Android Developers Blog

The latest Android and Google Play news for app and game developers.

Respecting Audio Focus

28 August 2013

Posted by Kristan Uccello, Google Developer Relations

It's rude to talk during a presentation, it disrespects the speaker and annoys the audience. If your application doesn't respect the rules of audio focus then it's disrespecting other applications and annoying the user. If you have never heard of audio focus you should take a look at the Android developer training material.

With multiple apps potentially playing audio it's important to think about how they should interact. To avoid every music app playing at the same time, Android uses audio focus to moderate audio playback—your app should only play audio when it holds audio focus. This post provides some tips on how to handle changes in audio focus properly, to ensure the best possible experience for the user.

Requesting audio focus

Audio focus should not be requested when your application starts (don't get greedy), instead delay requesting it until your application is about to do something with an audio stream. By requesting audio focus through the AudioManager system service, an application can use one of the

Listing 1. Requesting audio focus.

```
1. AudioManager am = (AudioManager) mContext.getSystemService(Context
2.
3.  int result = am.requestAudioFocus(mOnAudioFocusChangeListener,
4.    // Hint: the music stream.
5.    AudioManager.STREAM_MUSIC,
6.    // Request permanent focus.
7.    AudioManager.AUDIOFOCUS_GAIN);
8.  if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
9.    mState.audioFocusGranted = true;
10. } else if (result == AudioManager.AUDIOFOCUS_REQUEST_FAILED) {
11.   mState.audioFocusGranted = false;
12. }
```

In line 7 above, you can see that we have requested permanent audio focus. An application could instead request transient focus using `AUDIOFOCUS_GAIN_TRANSIENT` which is appropriate when using the audio system for less than 45 seconds.

Alternatively, the app could use `AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK`, which is appropriate when the use of the audio system may be shared with another application that is currently playing audio (e.g. for playing a "keep it up" prompt in a fitness application and expecting background music to duck during the prompt). The app requesting `AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK` should not use the audio system for more than 15 seconds before releasing focus.

Handling audio focus changes

In order to handle audio focus change events, an application should create an instance of `OnAudioFocusChangeListener`. In the listener, the application will need to handle the`AUDIOFOCUS_GAIN*` event and `AUDIOFOCUS_LOSS*` events (see Table 1). It should be noted that `AUDIOFOCUS_GAIN` has some nuances which are highlighted in Listing 2, below.

Listing 2. Handling audio focus changes.

```
4. public void onAudioFocusChange(int focusChange) {
5.   switch (focusChange) {
6.   case AudioManager.AUDIOFOCUS_GAIN:
7.     mState.audioFocusGranted = true;
8.
9.     if(mState.released) {
10.   initializeMediaPlayer();
11.    }
12.

13.    switch(mState.lastKnownAudioFocusState) {
14.    case UNKNOWN:
15.      if(mState.state == PlayState.PLAY && !mPlayer.isPlaying()) {
16.        mPlayer.start();
17.      }
18.      break;
19.    case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
20.      if(mState.wasPlayingWhenTransientLoss) {
21.        mPlayer.start();
22.      }
23.      break;
24.    case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
25.      restoreVolume();
26.      break;
27.    }
28.

29.    break;
30.  case AudioManager.AUDIOFOCUS_LOSS:
31.    mState.userInitiatedState = false;
32.    mState.audioFocusGranted = false;
33.    teardown();
34.    break;
35.  case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
36.    mState.userInitiatedState = false;
37.    mState.audioFocusGranted = false;
38.    mState.wasPlayingWhenTransientLoss = mPlayer.isPlaying();
39.    mPlayer.pause();
40.    break;
41.  case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
42.   mState.userInitiatedState = false;
43.   mState.audioFocusGranted = false;
44.   lowerVolume();
45.   break;
```

```
49.});
```

`AUDIOFOCUS_GAIN` is used in two distinct scopes of an applications code. First, it can be used when registering for audio focus as shown in Listing 1. This does NOT translate to an event for the registered `OnAudioFocusChangeListener`, meaning that on a successful audio focus request the listener will NOT receive an `AUDIOFOCUS_GAIN` event for the registration.

`AUDIOFOCUS_GAIN` is also used in the implementation of an `OnAudioFocusChangeListener` as an event condition. As stated above, the `AUDIOFOCUS_GAIN` event will not be triggered on audio focus requests. Instead the `AUDIOFOCUS_GAIN` event will occur only after an `AUDIOFOCUS_LOSS*` event has occurred. This is the only constant in the set shown Table 1 that is used in both scopes.

There are four cases that need to be handled by the focus change listener. When the application receives an `AUDIOFOCUS_LOSS` this usually means it will not be getting its focus back. In this case the app should release assets associated with the audio system and stop playback. As an example, imagine a user is playing music using an app and then launches a game which takes audio focus away from the music app. There is no predictable time for when the user will exit the game. More likely, the user will navigate to the home launcher (leaving the game in the background) and launch yet another application or return to the music app causing a resume which would then request audio focus again.

However another case exists that warrants some discussion. There is a difference between losing audio focus permanently (as described above) and temporarily. When an application receives an `AUDIOFOCUS_LOSS_TRANSIENT`, the behavior of the app should be that it suspends its use of the audio system until it receives an `AUDIOFOCUS_GAIN` event. When the `AUDIOFOCUS_LOSS_TRANSIENT` occurs, the application should make a note that the loss is temporary, that way on audio focus gain it can reason about what the correct behavior should be (see lines 13-27 of Listing 2).

Sometimes an app loses audio focus (receives an `AUDIOFOCUS_LOSS`) and the

see if it is receiving the gain after a temporary loss and can thus resume use of the audio system or if recovering from an permanent loss, setup for playback.

If an application will only be using the audio capabilities for a short time (less than 45 seconds), it should use an `AUDIOFOCUS_GAIN_TRANSIENT` focus request and abandon focus after it has completed its playback or capture. Audio focus is handled as a stack on the system — as such the last process to request audio focus wins.

When audio focus has been gained this is the appropriate time to create a `MediaPlayer` or `MediaRecorder` instance and allocate resources. Likewise when an app receives `AUDIOFOCUS_LOSS` it is good practice to clean up any resources allocated. Gaining audio focus has three possibilities that also correspond to the three audio focus loss cases in Table 1. It is a good practice to always explicitly handle all the loss cases in the `OnAudioFocusChangeListener`.

Table 1. Audio focus gain and loss implication.

GAIN	LOSS
`AUDIOFOCUS_GAIN`	`AUDIOFOCUS_LOSS`
`AUDIOFOCUS_GAIN_TRANSIENT`	`AUDIOFOCUS_LOSS_TRANSIENT`
`AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK`	`AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK`

> Note: `AUDIOFOCUS_GAIN` is used in two places. When requesting audio focus it is passed in as a hint to the `AudioManager` and it is used as an event case in the `OnAudioFocusChangeListener`. The gain events highlighted in green are only used when requesting audio focus. The loss events are only used in the `OnAudioFocusChangeListener`.

Table 2. Audio stream types.

Stream Type	Description
`STREAM_ALARM`	The audio stream for alarms

	videos) playback
`STREAM_NOTIFICATION`	The audio stream for notifications
STREAM_RING	The audio stream for the phone ring
STREAM_SYSTEM	The audio stream for system sounds

An app will request audio focus (see an example in the sample source code linked below) from the `AudioManager` (Listing 1, line 1). The three arguments it provides are an audio focus change listener object (optional), a hint as to what audio channel to use (Table 2, most apps should use `STREAM_MUSIC`) and the type of audio focus from Table 1, column 1. If audio focus is granted by the system (`AUDIOFOCUS_REQUEST_GRANTED`), only then handle any initialization (see Listing 1, line 9).

> Note: The system will not grant audio focus (`AUDIOFOCUS_REQUEST_FAILED`) if there is a phone call currently in process and the application will not receive `AUDIOFOCUS_GAIN` after the call ends.

Within an implementation of `OnAudioFocusChange()`, understanding what to do when an application receives an `onAudioFocusChange()` event is summarized in Table 3.

In the cases of losing audio focus be sure to check that the loss is in fact final. If the app receives an `AUDIOFOCUS_LOSS_TRANSIENT` or `AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK` it can hold onto the media resources it has created (don't call `release()`) as there will likely be another audio focus change event very soon thereafter. The app should take note that it has received a transient loss using some sort of state flag or simple state machine.

If an application were to request permanent audio focus with `AUDIOFOCUS_GAIN` and then receive an `AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK` an appropriate action for the application would be to lower its stream volume (make sure to store the original volume state somewhere) and then raise the volume upon receiving an `AUDIOFOCUS_GAIN` event (see Figure 1, below).

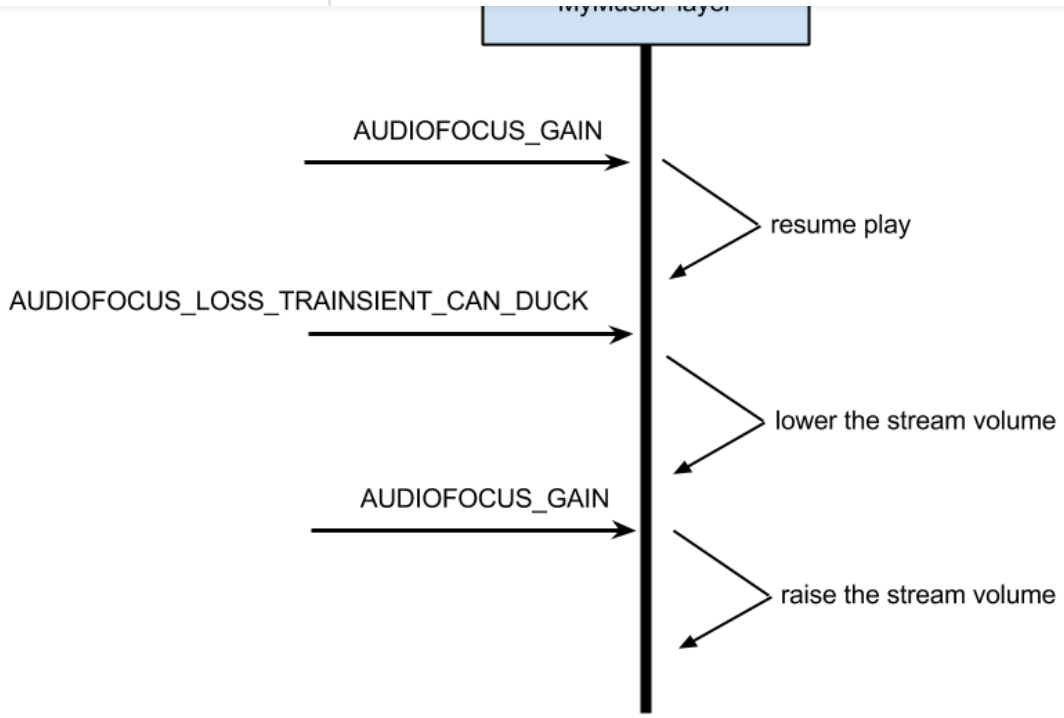

Table 3. Appropriate actions by focus change type.

Focus Change Type	Appropriate Action
`AUDIOFOCUS_GAIN`	**Gain event after loss event**: Resume playback of media unless other state flags set by the application indicate otherwise. For example, the user paused the media prior to loss event.
`AUDIOFOCUS_LOSS`	Stop playback. Release assets.
`AUDIOFOCUS_LOSS_TRANSIENT`	Pause playback and keep a state flag that the loss is transient so that when the `AUDIOFOCUS_GAIN` event occurs you can resume playback if appropriate. Do not

keeping track of state as with `AUDIOFOCUS_LOSS_TRANSIENT`. Do not release assets.

Conclusion and further reading

Understanding how to be a good audio citizen application on an Android device means respecting the system's audio focus rules and handling each case appropriately. Try to make your application behave in a consistent manner and not negatively surprise the user. There is a lot more that can be talked about within the audio system on Android and in the material below you will find some additional discussions.

- Managing Audio Focus, an Android developer training class.
- Allowing applications to play nice(r) with each other: Handling remote control buttons, a post on the Android Developers Blog.

Example source code is available here:

https://android.googlesource.com/platform/development/+/master/samples/RandomMusicPlayer

Labels: App quality , archive , Audio , Games , Media and Camera

Google Developers Blog

Programa con Google (Spanish LATAM)

Codigo (Portuguese LATAM)

Developers Italia

Google Developers China Blog

Google Developers Indonesia Blog

Google Developers Korea

Google Developers Japan