AudioFetch Android SDK API v2

Overview

The AudioFetch Android SDK allows AudioFetch partner companies to add real-time, low latency streaming audio capability to their own Android App. The SDK takes the form of an Android library that may be linked against. An example application showing how to use the SDK is also provided.

The SDK starts a service within a host application, and provides an API to send messages to and from that service to control what the AudioFetch Service is doing, and to be notified of AudioFetch Service state changes and events.

The API is implemented using RxJava to implement two message streams: incoming and outgoing. The incoming message api sends messages to the AudioFetch service such as: StartAudioMsg, StartDiscoveryMsg, etc. The outgoing message api sends messages to the host application such as: ChannelsReceivedMsg.

The use of RxJava to implement the API allows decoupling of the host application architecture from the AudioFetch SDK. Additionally RxJava allows easy use of

multi-threading if desired and other potential publish/subscribe architecture advantages, should the host application desire to take advantage of them.

For commands to the AudioFetch service, convenience wrapper functions are provided so that you may easily send messages to the AudioFetch service:

```
AFAudioService.api().startAudio()
AFAudioService.api().stopAudio()

AFAudioService.api().setChannel(1)

AFAudioService.api().startDiscovery()
AFAudioService.api().stopDiscovery()
```

The AudioFetch service will emit messages such as:

```
ChannelsReceivedMsg
AudioStateMsg
```

These messages are published on an outgoing message bus, implemented in RxJava. Initially when you start the AudioFetch service, you subscribe to these messages. The SDK sample app contains this code to start the service:

```
AFAudioService.api().outMsgs()
    .asFlowable()
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe(
        msg -> {
          if (msg instanceof AfApi.ChannelsReceivedMsg) {
            AfApi.ChannelsReceivedMsg  crMsg = (AfApi.ChannelsReceived-
Msg) msg;
            onChannelsReceivedEvent(crMsg);
          }
          else if (msg instanceof AfApi.WifiStatusMsg) {
            AfApi.WifiStatusMsg  pMsg = (AfApi.WifiStatusMsg) msg;
            onWifiStatusEvent(pMsg);
          }
          else if (msg instanceof AfApi.AudioFocusMsg) {
            AfApi.AudioFocusMsg  pMsg = (AfApi.AudioFocusMsg) msg;
            onAudioFocusEvent(pMsg);
          }
          else if (msg instanceof AfApi.ApplicationFinishMsg) {
            ApplicationBase ab = ApplicationBase.getInstance();
```

```
            if (null != ab) {
                ab.finish();
            }
        }

    });
```

The AudioFetch service does not make any assumptions about your application structure, but provides a foreground service to your application that processes the realtime audio, and also discovers AudioFetch boxes. The use of RxJava in the API means you have flexibility on how you handle in and out commands in any multi-threading manner you wish.

Note that volume is not handled in the AudioFetch Service, but is the responsibility of the application to manage, if desired. The sample application contains a volume slider that enables user control of volume.

Discovery Process

Discovery of AudioFetch boxes is initiated on startup when initAudioSubsystem() is called. This discovery typically lasts for up to 10 seconds, and automatically stops when complete and results in a ChannelsReceivedMsg being sent from the serivce to the app with appropriate channel information about any AudioFetch boxes that were discovered.

After this time period, a call to startDiscovery() will re-initiate this process. During the discovery process, a call to stopDiscovery() will abort the discovery process.

Steps To Integrate The AudioFetch SDK Into Your App

1. Add Packages and AudioFetch library to build.gradle

```
dependencies {

    ... your dependencies

    // Audiofetch SDK Dependancies
    implementation 'io.reactivex.rxjava2:rxjava:2.2.0'
    implementation 'io.reactivex.rxjava2:rxandroid:2.0.1'
    implementation 'com.jakewharton.rxrelay2:rxrelay:2.0.0'

    // Audiofetch SDK library
    implementation(name: 'afaudiolib', ext: 'aar')

}
```

2. Add AudioFetch Service to ApplicationManifest.xml

```
    <service android:name="com.audiofetch.afaudiolib.bll.app.AFAudioService"
        android:stopWithTask="true"
        android:singleUser="true"
        android:exported="false"
        android:label="AudioFetch Music">
        <intent-filter>
            <action android:name="com.audiofetch.afaudiolib.bll.app.AFAudio-
Service.NEXT"/>
            <action android:name="com.audiofetch.afaudiolib.bll.app.AFAudio-
Service.PAUSE"/>
            <action android:name="com.audiofetch.afaudiolib.bll.app.AFAudio-
Service.PLAY"/>
            <action android:name="com.audiofetch.afaudiolib.bll.app.AFAudio-
Service.PREV"/>
            <action android:name="com.audiofetch.afaudiolib.bll.app.AFAudio-
Service.STOP"/>
            <action android:name="com.audiofetch.afaudiolib.bll.app.AFAudio-
Service.CLOSE"/>
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </service>
```

3. Add Permissions to ApplicationManifest.xml

```
    <permission android:name="android.permission.MEDIA_CONTENT_CONTROL" />

    <uses-permission android:name="android.permission.KILL_BACKGROUND_PRO-
CESSES" />
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.REQUEST_IGNORE_BAT-
TERY_OPTIMIZATIONS"/>
    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAS-
T_STATE" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" /
>
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission
android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/
>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" /
>
    <uses-permission android:name="com.samsung.android.sdk.professionalau-
dio.permission.START_MONITOR_SERVICE"/>
    <uses-permission android:name="com.samsung.android.providers.context.per-
mission.WRITE_USE_APP_FEATURE_SURVEY" />
```

4. Start the AudioFetch Service

```
    protected ActivityBase startAFAudioService() {
        if (null == mAFAudioSvc) {
            final Intent serviceIntent = new Intent(this, AFAudioService.-
class);
            startService(serviceIntent);
            bindService(new Intent(this, AFAudioService.class), getAFAudio-
ServiceConnection(), 0);
        }
        return this;
    }

    /**
     * Starts the Audiofetch Service and returns a reference to it.
     *
```

```
     * @see AFAudioService
     * @return
     */
    protected ServiceConnection getAFAudioServiceConnection() {
        if (null == mAFAudioSvcConn) {
            mAFAudioSvcConn = new ServiceConnection() {
                @Override
                public void onServiceConnected(ComponentName className,
IBinder service) {
                    if (service instanceof AFAudioService.AFAudioBinder) {
                        LG.Debug(TAG, "AFAudioService connected");
                        AFAudioService.AFAudioBinder binder = (AFAudioSer-
vice.AFAudioBinder) service;
                        mAFAudioSvc = binder.getService();

                        if (null != mAFAudioSvc) {
                            Context ctx = getApplicationContext();
                            // app context must be set before initing audio
subsystem
                            AFAudioService.api().setAppContext( getApplica-
tionContext() );
                            AFAudioService.api().initAudioSubsystem();

                            mIsAFAudioSvcBound = true;
                            mAFAudioSvc.hideNotifcations();

                            mUiHandler.post(new Runnable() {
                                @Override
                                public void run() {
                                    startAFAudioServiceAudio();
                                }
                            });

                            LG.Debug(TAG, "AudioFetch Service In and out API
connected.");
                            doSubscriptions();
                        }
                    }
                }

                @Override
                public void onServiceDisconnected(ComponentName component-
Name) {
                    LG.Debug(TAG, "AFAudioService disconnected");
                    mIsAFAudioSvcBound = false;
                    mAFAudioSvcConn = null;
                    mAFAudioSvc = null;
                }
            };
        }
        return mAFAudioSvcConn;
    }
```

5. Subscribe to outgoing API messages from the AudioFetch Service

```
        AFAudioService.api().outMsgs()
            .asFlowable()
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(
                msg -> {
                  if (msg instanceof AfApi.ChannelsReceivedMsg) {
                    AfApi.ChannelsReceivedMsg  crMsg = (AfApi.ChannelsReceived-
    Msg) msg;
                    onChannelsReceivedEvent(crMsg);
                  }
                  else if (msg instanceof AfApi.WifiStatusMsg) {
                    AfApi.WifiStatusMsg  pMsg = (AfApi.WifiStatusMsg) msg;
                    onWifiStatusEvent(pMsg);
                  }
                  else if (msg instanceof AfApi.AudioFocusMsg) {
                    AfApi.AudioFocusMsg  pMsg = (AfApi.AudioFocusMsg) msg;
                    onAudioFocusEvent(pMsg);
                  }
                  else if (msg instanceof AfApi.ApplicationFinishMsg) {
                    ApplicationBase ab = ApplicationBase.getInstance();
                    if (null != ab) {
                        ab.finish();
                    }
                  }

                });
```

6. Send commands to the AudioFetch Service

```
        AFAudioService.api().startAudio()
        AFAudioService.api().stopAudio()

        AFAudioService.api().setChannel(1)

        AFAudioService.api().startDiscovery()
        AFAudioService.api().stopDiscovery()
```

AudioFetch API Details

Incoming Messages and Convenience Wrapper Functions:

Message	Convenience Wrapper	Notes
`InitAudioSubsystemMsg`	`initAudioSubsystem()`	Init audio service on application create.
`DestroyAudioSubsystemMsg`	`destroyAudioSubsystem()`	Destroy audio service on application destroy.
`SetChannelMsg`	`setChannel()`	Set the channel to a channel number.
`StartAudioMsg`	`startAudio()`	Start audio playback.
`StopAudioMsg`	`stopAudio()`	Stop audio playback.
`StartDiscoveryMsg`	`startDiscovery()`	Start or restart discovery of AudioFetch boxes.
`StopDiscoveryMsg`	`stopDiscovery()`	Stop any ongoing discovery of AudioFetch boxes.

Outgoing Messages:

Message	Notes
`ChannelsReceivedMsg`	Sent after AudioFetch boxes are discovered, contains a list of available channels.
`AudioStateMsg`	Sent when audio state changes. For details, see sample application.

Message	Notes
`WifiStatusMsg`	Sent when WiFi status changes, eg when WiFi connectivity is lost or gained.
`HeadsetMsg`	Sent when the headset is plugged in or unplugged.
`AudioFocusMsg`	Sent when the Android system audio focus changes.