About This Article

Many applications in Android use audio, and there may be several media services competing for use of the one available audio output. This document provides a description on how to handle audio focus (the AudioHandling.java class example can be found at the end).

Scope:

This article explains how to handle audio sound in android applications, to prevent it from overlapping with VOIP calls. It assumes that the user has already installed Android and the necessary tools to develop an application. It also assumes that the user is familiar with a basic knowledge of Android.

Introduction

As there is only one audio output and there may be several media services competing for its use, Android Applications should handle audio focus, to avoid a negative audio experience. For example, when a user is listening to music and another application needs to notify the user of something very important, the user may not hear the notification tone due to the volume of the music.
Before Android 2.2, there was no built-in mechanism to address this issue, which could in some cases lead to a bad user experience.
Android 2.2 offers a way for applications to negotiate their use of the device's audio output. This mechanism is called - Audio Focus.

Handling Audio Focus

An application should request Audio Focus when it needs to utilize the audio output, either with music or notification sounds. Once it has the priority of Audio Focus, it can use the sound output freely, while listening for focus changes. If Audio Focus loses the focus, it should immediately either kill the audio or lower it to a quiet level and only resume loud playback when it receives focus again. If an application needs to output music, Audio Focus should be requested. The method requestAudioFocus() should be called from the AudioManager.

```
AudioManager audioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);

int result = audioManager.requestAudioFocus( this, AudioManager.STREAM_MUSIC,
                                                 AudioManager.AUDIOFOCUS_GAIN);

if (result != AudioManager.AUDIOFOCUS_REQUEST_GRANTED)
{
        // could not get audio focus.
}
```

The first parameter to requestAudioFocus() is an AudioManager.OnAudioFocusChangeListener, whose onAudioFocusChange() method is called whenever there is a change in audio focus. Therefore, you should also implement this interface on your service and activities.

```
class  OnAudioFocus  extends  Service
       implements   AudioManager.OnAudioFocusChangeListener
     {
             // ....
```

```
             public void onAudioFocusChange(int focusChange)
            {
                      // Do something based on focus change...
             }
     }
```

The parameter focusChange gives information about the changes in the audio focus.

Given below are the values obtained based on audio focus

- •AUDIOFOCUS_GAIN

  When audio focus is gained
  •AUDIOFOCUS_LOSS

  You have lost the audio focus for a presumably long time. You must stop all audio playback and because you may not have focus back for some time, this is a great opportunity to clean up your resources as much as possible. For example, releasing you should release the MediaPlayer.
  •AUDIOFOCUS_LOSS_TRANSIENT

  You have temporarily lost audio focus, but should receive it back shortly. You must stop all audio playback, but you can keep your resources because you will probably get focus back shortly
  •AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK

  You have temporarily lost audio focus, but you are allowed to continue to play audio quietly (at a low volume) instead of killing audio completely.

The implementation of these values is shown below

```
public void onAudioFocusChange (int focusChange)
{
         switch (focusChange)
        {
                case AudioManager.AUDIOFOCUS_GAIN:
                                        // TODO : resume playback
                case AudioManager.AUDIOFOCUS_LOSS:
                                        // TODO : stop playback and release media
player
                case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
                                        // TODO : pause palyback
                case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
                                        // TODO : continue playing at an attenuated
level
         }
```

If an application doesn’t need to output audio anymore, audio focus should be abandoned.

```
int result = audioManager.abandonAudioFocus(this);
```

These audio focus APIs are available only with API Level 8 (Android 2.2) and above. Hence backward compatibility can be achieved by calling the audio focus methods by reflection or by implementing all the audio focus features in a separate class.

You can create an instance of this class only if you detect that the system is running API level 8 or above. For example

```
if (android.os.Build.VERSION.SDK_INT >= 8)
{
   //TODO: separate class implementing all audio focus features
}
```

Sample Example

Given below is the sample example on Handling Audio Focus

Class: AudioHandling.java

```
package com.audio.handling;

import android.app.Activity;
import android.content.Context;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class AudioHandling extends Activity implements OnClickListener {

         public Button playm = null;
         public Button pausem = null;
         public Button stopm = null;

         public MediaPlayer mp = null;
         public AudioManager audioManager = null;
         public boolean playing;

         /** Called when the activity is first created. */
         @Override
         public void onCreate(Bundle savedInstanceState) {
                  super.onCreate(savedInstanceState);
                  setContentView(R.layout.main);
                  audioManager
                                    = (AudioManager)
getSystemService(Context.AUDIO_SERVICE);
                  // PLAYER BUTTONS
                  playm = (Button) findViewById(R.id.Button01);
                  playm.setOnClickListener(this);
```

```
        pausem = (Button) findViewById(R.id.Button02);
        pausem.setOnClickListener(this);

        stopm = (Button) findViewById(R.id.Button03);
        stopm.setOnClickListener(this);
}

@Override
public void onClick(View v) {
        // PLAYER
        if (v == playm) {
                OnAudioFocus onAudioFocus=new OnAudioFocus(this);
                audioManager.requestAudioFocus(onAudioFocus,
        AudioManager.STREAM_MUSIC, AudioManager.AUDIOFOCUS_GAIN);
                playSong();
        }
        if (v == pausem) {
                pauseSong();

        }
        if (v == stopm) {
                stopSong();

        }

}

public void playSong() {
        if (mp == null) {
                mp = MediaPlayer.create(this, R.raw.mysong);
                try {
                        mp.start();
                        playing = true;
                } catch (IllegalArgumentException e) {
                        e.printStackTrace();
                } catch (IllegalStateException e) {
                        e.printStackTrace();
                }

        }
}
```

```
        public void stopSong() {
                if (mp != null) {
                        mp.stop();
                        mp.release();
                        playing = false;
                        mp = null;
                }
        }

        public void pauseSong() {
                if (mp != null) {
                        mp.pause();

                }
        }

}

class OnAudioFocus implements
                AudioManager.OnAudioFocusChangeListener {

        AudioHandling onAudioFocus;

        public OnAudioFocus(AudioHandling onAudioFocus) {
                this.onAudioFocus = onAudioFocus;
        }



        public void onAudioFocusChange(int focusChange) {
                switch (focusChange) {
                case AudioManager.AUDIOFOCUS_GAIN:
                        if (onAudioFocus.mp == null) {
                                onAudioFocus.playSong();
                        } else if (onAudioFocus.mp.isPlaying()) {
                                onAudioFocus.mp.start();
                        }

                        break;

                case AudioManager.AUDIOFOCUS_LOSS:
                        if (onAudioFocus.mp.isPlaying())
                                onAudioFocus.stopSong();
```

```
                        break;
                case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT:
                        if (onAudioFocus.mp.isPlaying())
                                onAudioFocus.pauseSong();

                        break;
                case AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK:
                        if (onAudioFocus.mp.isPlaying())
                                onAudioFocus.mp.setVolume(0.1f, 0.1f);
                        break;

                }
        }
}
```

ref:http://developer.samsung.com/android/technical-docs/Handling-Audio-Sound-from-Overlapping-with-VOIP-call