Multimedia and Camera

The Android multimedia framework includes support for capturing and playing audio, video and images in a variety of common media types, so that you can easily integrate them into your applications. You can play audio or video from media files stored in your application's resources, from standalone files in the file system, or from a data stream arriving over a network connection, all using the `MediaPlayer` or `JetPlayer` APIs. You can also record audio, video and take pictures using the `MediaRecorder`and `Camera` APIs if supported by the device hardware.

The following topics show you how to use the Android framework to implement multimedia capture and playback.

Media Playback

How to play audio and video in your application.

JetPlayer

How to play interactive audio and video in your application using content created with JetCreator.

Camera

How to use a device camera to take pictures or video in your application.

Audio Capture

How to record sound in your application.

Managing Audio Focus

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—only apps that hold the audio focus should play audio.

Before your app starts playing audio it should request—and receive—the audio focus. Likewise, it should know how to listen for a loss of audio focus and respond appropriately when that happens.

Request the Audio Focus

Before your app starts playing any audio, it should hold the audio focus for the stream it will be using. This is done with a call to `requestAudioFocus()` which returns `AUDIOFOCUS_REQUEST_GRANTED` if your request is successful.

You must specify which stream you're using and whether you expect to require transient or permanent audio focus. Request transient focus when you expect to play audio for only a short time (for example when playing navigation instructions). Request permanent audio focus when you plan to play audio for the foreseeable future (for example, when playing music).

The following snippet requests permanent audio focus on the music audio stream. You should request the audio focus immediately before you begin playback, such as when the user presses play or the background music for the next game level begins.

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE);
...

// Request audio focus for playback
int result = am.requestAudioFocus(afChangeListener,
                                  // Use the music stream.
                                  AudioManager.STREAM_MUSIC,
                                  // Request permanent focus.
                                  AudioManager.AUDIOFOCUS_GAIN);

if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
    // Start playback.
}
```

Once you've finished playback be sure to call `abandonAudioFocus()`. This notifies the system that you no longer require focus and unregisters the associated`AudioManager.OnAudioFocusChangeListener`. In the case of abandoning transient focus, this allows any interupted app to continue playback.

```
// Abandon audio focus when playback complete
am.abandonAudioFocus(afChangeListener);
```

When requesting transient audio focus you have an additional option: whether or not you want to enable "ducking." Normally, when a well-behaved audio app loses audio focus it immediately silences its playback. By requesting a transient audio focus that allows ducking you tell other audio apps that it's acceptable for them to keep playing, provided they lower their volume until the focus returns to them.

```
// Request audio focus for playback
int result = am.requestAudioFocus(afChangeListener,
                                  // Use the music stream.
                                  AudioManager.STREAM MUSIC,
                                  // Request permanent focus.
                                  AudioManager.AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK);

if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    // Start playback.
}
```

Ducking is particularly suitable for apps that use the audio stream intermittently, such as for audible driving directions.

Whenever another app requests audio focus as described above, its choice between permanent and transient (with or without support for ducking) audio focus is received by the listener you registered when requesting focus.

Handle the Loss of Audio Focus

If your app can request audio focus, it follows that it will in turn lose that focus when another app requests it. How your app responds to a loss of audio focus depends on the manner of that loss.

The onAudioFocusChange() callback method of they audio focus change listener you registered when requesting audio focus receives a parameter that describes the focus change event. Specifically, the possible focus loss events mirror the focus request types from the previous section—permanent loss, transient loss, and transient with ducking permitted.

Generally speaking, a transient (temporary) loss of audio focus should result in your app silencing it's audio stream, but otherwise maintaining the same state. You should continue to monitor changes in audio focus and be prepared to resume playback where it was paused once you've regained the focus.

If the audio focus loss is permanent, it's assumed that another application is now being used to listen to audio and your app should effectively end itself. In practical terms, that means stopping playback, removing media button listeners—allowing the new audio player to exclusively handle those events—and abandoning your audio focus. At that point, you would expect a user action (pressing play in your app) to be required before you resume playing audio.

In the following code snippet, we pause the playback or our media player object if the audio loss is transien and resume it when we have regained the focus. If the loss is permanent, it unregisters our media button event receiver and stops monitoring audio focus changes.

```
OnAudioFocusChangeListener afChangeListener = new OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT
            // Pause playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_GAIN) {
            // Resume playback
        } else if (focusChange == AudioManager.AUDIOFOCUS_LOSS) {
            am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
            am.abandonAudioFocus(afChangeListener);
            // Stop playback
        }
    }
};
```

In the case of a transient loss of audio focus where ducking is permitted, rather than pausing playback, you can "duck" instead.

Duck!

Ducking is the process of lowering your audio stream output volume to make transient audio from another app easier to hear without totally disrupting the audio from your own application.

In the following code snippet lowers the volume on our media player object when we temporarily lose focus, then returns it to its previous level when we regain focus.

```
OnAudioFocusChangeListener afChangeListener = new OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AUDIOFOCUS LOSS TRANSIENT CAN DUCK
            // Lower the volume
        } else if (focusChange == AudioManager.AUDIOFOCUS GAIN) {
            // Raise it back to normal
        }
```

```
        }
    };
```

A loss of audio focus is the most important broadcast to react to, but not the only one. The system broadcasts a number of intents to alert you to changes in user's audio experience. The next lesson demonstrates how to monitor them to improve the user's overall experience.