Images haven't loaded yet. Please exit printing, wait for images to load, and try to print again.

# Local mp3 Files (With Seek Functionality) in Flutter

Sam Upra Follow
Jan 7 · 6 min read

## Introduction

The purpose for writing this article is to be a full start-to-end guide for playing local mp3 files in Flutter for both the **iOS and Android** platform. All the youtube videos and code examples that I could find provided too less or were either outdated. So the purpose of this guide is to replace them and to help anyone that is interested in creating a music player (with seek functionality) for both platforms. Today, we will be writing code in Flutter that involves all the following functions that a local music player could need(and the corresponding widget): seek, play, stop, pause.

Code is provided at the end of this article.

## Play

First, we will be using the `audioplayers` flutter plugin. Let's grab it by adding the following line to our dependencies in `pubspec.yaml` :

```
audioplayers: ^0.7.8
```

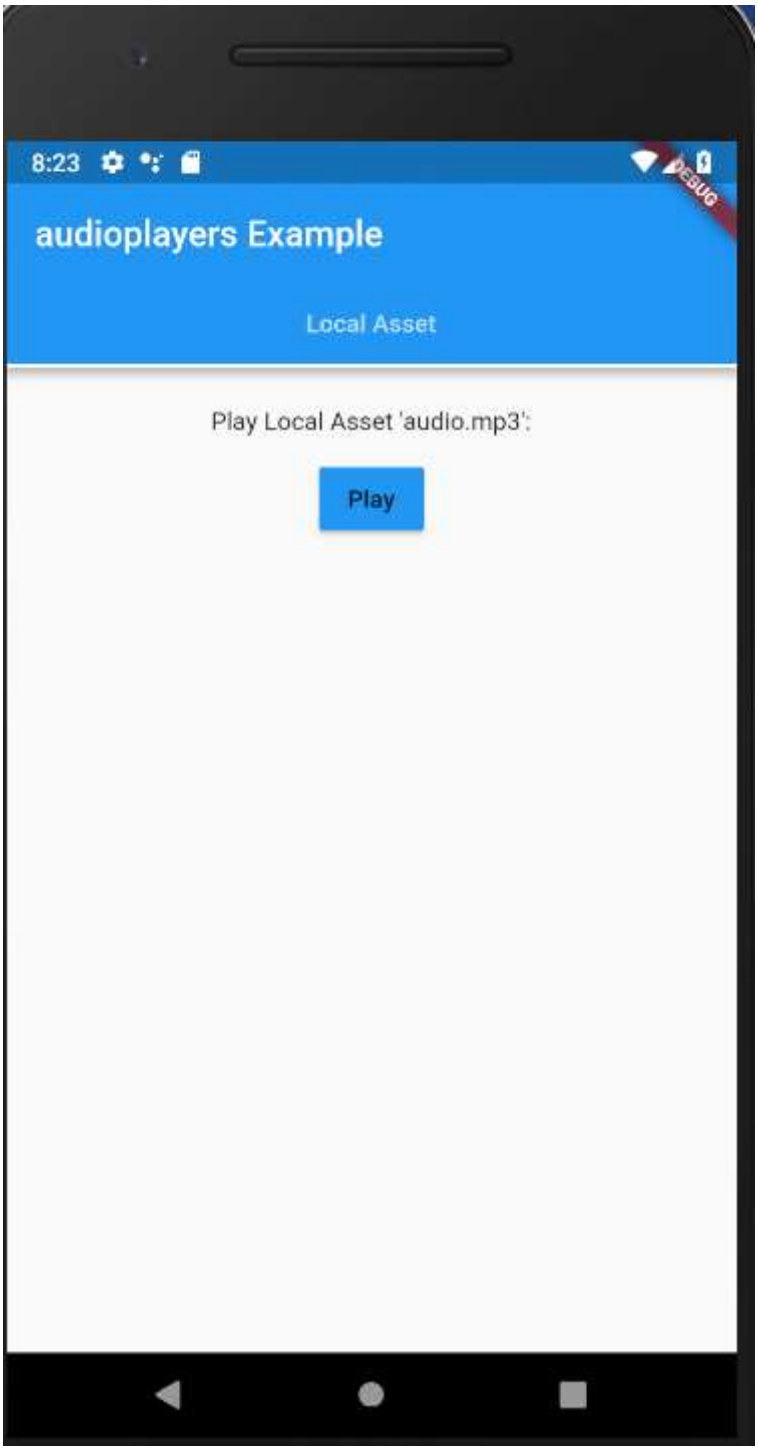
Now that we have the required dependency, we can use the code example provided by audioplayers as the base for our application. The code example contains two dart files (that separates the widget and the code), but for this tutorial we will combine them into one single file `main.dart` . We will also remove all non-related code as the code example contains four tabs but only one is concerned with playing local assets.

This is what is left of our `main.dart` after removing what we don't need.

```
import 'package:audioplayers/audio_cache.dart';
import 'package:audioplayers/audioplayers.dart';
import 'package:flutter/material.dart';

typedef void OnError(Exception exception);

void main() {
  runApp(new MaterialApp(home: new ExampleApp()));
}

class ExampleApp extends StatefulWidget {
  @override
  _ExampleAppState createState() => new _ExampleAppState();
}

class _ExampleAppState extends State<ExampleApp> {
  AudioCache audioCache = new AudioCache();
  AudioPlayer advancedPlayer = new AudioPlayer();
  String localFilePath;

  Widget _tab(List<Widget> children) {
    return Center(
      child: Container(
        padding: EdgeInsets.all(16.0),
        child: Column(
          children: children
              .map((w) => Container(child: w, padding:
EdgeInsets.all(6.0)))
              .toList(),
        ),
      ),
    );
  }

  Widget _btn(String txt, VoidCallback onPressed) {
    return ButtonTheme(
        minWidth: 48.0,
        child: RaisedButton(child: Text(txt), onPressed:
onPressed));
  }

  Widget localAsset() {
    return _tab([
      Text('Play Local Asset \'audio.mp3\':'),
      _btn('Play', () => audioCache.play('audio.mp3'))
    ]);
  }

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: 1,
      child: Scaffold(
        appBar: AppBar(
          bottom: TabBar(
            tabs: [
              Tab(text: 'Local Asset'),
            ],
          ),
          title: Text('audioplayers Example'),
        ),
        body: TabBarView(
          children: [localAsset()],
```

```
            ),
          ),
        );
      }
    }
```

At this stage, this is how it looks like in the emulator:




Of course, this won't work yet because we haven't added our asset `audio.mp3` and declared an asset folder. So let's go ahead and do that.

First, we will go grab a mp3 file and then create an `assets` folder in the root directory of our project and add that file in this directory.

We also need to make flutter aware that we have created an assets folder, so we will go ahead and add the following line under the `flutter` key in our `pubspec.yaml` :

• • •

```
assets:
  - assets/
```

After adding our mp3 file called `audio.mp3` in this example, we will go back and hot-reload our emulator and click play. Viola! Audio should be running now!

## A pause button would be nice?!

At this stage, we can only play the audio, so let's work on adding more functionality such as `pause` , `play` , `stop` and `seek` . Lets get to it one by one.

## AudioPlayer

At this point, we are using the `AudioCache` to play our local audio. If we inspect the source code of the `AudioCache` , and more specifically its constructor—we will notice the following lines:

```
/// This is an instance of AudioPlayer that, if present, will always be used.
///
/// If not set, the AudioCache will create and return a new instance of AudioPlayer every call, allowing for simultaneous calls.
/// If this is set, every call will overwrite previous calls.
AudioPlayer fixedPlayer;

AudioCache({this.prefix = "", this.fixedPlayer = null});
```

What this means is that, if we want to have control over our audio player, we will need to pass/grab an instance of the `AudioPlayer` which is returned from the `play()` call, or simply passed through the constructor. So let's go ahead and modify our code to gain control over the player. I have chosen the approach of simply passing an instance of the `AudioPlayer` to the `AudioCache` .

• • •

```
AudioPlayer advancedPlayer = new AudioPlayer();
AudioCache audioCache = new AudioCache(fixedPlayer:
advancedPlayer);
```

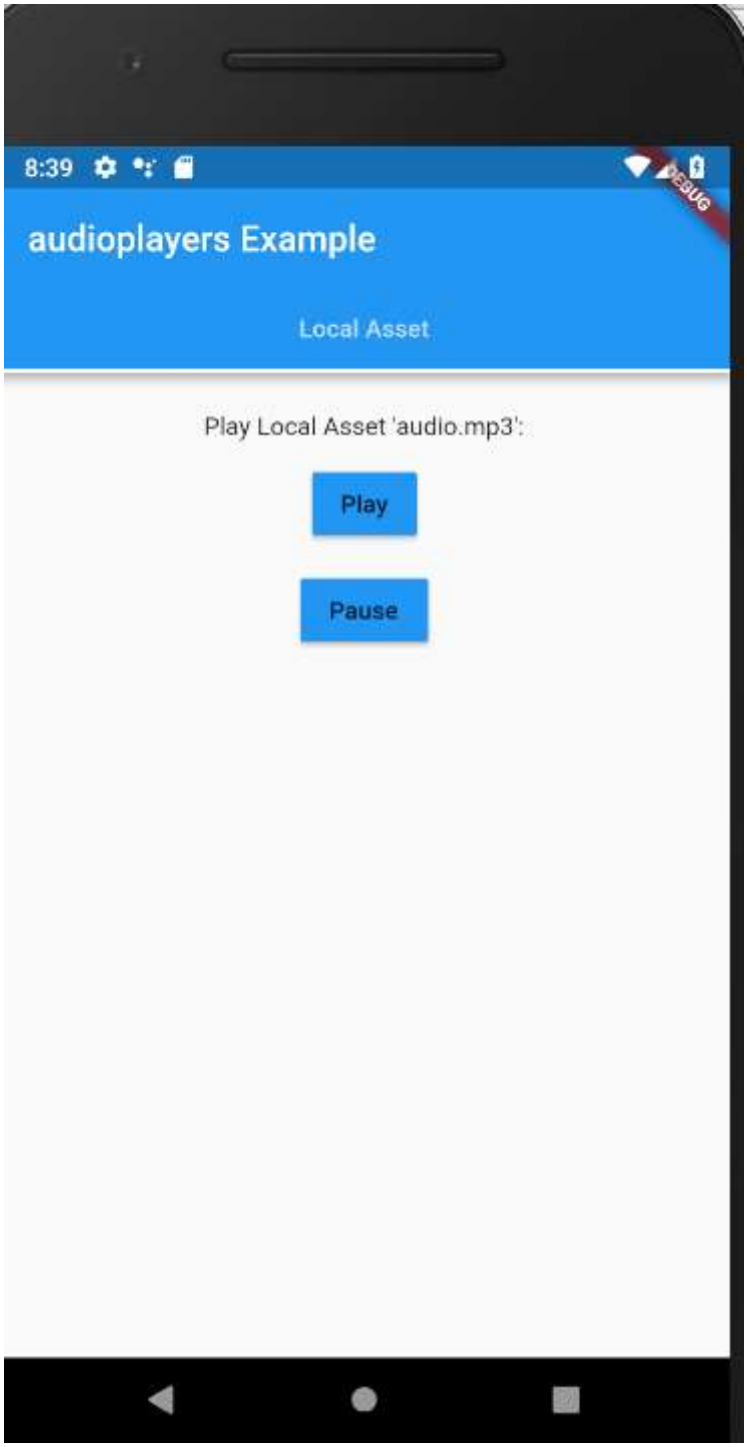
## Pause

Now, we can access all of the commands that the fixed player provides such as pause, seek and stop. So first of all, let's modify our code and add a `pause` button. Our `localAsset()` function now looks as follows:

• • •

```
Widget localAsset() {
  return _tab([
    Text('Play Local Asset \'audio.mp3\':'),
    _btn('Play', () => audioCache.play('audio.mp3')),
    _btn('Pause',() => advancedPlayer.pause())
  ]);
}
```

If you notice, you can see that we are actually referring to the `fixedPlayer` in this case and are no longer concerned with the `audioCache` . We will notice this *trend* for the rest of the functions that we are interested in. After we hot-reload the application, we should now see the following widget in our emulator:

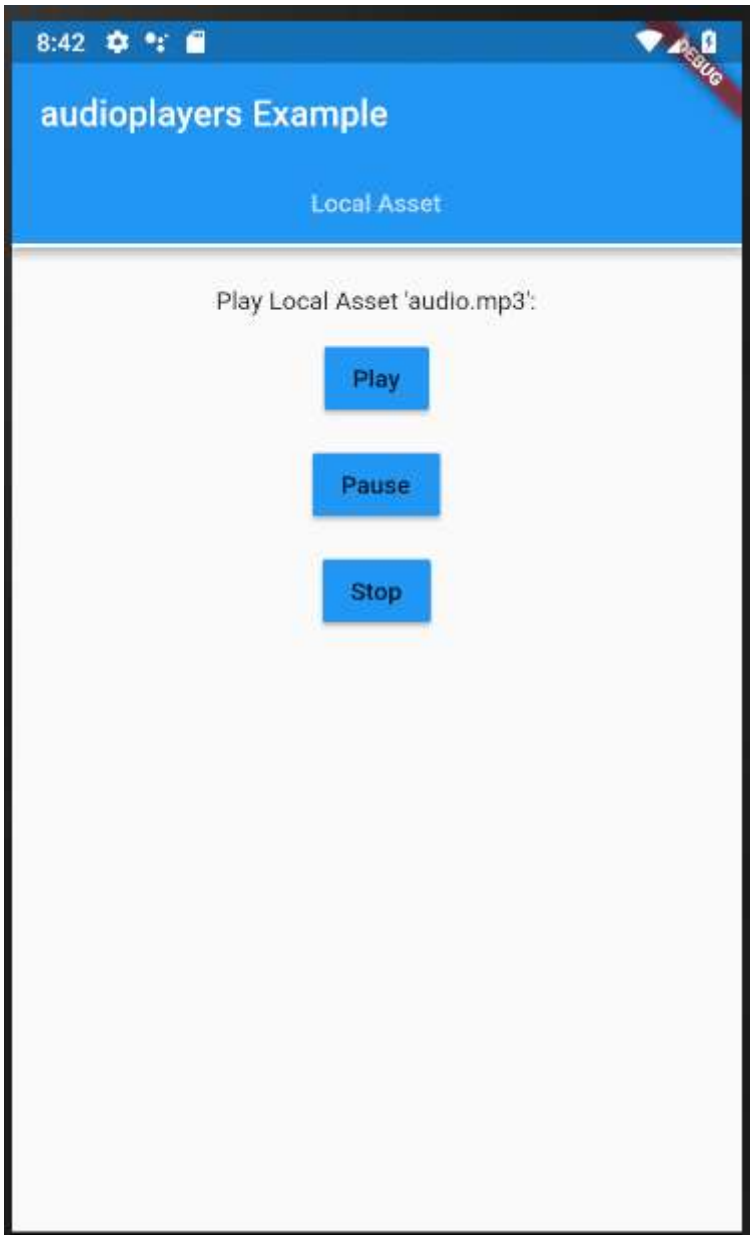



## Stop

Great! Now we can play and pause and audio. But what about stop? That's really simple, let's go back to our `localAsset()` function and add the following line after our `pause` line :

```
_btn('Stop', () => advancedPlayer.stop())
```

Now, our emulator should look as follows and be able to stop audio!




It doesn't look fancy enough and that makes me sad. All the audio players that I am used to have a slider. I'm sure you're the same, so let's go ahead and build one!

## Seek

There are two things that we need to keep track of, in order to make a seek bar work:

1. Current position of the song
2. Total duration of the song

So let's go ahead and declare these two variables in our `ExampleAppState` class as we will need them later. We will set a default value so that we don't have to worry about NPE's and stuff like that.

```
Duration _duration = new Duration();
Duration _position = new Duration();
```

Luckily for us, the `AudioPlayer` class provides a `durationHandler` and `positionHandler` and these handlers constantly update the position and handler once it is available. But first, let's refactor our code and add a `initPlayer()` function that handles all the initialization. We will be using the `initState()` function to call the player's initialization function as follows:

```
@override
void initState(){
  super.initState();
  initPlayer();
}

void initPlayer(){
  advancedPlayer = new AudioPlayer();
  audioCache = new AudioCache(fixedPlayer: advancedPlayer);

  advancedPlayer.durationHandler = (d) => setState(() {
    _duration = d;
  });

  advancedPlayer.positionHandler = (p) => setState(() {
  _position = p;
  });
}
```

Now we have a way to grab the duration and the position of the song. That's all we need to create a slider! So lets do it. The function `slider()` returns the slider Widget and its used in the `localAsset()` function which we are used to by now.

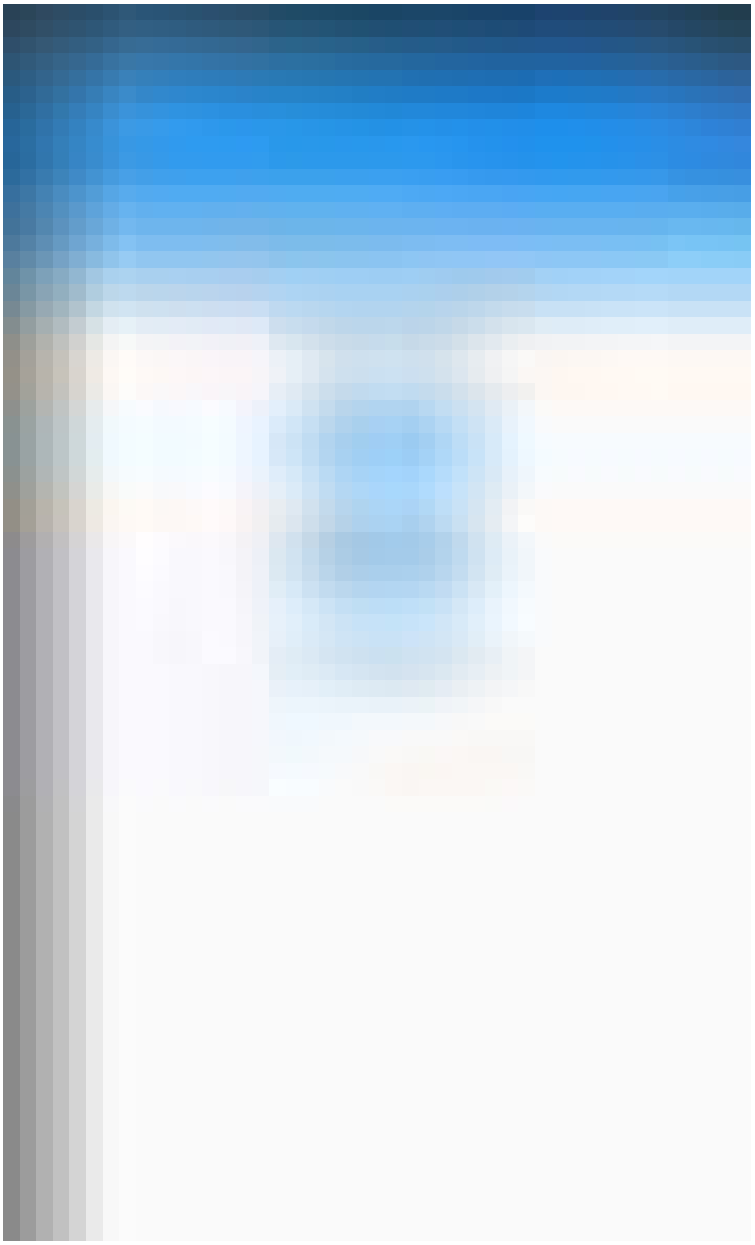
```
Widget slider() {
  return Slider(
      value: _position.inSeconds.toDouble(),
      min: 0.0,
      max: _duration.inSeconds.toDouble());
}
```

```
Widget localAsset() {
  return _tab([
    Text('Play Local Asset \'audio.mp3\':'),
    _btn('Play', () => audioCache.play('audio.mp3')),
    _btn('Pause',() => advancedPlayer.pause()),
    _btn('Stop', () => advancedPlayer.stop()),
    slider()
  ]);
}
```

There is a problem though, we haven't specified a way to dynamically change the slider value and to give the user control over the exact second of the song. We can do this by the built in function `onChanged()` which is provided in the Slider widget. So first, let's go ahead and create a seek function which seeks to the exact position in the song, and modify our previous slider code to call it:

```
void seekToSecond(int second){
  Duration newDuration = Duration(seconds: second);

  advancedPlayer.seek(newDuration);
}


Widget slider() {
  return Slider(
      value: _position.inSeconds.toDouble(),
      min: 0.0,
      max: _duration.inSeconds.toDouble(),
      onChanged: (double value) {
        setState(() {
          seekToSecond(value.toInt());
          value = value;
        });});
}
```

This is how it looks like in our emulator, and the slider works as expected, by seeking to the exact second that we are interested in.



That's it! We're done for now. If you enjoyed reading this, please clap and star the repo on github.

Code is provided here in the `main.dart` file:

https://github.com/samupra/local_flutter_audio_player