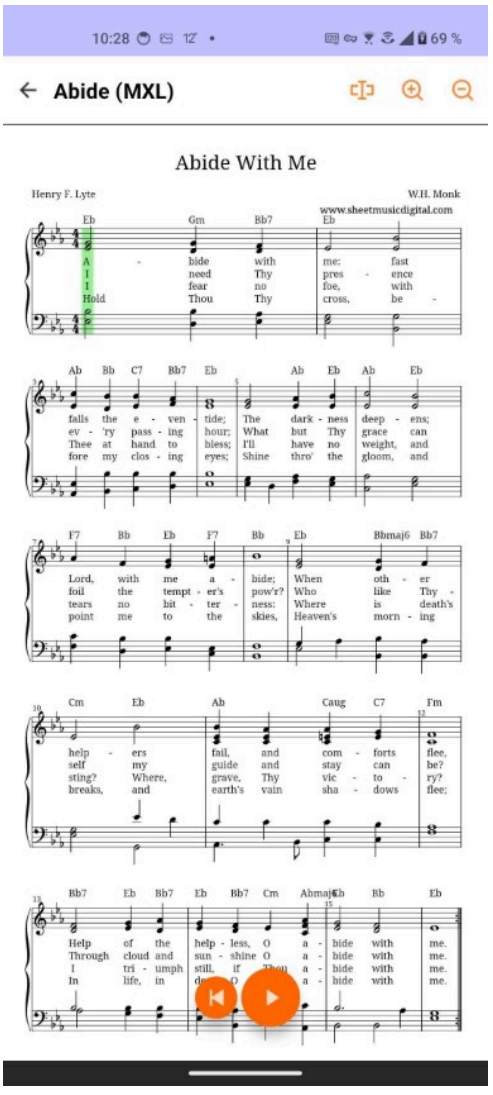
osmd-native

OpenSheetMusicDisplay for React Native & Kotlin / Compose
This repo contains sources for three platform libraries:

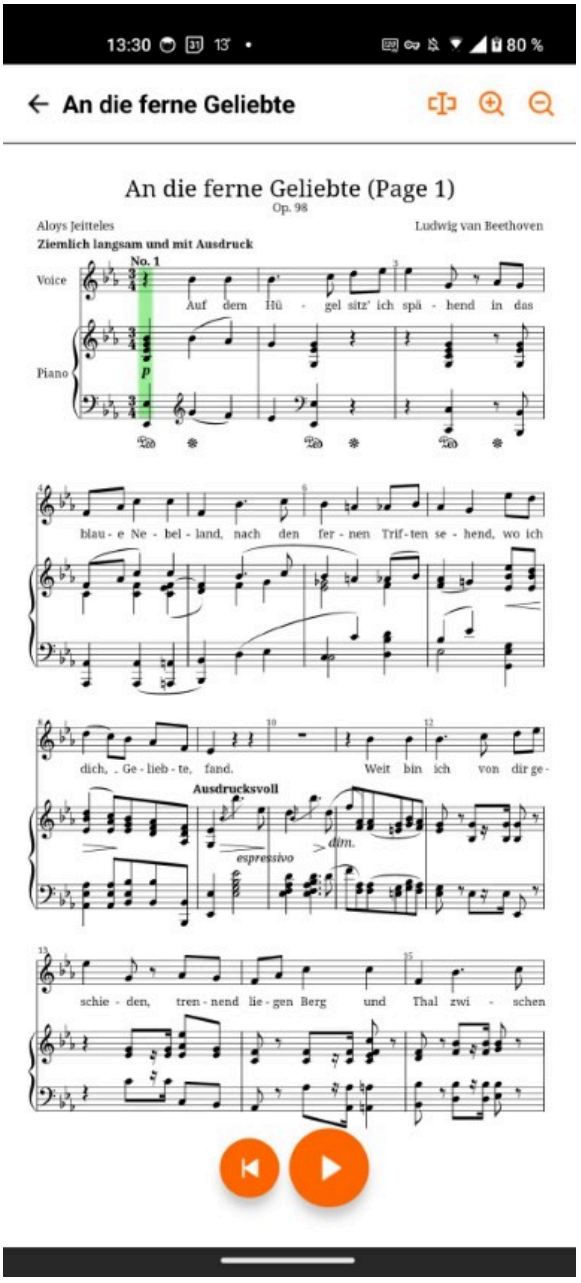
- osmd-kotlin for the Kotlin / Compose version
- react-native-osmd for the React Native version
- osmd-swift for the Swift / iOS Version (Readme coming soon)

Kotlin



Kotlin

React Native



React Native

osmd-kotlin

OpenSheetMusicDisplay for Kotlin / Compose Currently supports:

- setting OSMDOptions via props
- loading a local or remote music xml file
- playing audio & controlling playback
- zoom in / out
-

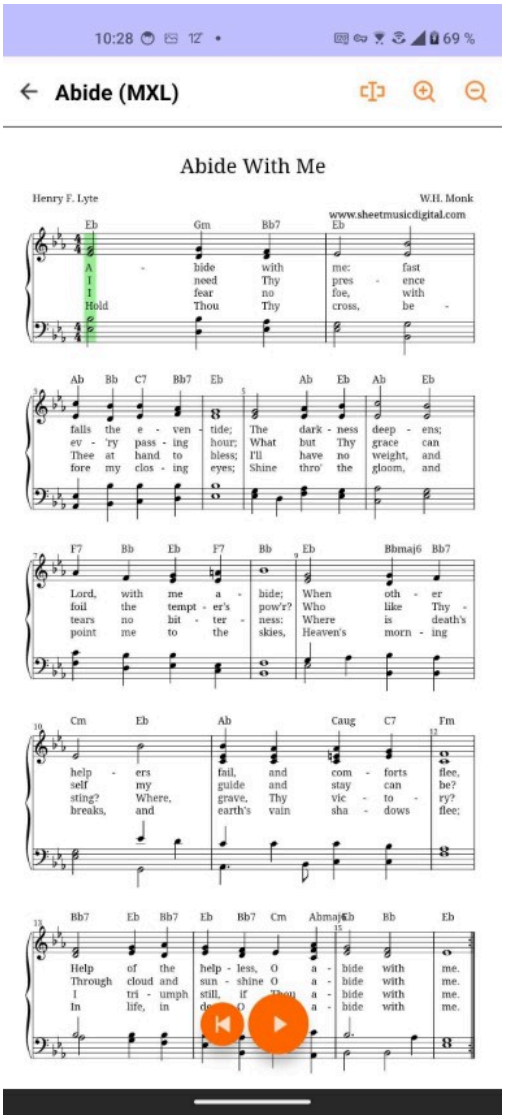


Table of contents

- Interface
- Updating OSMD
- Building & Publishing

Usage

Simplest usage rendering a music sheet:

```
import org.opensheetmusicdisplay.osmd.kotlin.OSMD

// path to a music xml file, either local or remote

val musicXML = "https://appassets.androidplatform.net/assets/AbideWithMe.mxl"

// osmd object controlling playback, zoom & cursor

val osmd = OSMD()

// ... in your composable layout:

Column {

  osmd.OSMDView(musicXML)

}
```

Note: for local files, the file path passed via `musicXML` needs to be in the `assets` folder within the main app directory.

Examples

See `./app` for an example kotlin app using this library.
Some usage scenario examples:

Controlling audio playback

https://github.com/user-attachments/assets/a1d053a1-18e0-4d27-8f8d-f78bec6fcc5f

Audio playback can be controlled via play / pause / stop methods on the `osmd` instance:

```
  val osmd = OSMD()

  // ...

  osmd.play()   // start playback

  osmd.pause()  // pause playback at current position
```

```
osmd.stop()   // stop playback and reset to beginning
```

Changing cursor color

https://github.com/user-attachments/assets/c4f96875-d16c-4ac1-9f11-168a184deb74

Cursor color can be set on the `osmd` instance:

```
osmd.setCursorColor('#f00')
```

Zoom In/Out

https://github.com/user-attachments/assets/feb45af8-51ed-42ad-9ae5-a0392d39ab69

Zoom scale can be set on the `osmd` object (default is `1.0`):

```
osmd.setZoom(1.1)
```

Development

Setup

Make sure your environment is setup for Android Studio & Kotlin w/ Jetpack Compose.
Check https://developer.android.com/develop/ui/compose/setup

1. Clone the repo
2. Open the project in Android Studio

The project will include both the example `app` and the `osmd-kotlin` lib, which is the source of the `OSMD` class providing functionality for audio playback, zoom, options & the composable `OSMDView` component. You can simply run the example app during development for testing.

Structure

The project directory has the following structure:

```
[root]                  (root project directory)
 ├─ app                 (kotlin example app source)
 ├─ [/../] osmd-kotlin  (lib source)
 ├── assets             (opensheetmusicdisplay.min.js & init scripts/html)
```

The architecture of this lib can be summarized like this:

- An OSMD build is encapsulated inside a skeleton webview that loads nothing but an empty html page with a single container inside to load OSMD into
- The `InjectionScripts.kt` file contains js that can be passed to and launched inside the webview to load OSMD, set options, load & render a music sheet and control playback within Kotlin. These scripts essentially expose the actual OSMD functionality.

With that setup, the kotlin library is defined via the `OSMD` class - it exports functions for playback control, cursor & zoom settings and the composable `OSMDView` component which is the main view component that renders a given music xml.

Interface

View Props

```
/**

 * The composable OSMDView rendering a music sheet.

 *

 * @param musicXML the path to the music sheet file (.xml or .mxl inside assets folder)

 * @param options optional list of OSMD options (see https://github.com/opensheetmusicdisplay/osmd-types-player )

 * @param onRender optional function callback to be after render (i.e., for loading indicators etc.)

 */

 @Composable

 fun OSMDView(musicXML: String, options: JSONObject? = null, onRender: (() -> Unit)? = null)
```

Object methods

```
val osmd = OSMD()

// ...

osmd.play()   // start playback

osmd.pause()  // pause playback at current position

osmd.stop()   // stop playback and reset to beginning
```

osmd.setCursorColor('#f00') // sets the color of the cursor

osmd.setZoom(1.1) // sets the zoom scale of the rendered sheet

Updating OSMD

If a new OSMD build is available, you'll need to update `opensheetmusicdisplay.min.js` in `osmd-kotlin/src/main/assets`

react-native-osmd

OpenSheetMusicDisplay for React Native
Currently supports:

- setting OSMDOptions via props
- setting a musicXML string or URL via props
- playing audio & controlling playback
- zoom in / out

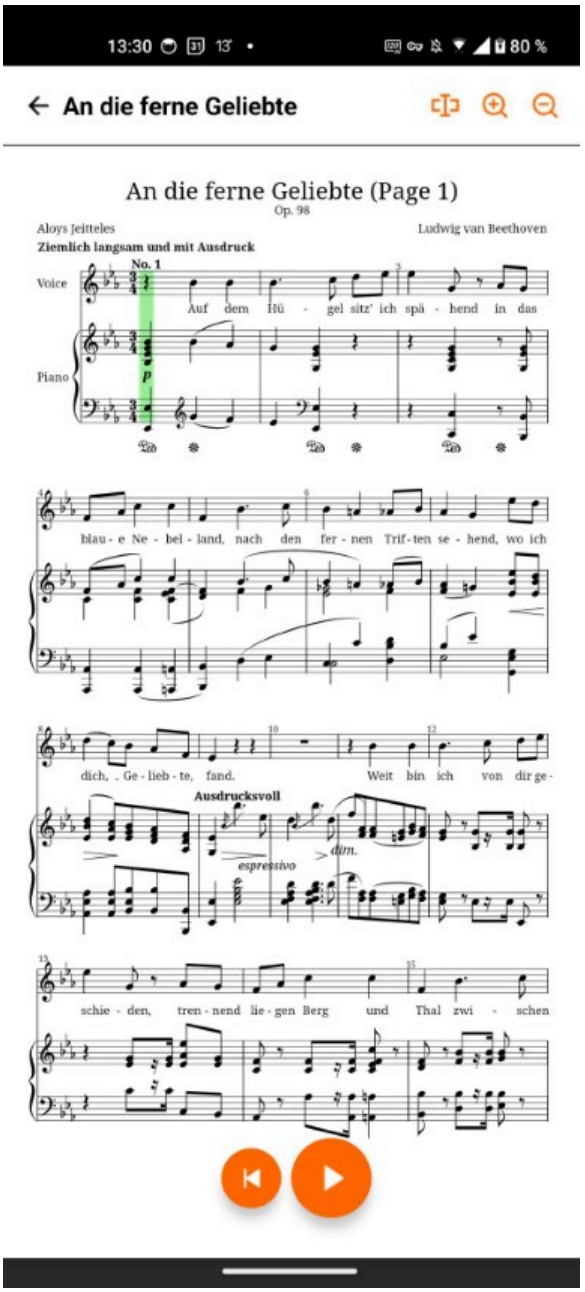


Table of contents

Usage

Simplest usage rendering a music sheet:

```
import { OSMDView } from 'react-native-osmd';

// this is a .ts file exporting a string

import { beethoven_geliebte } from '../assets/beethoven_geliebte';

// ...

<OSMDView

  options={{

    // optional, use whatever options you wish as supported by IOSMDOptions

    backend: 'svg',

    drawTitle: true,

    drawingParameters: 'leadsheet',

  }}

  musicXML={beethoven_geliebte}

/>
```

Note: Currently, you need to pass either a remote URL or a string to `musicXML` - passing a .xml file directly is not yet possible.

Examples

See `./example` for an example app using this library, you can run it by running `yarn example android` or `yarn example ios` (if on macOS).
Some usage scenario examples:

Controlling audio playback

https://github.com/user-attachments/assets/0408413c-6ea4-4409-b11c-4bb6d71acfd4

Audio playback can be controlled via play / pause / stop methods on the `OSMDView` ref:

```
const osmd = useRef<OSMDRef | null>(null);

// ...

<OSMDView

  ref={osmd}

  options={options}

  musicXML={musicXML}

/>

// ...

osmd.current?.play();   // start playback

osmd.current?.pause();  // pause playback at current position

osmd.current?.stop();   // stop playback and reset to beginning
```

Changing cursor color

https://github.com/user-attachments/assets/2b552498-520e-4207-b3f8-7480a72a824f

Cursor color can be set on the `OSMDView` ref directly:

```
osmd.current?.setCursorColor('#f00');
```

Zoom In/Out

https://github.com/user-attachments/assets/ef046c2d-ce80-4d93-9417-807886b77018

Zoom scale gan be set on the `OSMDView` ref (default is `1.0`):

```
osmd.current?.setZoom(1.1);
```

Development

Setup

Make sure your environment is setup for node.js & react-native. Check https://reactnative.dev/docs/environment-setup *(do **not** use expo)*.

Note: This project was scaffolded using react-native-builder-bob, it uses yarn with a monorepo configuration so you'll need to use yarn instead of npm.

1. Clone the repo
2. Switch into the project folder & install dependencies: `yarn`
3. Run the example app: `yarn example android` or `yarn example ios` (depending on your OS & target)

This will run the example app from `example/` which imports the library into a very simple client app to showcase the functionality. You can then modifiy the library source code in `src/` and test your changes via hot-reload inside the example app.

Structure

The project directory has the following structure:

[root] (root project directory)

├─ android (native android source files)

├─ example (example client app using this library)

├─ ios (native ios source files)

├─ src (typescript source files)

├── assets (static or generated assets like osmd_min.ts)

├── injection (js code that is injected into the webview containing OSMD)

├── index.tsx (main entry point: exports interfaces and views of the lib)

├─ generate_osmd_min_as_string.js (updates the osmd build asset which is loaded into the webview)

The architecture of this lib can be summarized like this:

- An OSMD build is encapsulated inside a skeleton react-native webview that loads nothing but an empty html string with a single div inside to load OSMD into

- The `injection_scripts.ts` file contains js that can be passed to and launched inside the webview to load OSMD, set options, load & render a music sheet and control playback by passing messages between the webview context & react-native. These scripts essentially expose the actual OSMD functionality

With that setup, the react native library is defined via `index.tsx` - it exports type interfaces and the central `OSMDView` which is the main react-native component that renders a given music xml and exposes methods to the parent component via a forwardRef.

Interface

Component Props

```
/** Defines the properties of the OSMD react component */

export interface OSMDProps {

  /** The music document to render.

   *  It needs to either be a URL to a MusicXML file or

   *  a string of the MusicXML content

   */

  musicXML: string;

  /** (optional) Options to set on OSMD */

  options?: IOSMDOptions;

  /** (optional) Custom styling to be applied to the content container */

  style?: StyleProp<ViewStyle>;

  /** (optional) Callback that is called once the content is rendered.

   *  Can be used to show/hide loading indicators, etc.

   */

  onRender?: () => void;

}
```

Component Interface

```
/** Defines the interface of the OSMD object */
```

```
export interface OSMDRef {

  /** starts audio playback */

  play: () => void;

  /** pauses audio playback at the current position */

  pause: () => void;

  /** stops audio playback and resets to initial position */

  stop: () => void;

  /** sets the osmd cursor color */

  setCursorColor: (color: string) => void;

  /** sets the zoom scale */

  setZoom: (scale: number) => void;

}
```

Updating OSMD

Since `react-native-webview` does not support import of local scripts inside the webview html, we need to pass the osmd build by injecting it as a string via the `injectedJavaScript` prop.

If a new OSMD build is available, you'll need to update `opensheetmusicdisplay.min.js` and then run `node generate_osmd_min_as_string.js` to make sure the `src/assets/osmd_min.ts` file gets updated.