LoopDAW

Mobile Applications Development- 85284 / 85281 – [2016-2017]

Damien Donovan

#20068672

Table of Contents

Introduction

Background

LoopDAW is an Android application designed for the creation of basic multitrack audio recordings. It supports basic user-edits in order to define samples or loops. The process can be repeated to build up a layered stack of clips that are ultimeately combined to make a complete song. This can be used as a practicing aid, a compsotition or production tool, or just for fun. The Android mobile platform is ideal for this kind of functionality – the devices and operating systems used with Android generally have implicit support for audio capture, processing & playback.

The idea for the application developed from personal experience, when I would often find myself with no way to capture a musical idea when away from my home studio. Most Android devices do feature a built-in sound recorder supporting the capture of a single track of audio up to a certain length, and while this is indeed a useful feature, it does not offer some key functions commonly seen in modern audio applications. Musical ideas are generally more complex than a single line of rhythm or melody, and so the ability to individually record and combine multiple seperate tracks is key to a powerful and useful audio app.

Technologies

LoopDAW v1.0 was built in Jetbrains Android Studio v.2.3.1. This version of Android Studio comes bundled with it's own Java 8 runtime environment, and the gradle 3.2 build platform. The application was compiled using Android SDK versions 25, targeting Android versions 21 and above (Lollipop). Test deployments were made to multiple virtual devices of differing API versions and form factors, and to a physical LG Nexus 5X device, running the most recent build of Android 7.1. The excercise of testing across multiple platforms exposed some differences in the native implementations of some Android API classes or methods which actually influenced the subsequent direction of some internal workings of the app in order to achieve better cross-platform support.

Frameworks

Obviously this application like most makes use of the Android development framework. Note this really is a framework and not just an SDK! The Android development framework makes use of the gradle build system architecture and expands on the Gradle paradigm of 'Convention over Configuration'. Along with Android Studio IDE, these three components comprise a full development stack for Android. When creating a project in Android Studio, the gradle build scripts are by default configured such that any later additions to the project will be automatically detected and integrated as part of the next buildscript sync event. Additions to the project under conventional folder structures are also automatically detected and included in the relevant classpath as defined by it's type and location. This level of intelligence within the IDE and default project configuration produces a very dynamic and pleasant development experience right from the outset.

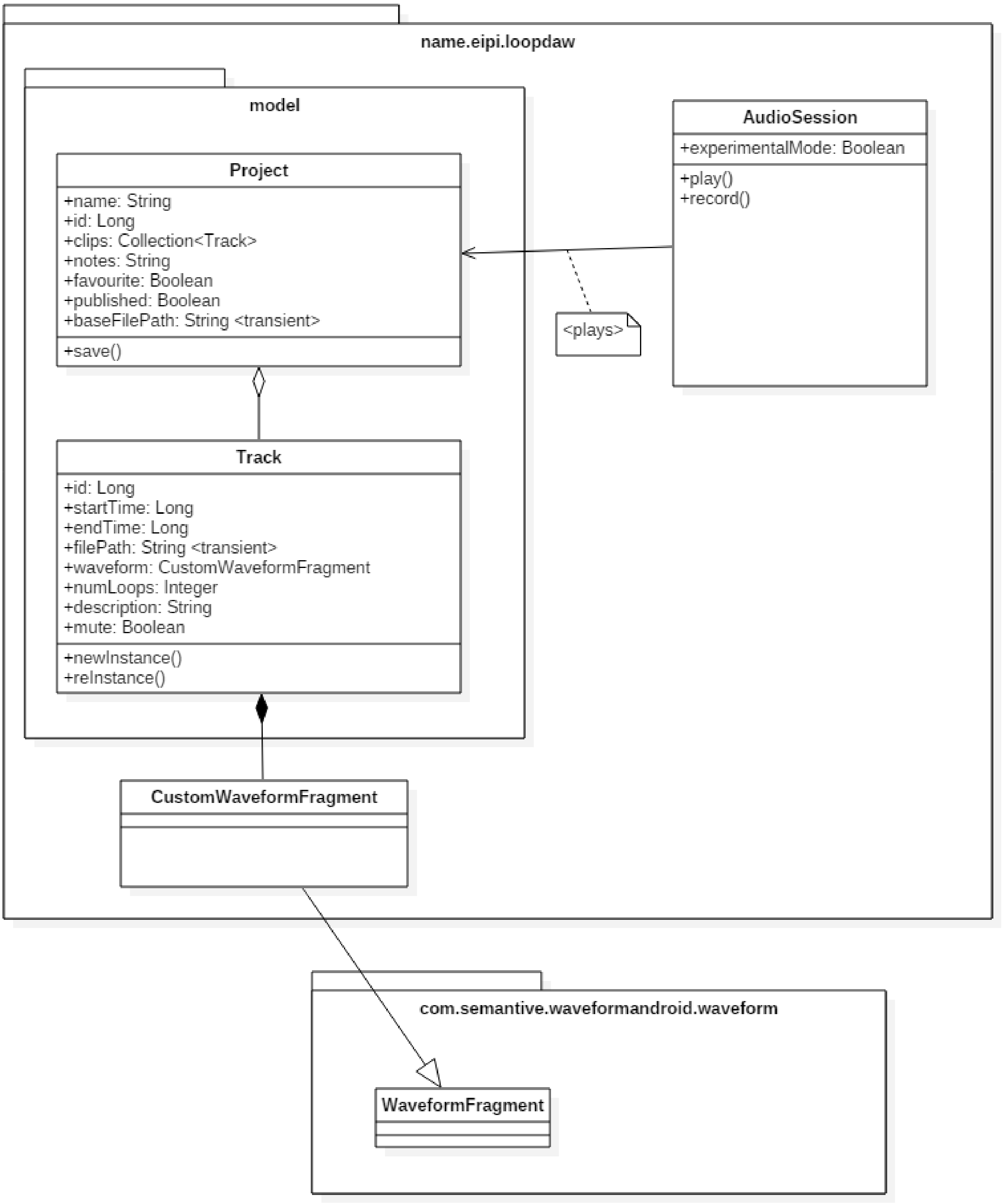
Design

Initially the applications design was based primarily on the sample application being used to illustrate the course content – 'Coffee Mate', but I soon found that this approach to UI desgin and construction was quite limiting and not as dynamic or reactive as I would have liked. After some research I decided to use some features of the 'Material Design' framework, which is part of the Google extended library. Components of the Material Design framework allow you to easily constuct screens that exhibit many of the smooth reactive UI components we would be familiar with from modern native Android applications. Examples include the Card View, Navigation Drawer and Floating Action Buttons.

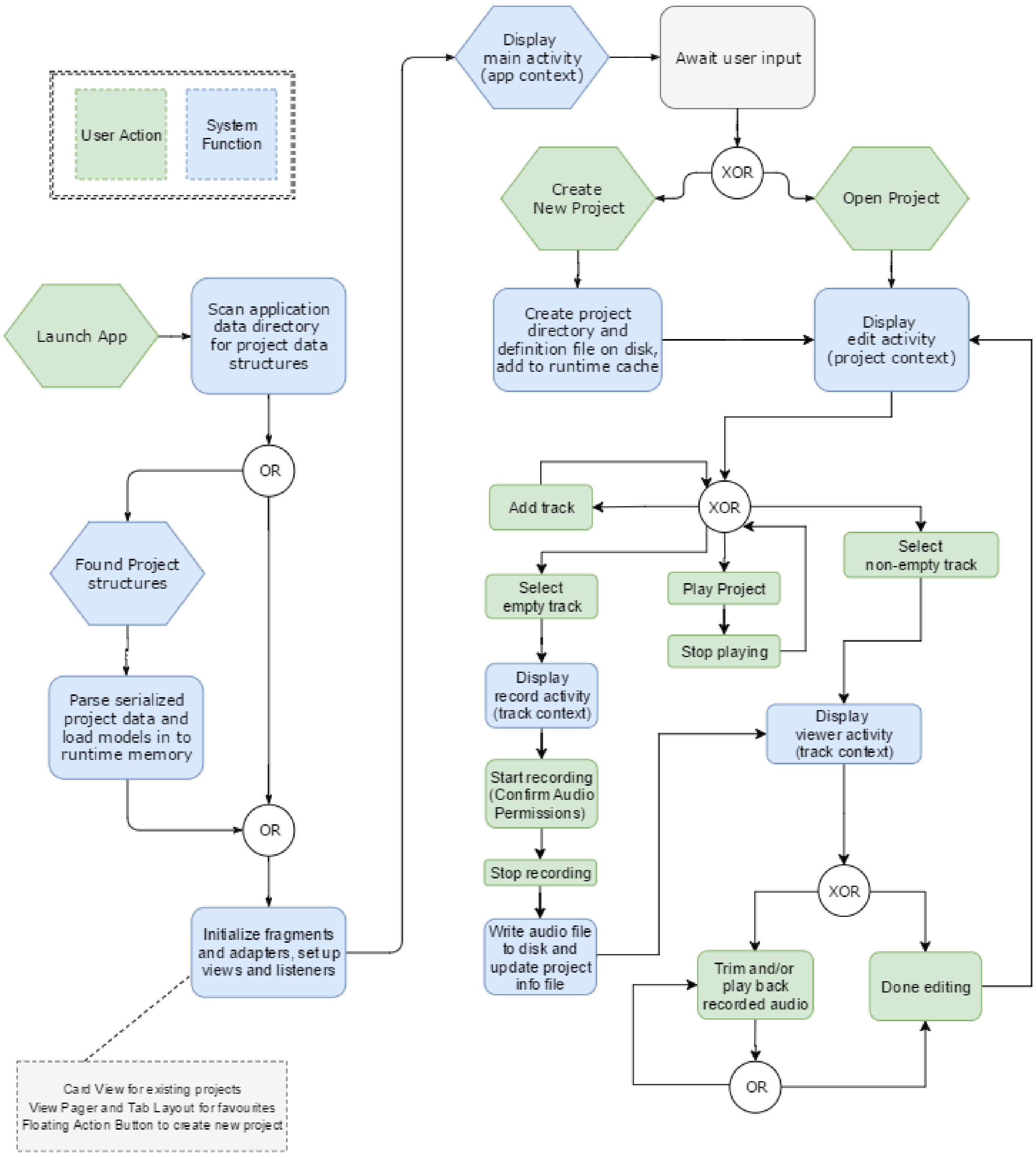
Many of the buttons and icons in the application were implemented as Vector drawables instead of simple images. Vectors provide better edge definition and a smoother overall appearance than raster images, especially when the form factor and scale of the underlying architecture cannot be predetermined, as is the case with most Android development.

Relationships and Patterns

Object Model



Application Flow



User Action
System Function
Launch App
Scan application data directory for project data structures
OR
Found Project structures
Parse serialized project data and load models in to runtime memory
OR
Initialize fragments and adapters, set up views and listeners
Card View for existing projects
View Pager and Tab Layout for favourites
Floating Action Button to create new project
Display main activity (app context)
Await user input
XOR
Create New Project
Open Project
Create project directory and definition file on disk, add to runtime cache
Display edit activity (project context)
XOR
Add track
Select empty track
Play Project
Stop playing
Select non-empty track
Display record activity (track context)
Start recording (Confirm Audio Permissions)
Stop recording
Write audio file to disk and update project info file
Display viewer activity (track context)
XOR
Trim and/or play back recorded audio
Done editing
OR

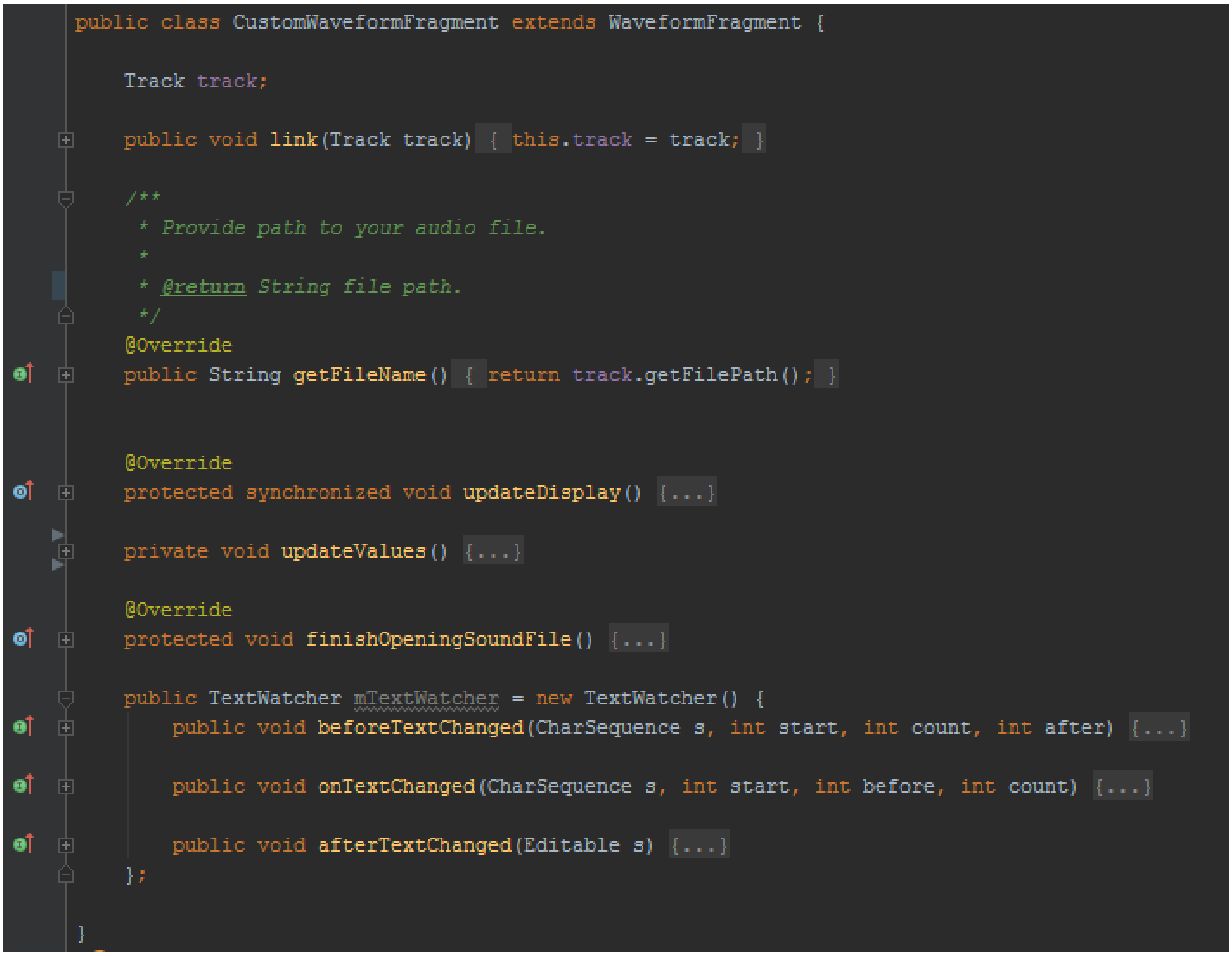
External Libraries

Audio

Waveform-Android

com.github.Semantive:waveform-android:v1.2
https://github.com/Semantive/waveform-android

The waveform-android plugin is an audio visualization component. In LoopDAW, this provides the wave representation of recorded sounds, and constitutes the UI through which the user can trim audio clips to the desired loop-length. This is done by either moving the start and end markers, or by directly entering the start and end times.
LoopDAW implements it's own customized version of this component by extending the core class WaveformFragment in to CustomWaveformFragment. The customized class overrides various setters and listeners in the base class, primarily in order to intercept the start and end times for the loop so that they can be persisted as part of the clip's metadata, and then used in the separate project playback engine.



```
public class CustomWaveformFragment extends WaveformFragment {

    Track track;

    public void link(Track track) { this.track = track; }

    /**
     * Provide path to your audio file.
     *
     * @return String file path.
     */
    @Override
    public String getFileName() { return track.getFilePath(); }


    @Override
    protected synchronized void updateDisplay() {...}

    private void updateValues() {...}

    @Override
    protected void finishOpeningSoundFile() {...}

    public TextWatcher mTextWatcher = new TextWatcher() {
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {...}

        public void onTextChanged(CharSequence s, int start, int before, int count) {...}

        public void afterTextChanged(Editable s) {...}
    };

}
```

ExoPlayer

`com.google.android.exoplayer:exoplayer:r2.3.0`

http://google.github.io/ExoPlayer/

ExoPlayer is an open source media player built on top of Android’s low level media APIs. The native Android MediePlayer and MediaRecorder classes are sufficient for very basic audio tasks like recording and playing back individual sounds one at a time. However, it does not directly provide any apis or methods for more advanced but still relativly simple use cases like combining two audio sources, playing subsections of a file, or setting loops. Though all of these things could ultimately be accomplished with the native audio classes, it would require a good deal of custom coding, some of which would be very sensitive to algorithmic efficiency in terms of application performance and ultimately user-experience.

ExoPlayer provides an easy to use set of implementations of it's MediaSource interface. Each MediaSource implementation offers a different aspect of control over the audio source's playback conditions, and ultimately multiple different MediaSources can be combined before being sent to the player itself for playback.

In LoopDAW, the primary MediaSource implementations used are ClippingMediaSource (to set the length of the loop section), and LoopingMediaSource (to set continuous playback of each ClippingMediaSource). The ExoPlayer library also offers it's own interface components and playback controls, but I did not find these suitable for the LoopDAW interface so used my own custom UI components to control the underlying player.

Java Advanced Audio Decoder

`net.sourceforge.jaadec:jaad:0.8.5`

http://jaadec.sourceforge.net/

Java Advanced Audio Decoder is a library to parse raw audio data from a number of compressed audio formats, including the AAC format used to record LoopDAW audio files. Once the raw audio samples have been decoded from the compressed and encoded source, they can be added together using simple summation (with consideration for clipping of the summed output).

Google Play Services

```
com.google.android.gms:play-services:9.0.0
com.google.android.gms:play-services-auth:9.0.0
```

Google Play Services allow the application to access the user's google account. This is done primarily in order to access the Google Drive API. Recording a large number of audio files on a mobile device may have an insignificant impact on disk space, so integration with a remote storage service like Google Drive would allow the user to store some or all projects remotely, and then re-download them individually or in bulk to the device as required.

Material Design

```
com.android.support:support-vector-drawable:25.3.1
com.android.support:design:25.3.1
com.android.support:cardview-v7:25.3.1
com.android.support:recyclerview-v7:25.3.1
```

https://material.io

https://www.materialpalette.com/lime/grey

Material Design is more of a concept or specification than an actual library, but Google have made available a number of component libraries that implement the specification, to varying degress. These libraries provide ready-to-use components that can be easily combined in a dynamic and adaptive way, producing a smooth and intuitive user experience. The Material components, though no more difficult to integrate than standard components, give the application a more professional and polished feel. Something as simple as the addition of shadows provides a sense of depth and substance in the application's UI. The shading in Google's Material components are rendered by the devices actual graphics processor, if supported, reducing the risk of locking up the UI with expensive graphical operations on the main processor.

Application Features & Design Considerations

User Interface

LoopDAW UI is made up of a number of different layouts of different types. Fragments for project and track entities, and custom waveform fragments for advanced audio views. To assist in user navigation and experience there are popups for text entry with basic validations and Snackbar messaging. The main screen uses a modern looking card view, wrapped in a tab layout to provide easy navigation to favourite projects. There are only a few screens in total, but each provides convenient access to all required functions.

Data Model & Persistence

The Project is the primary entity in LoopDAW. An application have have 0 or more projects defined at any given time. There is no effective limit to the number of projects that can exist in once, as the project definition itself is very small. Each project consists of a folder structure containing a project.info file. The info file is a simple ASCII text file containing a JSON representation of the serialized Project class instance.

Each project has a collection of tracks, each representing an individual audio file on disk. The audio files themselves are stored within the project folder. They are created by the Android Media Recorder class using the aac codec. Any metadata defined by the user for the recorded audio file is stored in the track object, and becomes serialized as part of the parent project entity on save.

When the application is re-started, an application lifecycle event in the Application extenion class scans the application's working directory for project folders and deserializes the project and track data in to memory for runtime usage. The audio files themselves are not loaded in to memory until actually specifically needed, and even then they are only streamed, not loaded in full.

Some fields within the Track and Project objects related to file paths are defined as transient and as such are not serialized with the other project data. These fields are recreated from first principles each time the application is opened in order to achieve better application portability.

Audio Engine

Core Engine

The core audio engine is built using basic native Andoid Audio classes and the ExoPlayer extension, along with the LoopDAW Project and Track entities. LoopDAW defines an additional helper class called AudioSession. An audio session instance is created for each project when it is opened. The AudioSession is responsible for all audio operations within that project. When the project is closed or suspended, the audio session is destroyed safely, and new ones created as needed.

The Audio record function is a simple wrapper for Android's native MediaReorder functionality. It uses the devices inbuilt microphone (once the relevant user permissions are obtained) to record ambient audio and stream to a file via AAC encoding. The path and name for the file are determined from the project and track context from which the record function is invoked.

Playback can be performed in either track or project context.

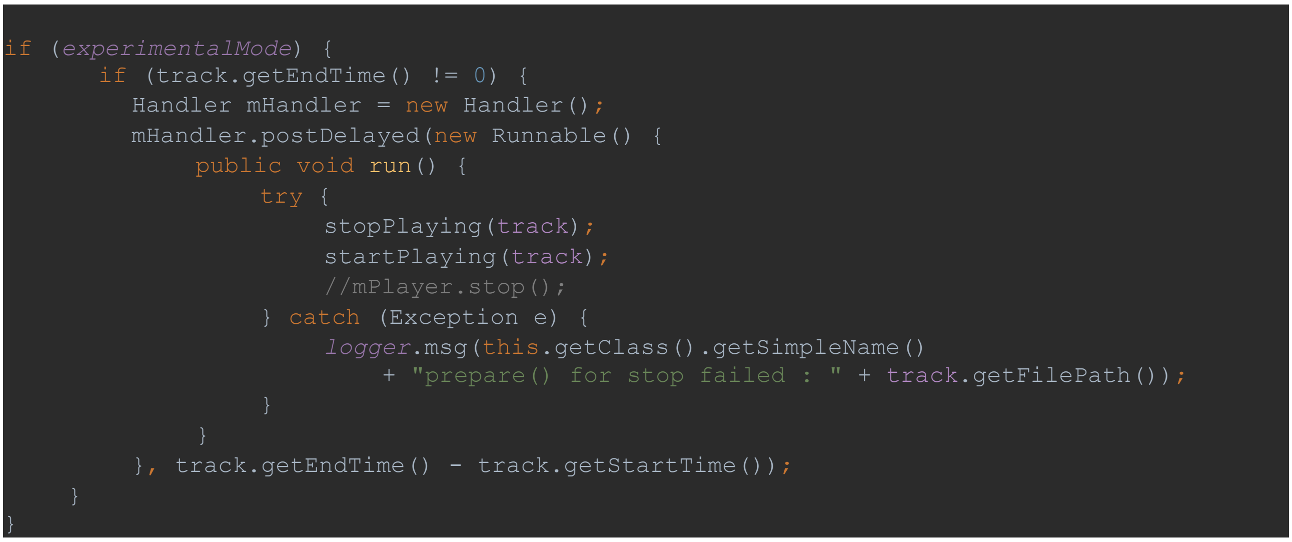
Track context playback allows the playback of a single track in isolation, and is most often done while reviewing a recording or defining a loop section within a track. Playback in this context is provided by the WaveformAndroid component.

Proect context playback allows the playback of all tracks within a project at once (with the ability to mute some tracks). This functionality is implemented using a combination of the ExoPlayer library and custom logic within LoopDAW AudioSession class.

Alternative Audio Rendering Implementation

In addition to the standard looping playback mode, LoopDAW offers an 'experimental' more dynamic looping playback method. Instead of defining the loop sections once at the start of playback, experimental mode resets the loop parameters with each playback. This means that new players are spawned and destroyed with each loop, which can be processor and memory intensive, resulting in spikes in memory usage and more frequent garbage collection.

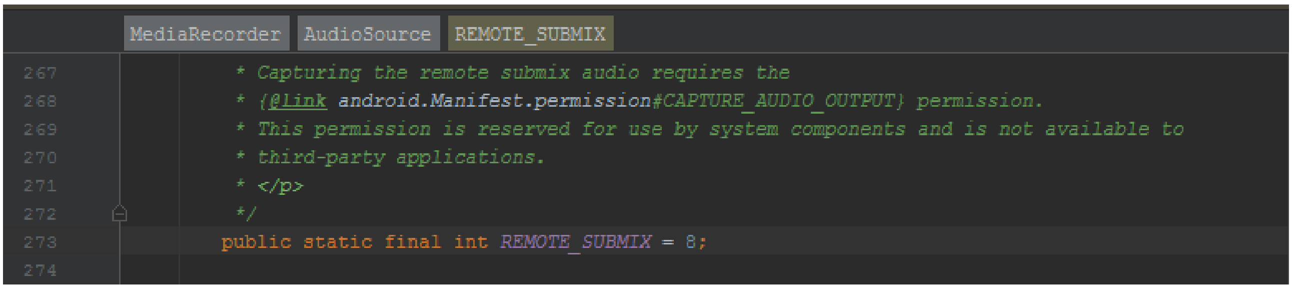
However, playback in this method offers a more interesting and explorative user experiencing, allowing loop sections to be redefined during playback, and a more dynamic result achieved.



```
if (experimentalMode) {
    if (track.getEndTime() != 0) {
      Handler mHandler = new Handler();
      mHandler.postDelayed(new Runnable() {
         public void run() {
            try {
               stopPlaying(track);
               startPlaying(track);
               //mPlayer.stop();
            } catch (Exception e) {
               logger.msg(this.getClass().getSimpleName()
                  + "prepare() for stop failed : " + track.getFilePath());
            }
         }
      }, track.getEndTime() - track.getStartTime());
   }
}
```

Authors Viewpoint

Earlier in the design exploration phase of the project, I had intended to implement the ability to render projects as a single audio file and save to disk. That is, that the user could record multiple audio tracks, define looping sections independently for each individual track as desired, and then finally render the entire multitrack project down to a single audio file. Soon after I started investigating this in depth though, it became clear that this would not be a trivial task on an Android device. Although the hardware itself is perfectly capable of performing such a task, the Android framework implements a number of restrictions around the capture and routing of audio, even when all opertations are confined within a single application. It would stand to reason that these restrictions have been implemented due to the security risk of applications gaining unauthocrized access to a user's phone calls or other sensitive audio input or passthrough.



```
     * Capturing the remote submix audio requires the
     * {@link android.Manifest.permission#CAPTURE_AUDIO_OUTPUT} permission.
     * This permission is reserved for use by system components and is not available to
     * third-party applications.
     * </p>
     */
    public static final int REMOTE_SUBMIX = 8;

```

The specific permission required to capture audio from the audio processor is not available for use in any existing versions of the ADK.

Ref : https://issuetracker.google.com/issues/37015422

If this permission was available, the application could make use of the devices hardware audio processor to perform audio mixing and clipping management, route the summed audio to a File via an OutputStream, and therefore negate the requirement for any software audio processing. That said, this may limit the audio processing to realtime, meaning that a 5 minute track would take exactly that amount of time to actually export to file.

The alternateive approach here is to mix the audio programmatically. In this case you would not be making use of the audio hardware on the device, but rather using mathematical summation and clipping adjustments in code. This approach has the potential to be much faster, potentially rendering minutes of audio in seconds, however it can be complex. Each type of audio file has a distinct header structure, and certain segments of the header will impact how the actual data should be interpreted. If multiple different audio file types can be used, the mixing logic must be capable of parsing each type of file, with logic to parse the headers and convert the compressed audio data back to PWM samples that can be summed or otherwise modified using mathematical operations.

The inclusion of the 'JAAD' library in the projected was intended to support these programmatic rendering functions, but I chose not to complete the implementation of this feature as part of my submission as I felt it was moving outside the main area of focus of the project and module. I will continue to implement this feature outside of the context of the course, as I feel it is inkeeping with the intended scope of the application, and would provide a convenient method of sharing musical ideas with others or between one's own devices.