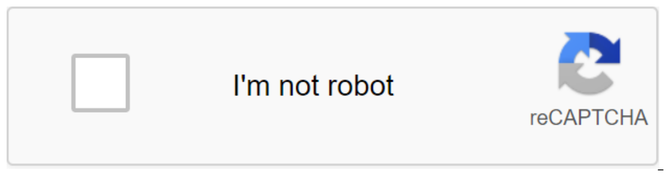
# Android audiomanager stop music




Continue

Two or more Android apps can play audio in the same output stream at the same time. The system mixes everything together. While this is technically impressive, it can be very aggravating for the user. To avoid every music app playing at the same time, Android introduces the idea of audio focus. Only one app can hold the sound focus at the same time. When your app needs to develop audio, it should request audio focus. When he has a focus, he can play sound. However, after purchasing audio focus you won't be able to save it until you're done playing. Another app can request a focus that anticipates the retention of audio focus. If this happens, your app should pause or slow down to make it easier for users to hear a new sound source. Audio focus collaboration. Apps are encouraged to follow audio-focus rules, but the system does not enforce the rules. If the app wants to keep playing loud even after losing its sound focus, nothing can prevent it. This is a bad experience and there is a good chance that users will delete an app that behaves badly this way. A well-behaved audio app should manage the audio focus according to these general guidelines: Call requestAudioFocus () just before playback and check that the call returns AUDIOFOCUS_REQUEST_GRANTED. If you're designing an app as we describe in this guide, the call for theAudioFocus request should be made in onPlay () a callback to your media session. When another app gets an audio focus, stop or pause playback, or duck the volume down. When playback stops, give up the sound focus. The audio focus is handled differently depending on the Android version that works: Starting with Android 2.2 (API Level 8), the apps control audiofocus, causing requestAudioFocus () and abandonAudioFocus (). Apps must also register AudioManager.OnAudioFocusChangeListener with both calls in order to receive callbacks and manage their own sound level. For apps that target Android 5.0 (API level 21) and later, audio apps should use AudioAttributes to describe the type of sound your app is playing. For example, apps that play a speech should indicate CONTENT_TYPE_SPEECH. Apps running Android 8.0 (API level 26) or more should use the requestAudioFocus method, which takes the AudioFocusRequest option. AudioFocusRequest provides information about the audio context and capabilities of your app. The system uses this information to automatically manage profits and loss of sound focus. Audio Focus in Android 8.0 as well starting with Android 8.0 (API level 26) when you call requestAudioFocus () you have to provide audioFocusRequest. To release the audio focus, call the abandonAudioFocusRequest method, which also takes AudioFocusRequest as an argument. The same audioFocusRequest should be used when requesting and refusing to focus. Use AudioFocusRequest to create Because the focus request should always indicate the type of query, the type is included in the builder's constructor. Use builder's methods to customize other query fields. FocusGain field required; all other fields are optional. MethodNotes setFocusGain () This field is required in every request. It takes the same values as the duration of the itit used in the pre-Android 8.0 call for queryAudioFocus (): AUDIOFOCUS_GAIN, AUDIOFOCUS_GAIN_TRANSIENT, AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK, or AUDIOFOCUS_GAIN_TRANSIENT_EXCLUSIVE. setAudioAttributes () AudioAttributes describes the case of use for your app. The system looks at them when the app receives and loses the audio focus. Attributes reinforce the concept of flow type. In Android 8.0 (API level 26) and later thread types for any operation other than volume controls, they get away with it. Use the same attributes in the focus query that you use in the audio player (as shown in the example in this table). First, use AudioAttributes.Builder to identify attributes, and then use this method to assign attributes to the query. If not specified, AudioAttributes AudioAttributes.USAGE_MEDIA by default. setWillPauseWhenDucked () When other app requests focus with AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK, an app that has a focus usually doesn't get onAudioFocusChange () callback because the system can do diving on its own. When you need to pause playback rather than duck volume, call setWillPauseWhenDucked (admittedly) and create and install OnAudioFocusChangeListener as described in automatic diving. setAcceptsDelayedFocusGain () The audio focus request may fail when the focus is blocked by another app. This method allows you to delay the increase in focus: the ability to asynchronically acquire focus when it becomes available. Please note that the delay in getting a focus only works if you have also indicated AudioManager.OnAudioFocusChangeListener in the audio query, as your app needs to get a callback to know that the focus has been provided. setOnAudioFocusChangeListener () OnAudioFocusChangeListener is required only if you also specify willPauseWhenDucked (true) or setAcceptsDelayedFocusGain (admittedly) in the request. There are two ways to set up the listener: one with and one without the handler's argument. A handler is the stream on which the listener works. If you don't specify the handler, use a handler associated with the main Looper. The following example shows how to use AudioFocusRequest.Builder to create AudioFocusRequest and request and opt out of audiofocus: audioManager and getSystemService (Context.AUDIO_SERVICE) as AudioManager focusRequest and AudioFocusRequest.Builder - installAudioAttributes (AudioAttributes.Builder ().start-up - setUsage (AudioAttributes.USAGE_GAME) setContentType (AudioAttributes.CONTENT_TYPE_MUSIC) assembly () ) setAcceptsDelayedFocusGain (admittedly) (admittedly) обработчик) сборка () - mediaPlayer - MediaPlayer () val focusLock - Any () var playbackDelayed - ложное var playbackNowAuthorized -/ val res - audioManager.requestAudioFocus (focusRequest) синхронизированный (focusLock) - воспроизведениеТеперь авторизовано - когда (res) - AudioManager.AUDIOFOCUS_REQUEST_FAILED -> false AudioManager.AUDIOFOCUS_REQUEST_GRANTED - > - воспроизведениеТеперь () правда - AudioManager.AUDIOFOCUS_REQUEST_DELAYED -> - воспроизведениеДейсемый ложный - еще -> ложный . . . . . переопределить удовольствие onAudioFocusChange (фокусИзменить: Int) - когда (фокус-изменение) - AudioManager.AUDIOFOCUS_GAIN -> если (воспроизведениеДеленое резюмеOnFocusGain) - синхронизировано (focusLock) - воспроизведениеДеленое - ложное резюмеOnFocusGain - ложное воспроизведениеТеперь () - AudioManager.AUDIOFOCUS_LOSS -AudioManager.AUDIOFOCUS_LOSS > - синхронизированный (focusLock) - резюмеOnFocusGain - ложное воспроизведениеОтделеное - ложное - паузаПлейбэк () - AudioManager.AUDIOFOCUS_LOSS_TRANSIENT - > - синхронизирован > AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK ный (focusLock) приостановка или ныряние зависит от вашего приложения - audioManager (AudioManager) Context.getSystemService (Context.AUDIO_SERVICE); воспроизведениеАтрибуны - новые AudioAttributes.Builder () .setUsage (AudioAttributes.USAGE_GAME) .setContentType (AudioAttributes.CONTENT_TYPE_MUSIC) .build (); focusRequest - новый AudioFocusRequest.Builder (AudioManager.AUDIOFOCUS_GAIN) .setAudioAttributes (воспроизведениеАтрибутивы) .setAcceptsDelayedFocusGain (правда) .setOnAudioFocusChangeListener (afChangeListener, обработчик) .build(); mediaPlayer - новый MediaPlayer (); окончательный объект focusLock - новый объект (); boolean воспроизведенияДеленые и ложные; boolean воспроизведенияТеперь уполномоченных и ложных; // ... int res - audioManager.requestAudioFocus (фокусРекест); синхронизированный (focusLock) - если (res - AudioManager.AUDIOFOCUS_REQUEST_FAILED) - воспроизведениеТеперь Авторизованная - ложная; - еще, если (res - AudioManager.AUDIOFOCUS_REQUEST_GRANTED) - воспроизведениеТеперь Авторизованная - правда; воспроизведениеТеперь ()AudioManager.AUDIOFOCUS_REQUEST_DELAYED; воспроизведениеТеперь Авторизованная - ложная; No // @Override публичная пустота наAudioFocusChange (int focusChange) - переключатель (фокусИзменяется) - случай AudioManager.AUDIOFOCUS_GAIN: если (воспроизведениеДелайненое резюмеOnFocusGain) случай AudioManager.AUDIOFOCUS_LOSS: синхронизированный (focusLock) - False; PlayDay is false; - PausePlayback (); A break; case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT: synchronized (focusLock) - summaryOnFocusGain - true; ReproductionDeled - false; - PausePlayback (); A break; Case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK: / Suspension or diving depends on the app break; Automatic diving in Android 8.0 (API level 26) when other app requests focus with AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK system can duck and and and volume without calling the application onAudioFocusChange () callback. While automatic diving is acceptable behavior for music and video playback apps, it's not helpful when playing conversational content, such as in an audiobook app. In this case, the application should pause. If you want your app to stop when asked to duck rather than reduce its volume, create OnAudioFocusChangeListener using onAudioFocusChange,, which implements the desired pause/resume behavior. Call setOnAudioFocusChangeListener to register a listener, and call setWillPauseWhenDucked (admittedly) to tell the system to use a callback rather than perform automatic diving. Delaying focus is sometimes unable to provide an audio focus request because the focus is blocked by another app, for example during a phone call. In this case, the AtdioFocus request is returned AUDIOFOCUS_REQUEST_FAILED. When this happens, your app should not start playing the sound because it wasn't focused. A method, setAcceptsDelayedFocusGain (admittedly), which allows your application to process a request for a trick asynchronous. With this set of flags, the request made when the focus is blocked is returned AUDIOFOCUS_REQUEST_DELAYED. When a condition that has blocked the sound focus no longer exists, such as when a phone call ends, the system provides a pending focus request and calls onAudioFocusChange to notify your app. In order to cope with the delay in focusing gain, you must create OnAudioFocusChangeListener with onAudioFocusChange () a callback method that implements the desired behavior and register the listener by calling setOnAudioFocusChangeListener. Audio Focus pre-Android 8.0 When you call requestAudioFocus () you have to specify a hint of duration that can be performed by another app that currently keeps focus and playback: Request constant audio focus (AUDIOFOCUS_GAIN) when you plan to play audio in the foreseeable future (for example, when playing music) and you expect the previous owner of the audio focus to stop playing. Request a transitional focus (AUDIOFOCUS_GAIN_TRANSIENT) when you expect to play audio for a short time and you expect the previous owner to pause playback. Request a transitional focus with diving (AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK) to indicate that you expect to play audio only for a short time, and that it is normal for the previous owner to focus to keep playing if it ducks (reduces) its audio output. Both audio outputs are mixed into the audio stream. The duckling is especially suitable for applications that Audio stream intermittently, for example, for sound driving directions. The requestAudioFocus method also requires AudioManager.OnAudioFocusChangeListener. This listener should be created in the same activity or service that owns your media session. It implements the callback onAudioFocusChange that your app receives when any other app purchases or opts out of audio focus. Teh Teh The segment asks for constant audio to focus on the flow STREAM_MUSIC registers OnAudioFocusChangeListener to process subsequent changes in audio focus. (The change of listener is discussed in response to a change in sound focus.) audioManager - getSystemService (Context.AUDIO_SERVICE) as AudioManager lateinit var playbackAttributes - AudioAttributes.Builder () .setUsage (AudioAttributes.USAGE_MEDIA) .setContentType (AudioAttributes.CONTENT_TYPE_SPEECH) .build () var playbackDelayed - AudioManager.AUDIOFOCUS_REQUEST_GRANTED) / Playback start - AudioManager audio manager (AudioManager) context.getSystemService (Context.AUDIO_SERVICE); AudioManager.OnAudioFocusChangeListener afChangeListener; Request audio focus to play int result - audioManager.requestAudioFocus (afChangeListener, / Use music stream. AudioManager.STREAM_MUSIC, / Requesting constant attention. AudioManager.AUDIOFOCUS_GAIN); If (result - AudioManager.AUDIOFOCUS_REQUEST_GRANTED) / Start playing - When you finish playing, call abandonAudioFocus. audioManager.abandonAudioFocus (afChangeListener) / Refuse audio focus when playing full audioManager.abandonAudioFocus (afChangeListener); This notifies the system that you no longer need to focus and does not register the Associated OnAudioFocusChangeListener. If you request a transitional focus, it will notify the app that has stopped or dived that it can continue to play or restore its volume. Responding to a change in sound focus When an app acquires an audio focus, it should be able to release it when another app requests an audio focus for itself. When this happens, your app receives a call on onAudioFocusChange () method in AudioFocusChangeListener, which you pointed out when the app is called requestAudioFocus. The focusChange option transmitted by onAudioFocusChange indicates what changes are taking place. This corresponds to the length hint used by the app, which is intelligently focused. Your app needs to respond accordingly. Transitional Focus Loss If the focus change is temporary (AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK or AUDIOFOCUS_LOSS_TRANSIENT), your app should dive (unless you rely on automatic diving) or pause playback, but otherwise maintain the same state. During the temporary loss of sound focus, you should continue to monitor changes in audio focus and be ready to resume normal playback when you regain focus. When the blocking app from focus, you get a callback (AUDIOFOCUS_GAIN). At this point, you can restore the volume to a normal level or restart playback. Constant Focus Loss If the loss of sound focus is constant (AUDIOFOCUS_LOSS), another app plays audio. Your app should immediately pause playback because it will never receive AUDIOFOCUS_GAIN callback. To restart playback, user take explicit action, such as clicking on the playback traffic control in the notification or user interface of the app. The next piece of code demonstrates how to implement OnAudioFocusChangeListener and its onAudioFocusChange () callback. Note using the handler to delay the stop call when the sound focus is constantly lost. Val Private Handler - Handler () Private val afChangeListener - AudioManager.OnAudioFocusChangeListener - focusChange -'gt; when (focus-change) - AudioManager.AUDIOFOCUS_LOSS - / Permanent loss of sound focus / Immediate playback of pause mediaController.transportControls.pause() PostDelayed (delay). TimeUnit.SECONDS.toMillis (30)) - AudioManager.AUDIOFOCUS_LOSS_TRANSIENT - / Replay pause - AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK - / / Lower volume, continue to play - AudioManager.AUDIOFOCUS_GAIN - / Your app has been provided with audio focus again / Raise the volume to normal, restart playback if necessary . AudioManager.OnAudioFocusChangeListener afChangeListener - new AudioManager.OnAudioFocusChangeListener () - public void onAudioFocusChange (int focusChange) - if (focus changes to AudioManager.AUDIOFOCUS_LOSS) -/ Permanent loss of audiofocus // Pause of playback immediately AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK AudioManager.AUDIOFOCUS_LOSS_TRANSIENT before you stop playing handler.postDelayed (delayStopRunnable, TimeUnit.SECONDS.toMillis (30)) AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK AudioManager.AUDIOFOCUS_LOSS_TRANSIENT; Keep playing if (focus Change AudioManager.AUDIOFOCUS_GAIN) / Your app has again been given a sound focus / Raise the volume to normal, restart playback if necessary. The handler uses Runnable, which looks like this: private var delayedStopRunnable @Override - Runnable - mediaController.transportControls.stop () - private Runnable delayedStopRunnable - new Runnable (); To ensure that the stop delay doesn't turn on if the user restarts playback, call mHandler.removeCallbacks (mDelayedStopRunnable) in response to any status changes. For example, a call removes callbacks in onPlay of your Callback(), onSkipToNext, etc. Service.

87218875572.pdf