Media app architecture overview

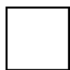
bookmark_border

This section explains how to separate a media player app into a media controller (for the UI) and a media session (for the actual player). It describes two media app architectures: a client/server design that works well for audio apps and a single-activity design for video players. It also shows how to make media apps respond to hardware controls and cooperate with other apps that use the audio output stream.

Player and UI

A multimedia application that plays audio or video usually has two parts:

- A player that takes digital media in and renders it as video and/or audio
- A UI with transport controls to run the player and optionally display the player's state



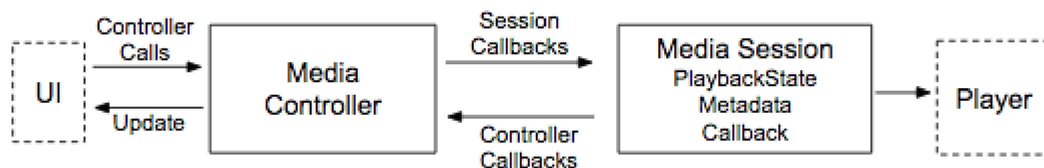
In Android you can build your own player from the ground up, or you can choose from these options:

- The MediaPlayer class provides the basic functionality for a bare-bones player that supports the most common audio/video formats and data sources.
- ExoPlayer is an open source library that's built on top of lower-level media framework components like MediaCodec and AudioTrack. ExoPlayer supports high-performance features like DASH which are not available in MediaPlayer. You can customize the ExoPlayer code, making it easy to add new components. ExoPlayer can only be used with Android version 4.1 and higher.

Media session and media controller

While the APIs for the UI and player can be arbitrary, the nature of the interaction between the two pieces is basically the same for all media player apps. The Android framework defines two classes, a *media session* and a *media controller,* that impose a well-defined structure for building a media player app.

The media session and media controller communicate with each other using predefined callbacks that correspond to standard player actions (play, pause, stop, etc.), as well as extensible custom calls that you use to define special behaviors unique to your app.



Media session

A media session is responsible for all communication with the player. It hides the player's API from the rest of your app. The player is only called from the media session that controls it.

The session maintains a representation of the player's state (playing/paused) and information about what is playing. A session can receive callbacks from one or more media controllers. This makes it possible for your player to be controlled by your app's UI as well as companion devices running Wear OS and Android Auto. The logic that responds to callbacks must be consistent. The response to a MediaSession callback should be the same no matter which client app initiated the callback.

Media controller

A media controller isolates your UI. Your UI code only communicates with the media controller, not the player itself. The media controller translates transport control actions into callbacks to the media session. It also receives callbacks from the media session whenever the session state changes. This provides a mechanism to automatically update the associated UI. A media controller can only connect to one media session at a time.

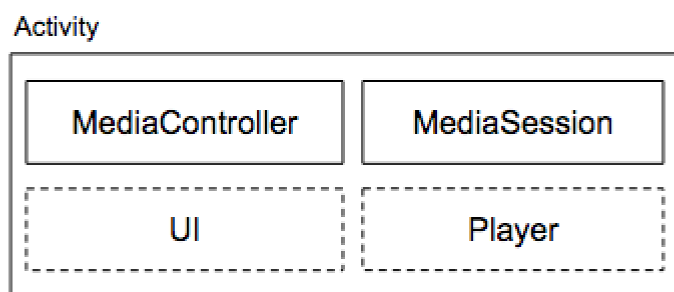
When you use a media controller and a media session, you can deploy different interfaces and/or players at runtime. You can change your app's appearance and/or performance independently depending on the capabilities of the device on which it's running.

Video apps versus audio apps

When playing a video, your eyes and ears are both engaged. When playing audio, you are listening, but you can also work with a different app at the same time. There's a different design for each use case.

Video app

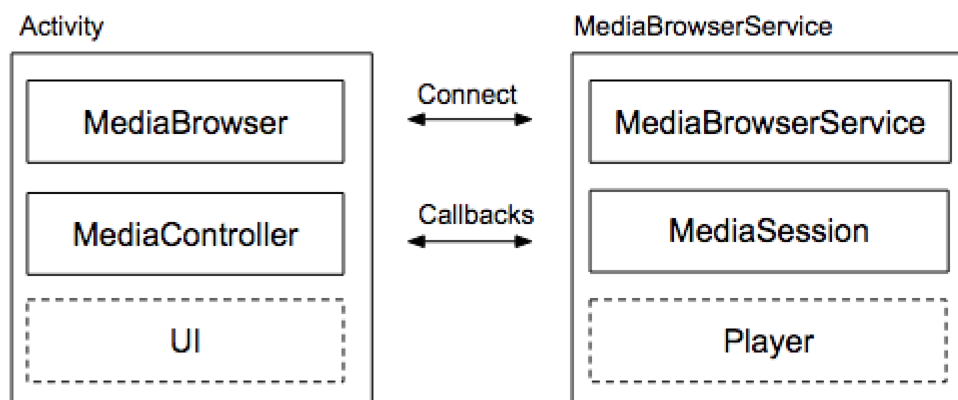
A video app needs a window for viewing content. For this reason a video app is usually implemented as a single Android activity. The screen on which the video appears is part of the activity.



Audio app

An audio player does not always need to have its UI visible. Once it begins to play audio, the player can run as a background task. The user can switch to another app and work while continuing to listen.

To implement this design in Android, you can build an audio app using two components: an activity for the UI and a service for the player. If the user switches to another app, the service can run in the background. By factoring the two parts of an audio app into separate components, each can run more efficiently on its own. A UI is usually short-lived compared to a player, which may run for a long time without a UI.

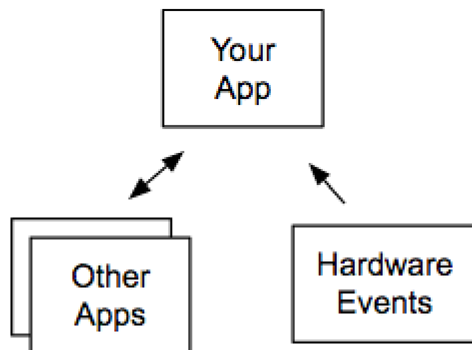


The support library provides two classes to implement this client/server approach: MediaBrowserService and MediaBrowser. The service component is implemented as a subclass of MediaBrowserService containing the media session and its player. The activity with the UI and the media controller should include a MediaBrowser, which communicates with the MediaBrowserService.

Using MediaBrowserService makes it easy for companion devices (like Android Auto and Wear) to discover your app, connect to it, browse for content, and control playback, without accessing your app's UI activity at all. In fact, there can be multiple apps connected to the same MediaBrowserService at the same time, each app with its own MediaController. An app that offers a MediaBrowserService should be able to handle multiple simultaneous connections.

Media apps and the Android audio infrastructure

A well-designed media app should "play well together" with other apps that play audio. It should be prepared to share the phone and cooperate with other apps on your device that use audio. It should also respond to hardware controls on the device.



All of this behavior is described in Controlling Audio Output.

The media-compat library

The media-compat library contains classes that are helpful for building apps that play audio and video. These classes are compatible with devices running Android 2.3 (API level 9) and higher. They also work with other Android features to create a comfortable, familiar Android experience.

The recommended implementation of media sessions and media controllers are the classes MediaSessionCompat and MediaControllerCompat, which are defined in the media-compat support library. They replace earlier versions of the classes MediaSession and MediaController that were introduced in Android 5.0 (API level 21). The compat classes offer the same functionality but make it easier to develop your app because you only need to write to one API. The library takes care of backward compatibility by translating media session methods to the equivalent methods on older platform versions when available.

If you already have a working app that's using the older classes, we recommend updating to the compat classes. When you use the compat versions you can remove all calls to registerMediaButtonReceiver() and any methods from RemoteControlClient.

Measuring performance

In Android 8.0 (API level 26) and later, the getMetrics() method is available for some media classes. It returns a PersistableBundle object containing configuration and performance information, expressed as a map of attributes and values. The getMetrics() method is defined for these media classes:

- MediaPlayer.getMetrics()
- MediaRecorder.getMetrics()
- MediaCodec.getMetrics()
- MediaExtractor.getMetrics()

Metrics are collected separately for each instance and persist for the lifetime of the instance. If no metrics are available the method returns null. The actual metrics returned depend on the class.