What exactly are we talking about? A high-performance audio app can be an example of karaoke app, music production or synthetic machine learning app. Frankly, it can be anything, but they all are all together in one goal. - Minimal latency and enhanced quality Latency is the time it takes for a signal to travel through the system, so to have a responsive user interface and a good user experience, latency should be as low as possible. The best Android devices currently on the market can go as low as 20 ms for back-to-back latency. Back-to-back latency is the sum of input latency, app processing time and output latency. This 20ms latency is called PRO latency, it also has low latency, which guarantees 45ms latency, and you can check it in use if the current device supports one of these two features. Finally, before we see high-performance audio options, minimum digital audio vocabulary: PCM — Pulse code adjustment is the standard method in computer science to convert analog signals into digital agents. During PCM, continuous analog signals are recorded and converted to small audio samples regularly, sampling rate —the speed at which each sample is generated. It is usually 44,1 or 48 kHz, which means pcm produces 44 100 or 48 000 samples per second from analog signals. Sample size - or bit depth is the maximum amplitude of the sample shown in bits. For example, an 8-bit depth, which can display only 256 levels (2^8) of amplitude, has sound quality and a resolution lower than the standard 16-bit depth, which can store amplitude. 65536 (2^{16}) channel — can be either mono (1 channel) or stereo (2 channels). The SoundPool class, the SoundPool class, can be used to create sample collections from existing resources that are preloaded into memory. In addition to basic operations, SoundPool has several good features such as: setting the maximum number of streams that can be played at a time from a single SoundPoolPrioritisation of streaming playback, adjusting the playback rate, adjusting from 0.5 to 2.0Pausing, and playing per sounddPool. As a convenient tool, if you want to play and manage short audio, which often includes repetitive sounds contained in your APK or in files, I often use SoundPool to play low latency UI or audio games, AudioTrack/AudioRecordAudioTrack and AudioRecord are the lowest apis you can access while still using Java or Kotlin AudioTrack. They were both added early in the Android API Level 3 and provided a convenient way to work with raw audio data. I'll focus on AudioTrack, since this article is mostly about audio playback, but don't hesitate to explore AudioRecord with AudioTrack, it can work under two modes: fixed mode is better for short audio that requires frequent playback and has minimal latency similar to SoundPool, but compared to SoundPool, you can modify or update raw audio data. In streaming mode, you can play audio that is too large to fit in memory due to the length or appearance of the sound to be played or received or created instantly. I think AudioTrack is the perfect introductory API to start working with raw audio on Android, with all the high performance features available, but still in the Java OpenSL ES ES layer, a cross-platform audio API that accelerates with hardware optimized for embedded systems such as mobile devices. The only use of Android is part of Android NDK, but you need to note that there are some limitations compared to the standard OpenSL ES requirements and may increase the existing code performance a little, if you haven't written from scratch, this API is not intended to be used to write pure C/C++ applications, it is like an API, the only way, because it is not expected to have a full-featured API, which can be used to perform shared libraries, such as with the iOS team, but with a small catchment of opensl ES applications on Android does not guarantee performance improvements compared to the platform. Although OpenSL ES can be used from the Android API level 9, significant improvements to audio output latency are added later in Android API Level 17 (Android 4.2), and these are only specific devices, which means you need to check at runtime if the device supports low latency. It is not ideal, but the number of low latency devices is growing steadily, and to this day there are many devices on the market that already support low latency. So OpenSL ES may be good for you if you want to share the code between platforms or you're familiar with it, then AAudioAAudio is the new Android C API launched on Android Oreo, it's a traditional Android API designed for high-performance audio applications that require low latency. Some things that Android has been missing for a long time. You can write or read from the AAudio audio stream in your application, and AAudio ensures that the data transmitted is moving between the application and the audio input or output on your Android device. The stream is defined by the following components: audio device mode. Audio FormatAudio devices can be either a source or a sink for continuous digital audio streams. Don't confuse this terminology with real devices such as phones or watches that are running your application. Sharing mode can be Streams have exclusive access to their audio devices. While sharing means that AAudio mixes all streams assigned to the same device. Audio formats are words for all standard features used in digital audio that streams can have, such as sample rates, sample formats, and samples per frame. By default, the audio stream has a default performance mode that balances latency and energy savings. You can improve low latency AAUDIO_PERFORMANCE_MODE_LOW_LATENCY special modes, which are useful for very interactive applications such as musical instruments. I prefer to try AAudio, but since AAudio has been introduced on Android Oreo and is retrospectively incompatible, I believe I can't use it in production applications. Fortunately, there's a new native library solution by Google that solves AAudio's retrospective compatibility solution, it works with API 16 and above, which means it's a witch-compatible, more than 99% of internal Android devices use AAudio on API 27 and above, but retreats back to OpenSL ES on the old API, this simple API is something android is missing for a long time, and it becomes the new traditional audio standard on Android.Oboe. The transition to Oboe should be straightforward, so if you want to try oboe, you can add it to your project manually like this, first clone the Oboe archive and edit your CMakeLists .txt. Add Oboe to your project library based on and provide a path to the folder where you clone the repository, and finally, remember to index the newly added Oboe library to start using... First published on www.ackee.cz October 10, 2018.