streamaxia

OpenSDK 3.3 for Android

Setup Guide

Contents

Introduction

Streamaxia OpenSDK is a live streaming (broadcasting) library for mobile devices.

This is a short programming guide about how to integrate Streamaxia OpenSDK library in your Android project.

For the full API documentation, please download the “JavaDocs” folder and run index.html.

Components

OpenSDK Main Interface

The main interface is responsible for the initialization and configuration of the SDK. It provides information about the available features.

Important: Before using the publisher, the SDK must be properly initialized and confi gured.

Handlers and Listeners

Advice: To provide the best user experience, you must treat each state diff erently as per your application logic and desired UX.

RtmpHandler.RtmpListener

The rtmp listener interface represents all the states of the RTMP connection. It allows you to get notified about the latest connection state. You must create a new handler and implement the Rtmp Listener interface in your activity, then pass it over in the constructor as follows:

```
mStreamaxiaPublisher.setRtmpHandler(new RtmpHandler(this));
```

To provide the best user experience, you must treat each state differently.

The states are as follow:

- *onRtmpAuthenticating*
- *onRtmpConnecting*
- *onRtmpConnected*

- *onRtmpVideoStreaming*
- *onRtmpAudioStreaming*
- *onRtmpStopped*
- *onRtmpDisconnected*
- *onRtmpOutputFps*
- *onRtmpVideoBitrate*
- *onRtmpAudioBitrate*
- *onRtmpIOException*
- *onRtmpIllegalArgumentException*
- *onRtmpIllegalStateException*
- *onRtmpSocketException*

RecordHandler.RecordListener

The record listener interface represents all the states of the MP4 local recording/saving. It allows you to get notified about the state of the recording. You must create a new handler and implement the Rtmp Listener interface in your activity, then pass it over in the constructor as follows:

```
mStreamaxiaPublisher.setRecordEventHandler(new RecordHandler(this));
```

The states are as follow:

- *onRecordPause*
- *onRecordResume*
- *onRecordStarted*
- *onRecordFinished*
- *onRecordIllegalArgumentException*
- *onRecordIOException*

EncoderHandler.EncoderListener

The network handler interface represents all the states of the network events that occur during the streaming process. You must create a new handler and implement the Encoder Listener interface in your activity, then pass it over in the constructor as follows:

```
mStreamaxiaPublisher.setEncoderHandler(new EncoderHandler(this));
```

The states are as follow:

- *onNetworkResume*
- *onNetworkWeak*
- *onEncoderIllegalArgumentException*

Streamaxia Streamer

Streamer provides a way to stream from another source than the phone's camera. Streamer takes the raw video frames and and raw audio samples from any source, encodes them and sends the over rtmp to the stream sources of your choice.

Same handlers are available for streamer as for publisher.

OpenSDK Project Setup

1. Open an existing project or create a new one.
2. Add the *.aar* file provided, as a module to the main project
3. Add the *license.dat* file in the */assets* folder of the main project.

Initialize the SDK Main Interface

To initialize the SDK, you need to make sure that the license file is added to the assets folder.

Create a new *StreamaxiaPublisher* object as follows:

```
mStreamaxiaPublisher = new StreamaxiaPublisher(mCameraView, this);
```

The constructor needs the following parameters:

1. The *CameraView* - a view where the frames from the camera are rendered

2. The context - the activity's context NOT the application context

In the activity XML file, you will need to create a *CameraView*. A view that will be used to render the actual frames that come from the camera.

```
<com.streamaxia.android.CameraPreview
    android:id="@+id/preview"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:adjustViewBounds="true"
    android:keepScreenOn="true"/>
```

Reference the view in your activity, to pass it over to the constructor of the main interface.

```
@BindView(R.id.preview)
CameraPreview mCameraView;
```

or

```
mCameraView = (CameraView) findViewById(R.id.preview);
```

Handlers and Listeners set up

Some of the listeners will deliver a specific message, letting you know what is happening.

```
@Override
public void onRtmpConnecting(String msg) {
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
}
```

RtmpHandler.RtmpListener

```
@Override
public void onRtmpConnecting(String msg) {
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
}

@Override
```

```java
public void onRtmpConnected(String msg) {
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
}

@Override
public void onRtmpVideoStreaming() {
    Log.i(TAG, "Video Streaming");
}

@Override
public void onRtmpAudioStreaming() {
    Log.i(TAG, "Audio Streaming");
}

@Override
public void onRtmpStopped() {
    Toast.makeText(getApplicationContext(), "Stopped", Toast.LENGTH_SHORT).show();
}

@Override
public void onRtmpDisconnected() {
    Toast.makeText(getApplicationContext(), "Disconnected", Toast.LENGTH_SHORT).show();
}

@Override
public void onRtmpVideoFpsChanged(double fps) {
    Log.i(TAG, String.format("Output Fps: %f", fps));
}

@Override
public void onRtmpVideoBitrateChanged(double bitrate) {
    int rate = (int) bitrate;
    if (rate / 1000 > 0) {
        Log.i(TAG, String.format("Video bitrate: %f kbps", bitrate / 1000));
    } else {
        Log.i(TAG, String.format("Video bitrate: %d bps", rate));
    }
}

@Override
public void onRtmpAudioBitrateChanged(double bitrate) {
    int rate = (int) bitrate;
    if (rate / 1000 > 0) {
        Log.i(TAG, String.format("Audio bitrate: %f kbps", bitrate / 1000));
    } else {
        Log.i(TAG, String.format("Audio bitrate: %d bps", rate));
    }
}

@Override
public void onRtmpSocketException(SocketException e) {
    handleException(e);
}

@Override
public void onRtmpIOException(IOException e) {
    handleException(e);
}

@Override
public void onRtmpIllegalArgumentException(IllegalArgumentException e) {
    handleException(e);
}

@Override
public void onRtmpIllegalStateException(IllegalStateException e) {
    handleException(e);
}

@Override
public void onRtmpAuthenticationg(String msg) {
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
}
```

Recording Handlers

```java
@Override
public void onRecordPause() {
    Toast.makeText(getApplicationContext(), "Record pause", Toast.LENGTH_SHORT).show();
}

@Override
public void onRecordResume() {
    Toast.makeText(getApplicationContext(), "Record resume", Toast.LENGTH_SHORT).show();
}

@Override
public void onRecordStarted(String msg) {
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
}

@Override
public void onRecordFinished(String msg) {
    Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
}

@Override
public void onRecordIllegalArgumentException(IllegalArgumentException e) {
    handleException(e);
}

@Override
public void onRecordIOException(IOException e) {
    handleException(e);
}
```

Network Handlers

```
@Override
public void onNetworkWeak() {
    Toast.makeText(getApplicationContext(), "Network weak", Toast.LENGTH_SHORT).show();
}

@Override
public void onNetworkResume() {
    Toast.makeText(getApplicationContext(), "Network resume", Toast.LENGTH_SHORT).show();
}

@Override
public void onEncodeIllegalArgumentException(IllegalArgumentException e) {
    handleException(e);
}
```

Example of Exception handling

Sometimes there may be some exception that you will need to be handled. Below you will find a way of how the demo app handles an exception.

Note: Your feedback regarding current issues and possible improvements is priceless! Help improve OpenSDK by submitting your feedback to support@streamaxia.com.

```
private void handleException(Exception e) {
    try {
        Toast.makeText(getApplicationContext(), e.getMessage(),
Toast.LENGTH_SHORT).show();
        mStreamaxiaPublisher.stopPublish();
        mStreamaxiaPublisher.stopRecord();
        btnPublish.setText("publish");
        btnRecord.setText("record");
        btnSwitchEncoder.setEnabled(true);
    } catch (Exception e1) {
        //
    }
}
```

Life Cycle

Don’t forget to handle the publisher functions also on the activities different lifecycle events.

```
@Override
protected void onResume() {
    super.onResume();
    if (mStreamaxiaPublisher != null) {
        mCameraView.startCamera();
    }
}

@Override
protected void onPause() {
    super.onPause();
    if (mStreamaxiaPublisher != null) {
        mCameraView.stopCamera();
        mStreamaxiaPublisher.pauseRecord();
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    if (mStreamaxiaPublisher != null) {
        mStreamaxiaPublisher.stopPublish();
        mStreamaxiaPublisher.stopRecord();
    }
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    mStreamaxiaPublisher.setScreenOrientation(newConfig);
}
```

Using OpenSDK Features

Camera View

Make sure you call the start method on the camera view **before** streaming. The start method will enable the frames from the camera to get rendered to the user and will send the actual data to the encoder part.

If the start method is not called, the encoder will have no frames to publish.

```
mCameraView.start();
```

Start and Stop Publishing

To start publishing a video, call the *startPublish* method on the StreamaxiaPublisher instance. The method requires the URL of the endpoint.

```
mStreamaxiaPublisher.startPublish(rtmpUrl);
```

To stop publishing just call the *stopPublish* method on the StreamaxiaPublisher instance.

```
mStreamaxiaPublisher.stopPublish();
```

Start, Stop, Pause and Resume Recording

To start recording (saving locally) the video, just call *startRecord* on the StreamaxiaPublisher instance.

```
mStreamaxiaPublisher.startRecord(recPath);
```

The method requires as a parameter the patch where to save the video. For example:

```
private static final String recPath =
                        Environment.getExternalStorageDirectory().getPath() + "/test.mp4";
```

To stop the recording just call *stopRecord* on the object.

```
mStreamaxiaPublisher.stopRecord();
```

To pause the recording just call *pauseRecord* on the object.

```
mStreamaxiaPublisher.pauseRecord();
```

The difference between pause and stop is that when you call *stopRecord*, the video will be ended and saved locally on the given path. If you paused a video, you can resume (continue) the video with the *resumeRecord* method.

```
mStreamaxiaPublisher.resumeRecord();
```

Handling Configuration Changes (API 18+)

The SDK also supports orientation changes before the streaming is started. In order for this to be possible, you need to declare inside your Activity field in the AndroidManifest.xml file the following:

```
android:configChanges="orientation|screenSize"
```

Then in the Activity override the on configuration changed method as follows:

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    //Update the orientation for stream
    mStreamaxiaPublisher.setScreenOrientation(newConfig.orientation);
    //The new orientation might not support the current resolution
    Size supportedRes = mStreamaxiaPublisher.getSupportedPictureSizes(newConfig.orienta-
tion).get(0);
    mStreamaxiaPublisher.setVideoOutputResolution(supportedRes.width, supported-
Res.height, newConfig.orientation);
}
```

By doing so, the user will be able to choose the orientation of the stream(portrait or landscape) by rotating the device, before the streaming is started

While streaming, the Activity will be locked on the current position and after finishing the streaming, it will be unlocked and the orientation can be changed again.

This feature can be disabled and you can choose only one orientation, as follows:

```
mStreamaxiaPublisher.setScreenOrientation(Configuration.ORIENTATION_LANDSCAPE);
```

or

```
mStreamaxiaPublisher.setScreenOrientation(Configuration.ORIENTATION_PORTRAIT);
```

By doing so, you will need to lock also the activity's orientation from the AndroidManifest.xml file as follows:

```
android:screenOrientation="landscape"
```

or

```
android:screenOrientation="portrait"
```

You can also take screenshots at any point using the following code:

```
mCameraView.takeSnapshot(new CameraPreview.SnapshotCallback() {
     @Override
     public void onSnapshotTaken(Bitmap image) {
         //Do something with the snapshot
     }
 });
```

Realtime zooming in/out is also supported, you can use the following API:

```
// Get zoom range
int[] range = mCameraView.getZoomRange();
// Set max zoom
mCameraView.setZoom(range[1]);
```

Framerate and keyframe interval can be set BEFORE starting the stream using:

```
// Set the framerate to 30fps, we can also use the getter to check the value
mPublisher.setFramerate(30);
// Set the keyframe interval to 5 seconds, we can also use the getter to check the value
mPublisher.setKeyframeInterval(5);
```

NOTE : we do not support reverse landscape nor reverse portrait!

Streamer

You can stream from any stream source (from a file for instance) using Streamaxia streamer API. Basic configuration for streamer is:

```
StreamaxiaStreamer streamaxiaStreamer = new StreamaxiaStreamer(this);

streamaxiaStreamer.setEncoderHandler(new EncoderHandler(this));

streamaxiaStreamer.setRtmpHandler(new RtmpHandler(this));

streamaxiaStreamer.setRecordEventHandler(new RecordHandler(this));

streamaxiaStreamer.setColorFormat(MediaCodecInfo.CodecCapabilities.COLOR_FormatYUV420Planar);

streamaxiaStreamer.setVideoOutputResolution(1920, 1080);
```

You can start/stop the streamer using:

```
// Set the framerate to 30fps, we can also use the getter to check the value
```

```
String rtmpUrl = "rtmp://rtmp.streamaxia.com/streamaxia/" + streamaxiaStreamName;

streamaxiaStreamer.startPublish(rtmpUrl);

streamaxiaStreamer.stopPublish();
```

You can also record locally while streaming from your audio and video source:

```
String outputpath = Environment.getExternalStorageDirectory().getPath() + "/test.mp4";

streamaxiaStreamer.startRecord(outputpath);

streamaxiaStreamer.stopRecord();
```

To feed your raw audio and video data to the streamer the following API is available:

```
// Populate buffer with raw data and send it to the streamer

byte[] buffer;

streamaxiaStreamer.sendVideoFrames(buffer);

streamaxiaStreamer.sendAudioFrames(buffer);
```

Resources

OpenPlayer for Android

Thousands of personal users use OpenPlayer daily to play live or on demand content using Mobile Apps powered by Streamaxia.

Not sure yet? See the working demo of the library here or contact us here.

OpenSDK for Broadcasting

For iOS and Android Mobile App Developers

Easy to integrate, low-latency live video streaming library.
Open broadcast – not limited to any specific CDN, RTMP Media Server or proprietary protocols.

Drag, Drop & Go Live!

[OpenSDK 3.0 for iOS]()

[OpenSDK 3.0 for Android]()

BroadcastMe App

Live Video Streaming Broadcast Apps

Private Label for iOS and Android Available

BroadcastMe Developer Edition is designed by Streamaxia to be used by Mobile App Developers and Digital Media experts as is, and it is available for private label for your brand.

Read More

Help and Support

Our team looks forward to helping you with business inquiries and technical support questions. We will review all messages within two business days and respond back to you promptly about your request for information.

support@streamaxia.com

Mo-Fri 9AM-5PM EET