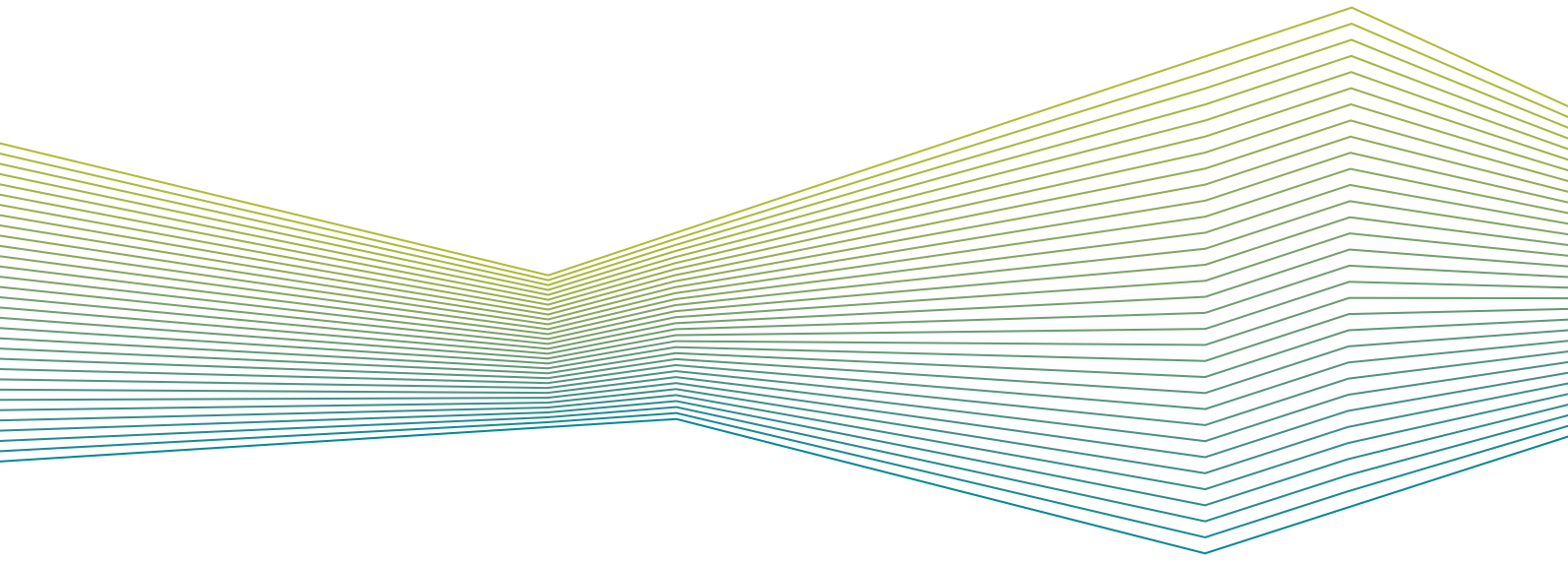
Dolby Digital Plus playback guide for Android application developers

11 October 2016



Copyright

For information, contact:

Dolby Laboratories, Inc.
1275 Market Street
San Francisco, CA 94103-1410 USA
Telephone 415-558-0200
Fax 415-863-1373
http://www.dolby.com

Trademarks

Dolby and the double-D symbol are registered trademarks of Dolby Laboratories. Following are trademarks of Dolby Laboratories:

Dialogue Intelligence™
Dolby®
Dolby AccessLink™
Dolby Advanced Audio™
Dolby Atmos®
Dolby Audio™
Dolby CaptiView™
Dolby CineAsset™
Dolby CineAsset Player™
Dolby Cinema™
Dolby Digital Plus™
Dolby Digital Plus Advanced Audio™
Dolby Digital Plus Home Theater™
Dolby Fidelio™
Dolby Home Theater®
Dolby Theatre®
Dolby Vision™
Dolby Voice®
Feel Every Dimension™
Feel Every Dimension in Dolby™
Feel Every Dimension in Dolby Atmos™
MLP Lossless™
Pro Logic®
Surround EX™

All other trademarks remain the property of their respective owners.

Contents

1 Introduction to the Dolby Digital Plus playback guide for Android application developers

You can use MediaPlayer and MediaCodec classes to add support for Dolby Digital Plus content playback in your applications for Dolby certified Android devices.

1.1 Using this documentation

This documentation provides an overview of two Android media classes that developers can use to add Dolby Digital Plus playback in their applications, and demonstrates the process with code snippets.

For standard playback applications, follow the standard Android audio focus guidelines. For more details, see https://developer.android.com/training/managing-audio/audio-focus.html.

This documentation and the code snippets included are created based on Android 6.0 Marshmallow.

1.2 Overview of Dolby Digital Plus playback on Android

MediaPlayer and MediaCodec application programming interfaces (APIs) are commonly used for Dolby Digital Plus audio playback in Java applications.

The Android multimedia framework provides MediaPlayer, MediaCodec, OpenSL ES, OpenMAX AL, and SoundPool APIs to process audio/video files and streams. MediaPlayer, MediaCodec, and SoundPool classes are for Java applications. OpenSL ES and OpenMAX AL are for C/C++ applications for which audio latency is critical. On Dolby certified Android devices, all of these APIs can be used for the playback of Dolby Digital Plus audio in applications. This documentation focuses on MediaPlayer and MediaCodec APIs.

The MediaPlayer class is provided to access built-in MediaPlayer services. It creates a basic media player that allows you to play, pause, fast forward, and rewind file-based or streaming media.

The MediaCodec class provides low-level access to platform hardware and software codecs (decoders and encoders) on a device. It can query the supported codecs on a device. MediaCodec APIs include methods that get the input media sample and output decoded data. On a device with an integrated Dolby Audio solution, an application can use MediaCodec APIs to access Dolby Digital Plus decoder and play Dolby Digital Plus streams.

The diagrams show the architectures for Android MediaPlayer and MediaCodec APIs.

Figure 1: MediaPlayer architecture

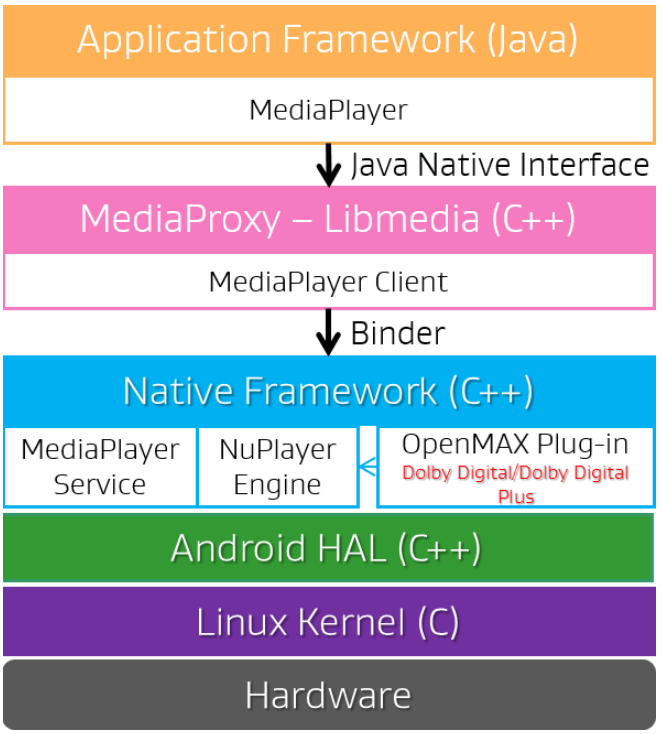
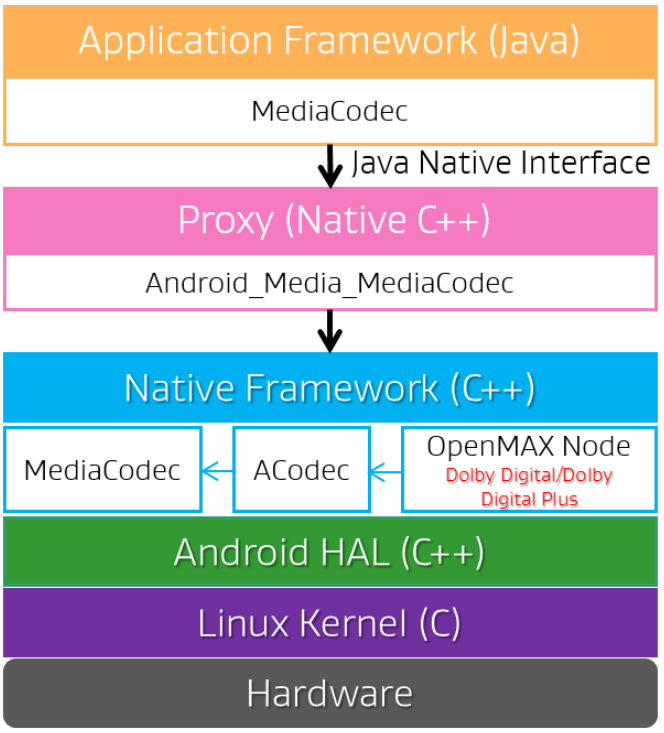


Figure 2: MediaCodec architecture



1.3 Device requirements

The material in this sample project is designed to be used on devices that have been certified by Dolby to use Dolby Audio technologies.

To determine whether the target devices for your application support Dolby Digital Plus, you can use this code snippet, which is a wrapper function based on MediaCodec APIs.

```
private static boolean verifyDecoderAvailable(String mimeType) {
            MediaCodecInfo[] codecInfos = new
            MediaCodecList(MediaCodecList.REGULAR_CODECS).getCodecInfos();
            for (MediaCodecInfo info : codecInfos) {
              if (!info.isEncoder()) {
              String[] typeList = info.getSupportedTypes();
                 for (int j = 0; j < typeList.length; ++j) {
                 if (typeList[j].equalsIgnoreCase(mimeType)) {
                 Log.i(TAG, "Found codec " + info.getName() + " for the type " +
mimeType);
                 return true;
                 }
                 }
                 }
                 }
              return false;
}
```

Android includes Multipurpose Internet Mail Extensions (MIME) types for Dolby Digital Plus and Dolby Digital:

- audio/eac3 for Dolby Digital Plus
- audio/ac3 for Dolby Digital

1.4 Media formats supported by Dolby Audio

Dolby Audio certified Android devices include support for media formats as described in this table.

Table 1: Media format and codec support by Dolby Audio

Type	Format/codec	Encoder	Decoder	Details	Supported file types/container formats
Audio	Dolby Digital Plus (E-AC-3)	No	Yes	Support for mono/stereo/ 5.1/7.1 content with sampling rates of 32, 44.1, and 48 kHz	• MPEG-4 (.mp4, .m4a) • MPEG transport stream (.ts)
	Dolby Digital (AC-3)	No	Yes	Support for mono/stereo/5.1 content with sampling rates of 32, 44.1, and 48 kHz	• MPEG-4 (.mp4,.m4a) • MPEG-TS (.ts)

For details about core media format and codec support on the Android platform, see the supported media formats document at http://developer.android.com/guide/appendix/media-formats.html.

1.5 Contacting Dolby Support

For any questions regarding the design of an application, contact Dolby at developer@dolby.com. By utilizing Dolby expertise, especially during the design process, many problems that might require design revisions before an application is released may be prevented.

2 Adding Dolby Digital Plus playback in your applications

Use MediaPlayer APIs to add basic audio playback, or use MediaCodec APIs to add audio playback with hardware-level encoding and decoding control in your applications.

2.1 Declaring permissions in the project manifest file

The permissions to be declared depend on the use scenarios of your application.

Procedure

- To play media files on an SD card, you need to declare the SD card access permission:

```
<uses--permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

- To stream media files, you need to declare the Internet access permission:

```
<uses--permission android:name="android.permission.INTERNET " />
```

Note: Beginning with Android 6.0 (API level 23), users grant permissions to an application when the application runs, not when the application is installed. To check and request permissions, use the `checkSelfPermission()` and `requestPermissions()` functions. For details, see https://developer.android.com/training/permissions/requesting.html.

2.2 Using Android MediaPlayer APIs

Use MediaPlayer APIs to create a basic media player that supports playback, pause, fast forward, and rewind for file-based and streaming media.

Procedure

1. Initialize a MediaPlayer object.
2. Set the media source.
3. Prepare the media for playback.
4. Start the playback.

   This snippet shows how to use the MediaPlayer APIs:

```
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnBufferingUpdateListener;
import android.media.MediaPlayer.OnCompletionListener;
import android.media.MediaPlayer.OnErrorListener;
import android.media.MediaPlayer.OnPreparedListener;

String url = "URL"; // local path or streaming URL
MediaPlayer mMediaPlayer = new MediaPlayer();
```

```
mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mMediaPlayer.setDataSource(url);
mMediaPlayer.prepare();
mMediaPlayer.start();
```

The prepare() method prepares the media for synchronized playback. While the audio is preparing, the application UI is not functional. Therefore, for a streaming use case, the prepareAsync() method should be used to prepare the MediaPlayer in the background for another thread; it then notifies the onPreparedListners when the preparation completes. The MediaPlayer class also provides completion, error, and buffer status call backs.

This snippet shows how prepareAsync() and onPreparedListners functions are used:

```
public class MediaPlayerController implements
OnCompletionListener, OnPreparedListener, OnErrorListener,
OnBufferingUpdateListener {
……
mMediaPlayer.setOnCompletionListener(this);
mMediaPlayer.setOnPreparedListener(this);
mMediaPlayer.setOnErrorListener(this);
mMediaPlayer.setOnBufferingUpdateListener(this);
……
……
@Override
public boolean onError(MediaPlayer arg0, int arg1,
int arg2) {
this.onError(arg1, arg2);
return false;
}
@Override
public void onPrepared(MediaPlayer mp) {
mPrepared = true;
this.onPrepared();
}
@Override
public void onCompletion(MediaPlayer mp) {
this.onCompletion();
}
……
```

2.3 Using Android MediaCodec APIs

Use MediaCodec APIs to add audio playback with hardware-level encoding and decoding control in your applications.

About this task

The MediaCodec class provides a low-level multimedia support infrastructure. With MediaCodec APIs, you can access hardware-level encoding and decoding functionalities from a Java application, and have greater control over the playback process. To learn more about MediaCodec APIs, see https://developer.android.com/reference/android/media/MediaCodec.html.

Procedure

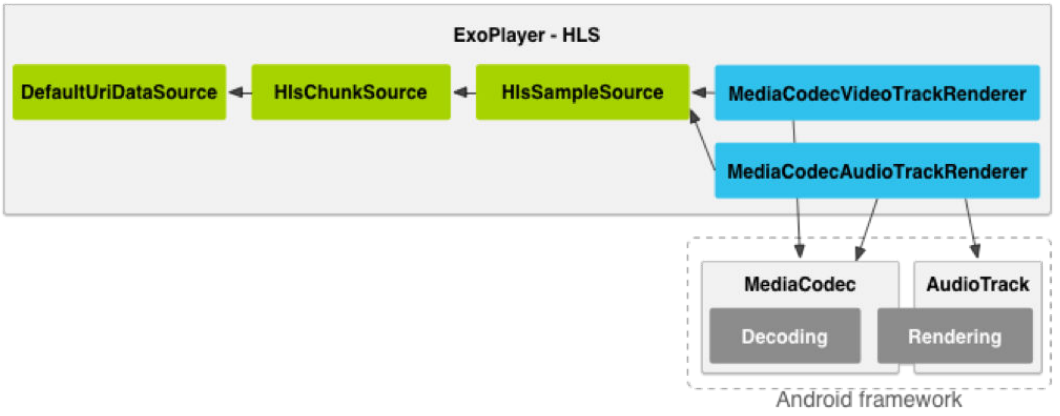
1. Create a MediaExtractor instance, and select the audio stream according to the MIME type of Dolby Digital Plus.

```
public final static String EAC3_AUDIO = "audio/eac3";
mExtractor = new MediaExtractor();

for (int i = 0; i < numTracks; ++i) {
    MediaFormat format = mExtractor.getTrackFormat(i);
    String mime_type = format.getString(MediaFormat.KEY_MIME);
        if (mime_type.equals(EAC3_AUDIO)) { //eac3 stands for Dolby Digital Plus.
        mExtractor.selectTrack(i);
        mFormatAudio = format;
        mMimeTypeAudio = mime_type;
        Log.d(TAG, "selected audio track with type: " + mime_type);
        }
    ......
 }
```

For HTTP Live Streaming (HLS) of Dolby Digital Plus content, refer to the ExoPlayer open source project of Google for implementation details. ExoPlayer asynchronously loads a manifest (also called a playlist) and notifies the listener upon the completion of the loading process. Audio and video samples are decoded with MediaCodec APIs. This illustration shows how HLS playback is handled in ExoPlayer.

Figure 3: HLS handling in ExoPlayer



2. Create and configure a MediaCodec instance.

```
mCodecAudio = MediaCodec.createDecoderByType(mMimeTypeAudio);
mCodecAudio.configure(mFormatAudio, null, null, 0);
```

3. Start the MediaCodec instance to decode the audio stream.

```
mCodecAudio.start();
```

4. Create and start an AudioTrack instance to render the decoded PCM audio data.

```
mTrack = new AudioTrack(AudioManager.STREAM_MUSIC, sampleRateHz,
channelConfig, AudioFormat.ENCODING_PCM_16BIT, bufferSize*2,
AudioTrack.MODE_STREAM, 0);
```

```
mTrack.play();
mTrack.write(pcm_data, 0, pcm_data.length);
```

5. Release the instances of MediaExtractor, MediaCodec, and AudioTrack when the playback stops.

```
if (mCodecAudio != null) {
mCodecAudio.stop();
mCodecAudio.release();
mCodecAudio = null;
}
if (mExtractor != null) {
mExtractor.release();
mExtractor = null;
}
if (null != mTrack) {
mTrack.stop();
mTrack.flush();
mTrack.release();
mTrack = null;
}
```

Glossary

API

Application programming interface. A set of functions that can be used to access the functions of an operating system or other type of software.

HLS

HTTP Live Streaming. An adaptive streaming protocol for delivery of media content developed by Apple. The format uses the MPEG-2 transport streams to contain and deliver the content.

MIME

Multipurpose Internet Mail Extensions. An Internet standard that extends the format of email to support, and that is also used to describe the format of content in communication protocols such as HTTP.