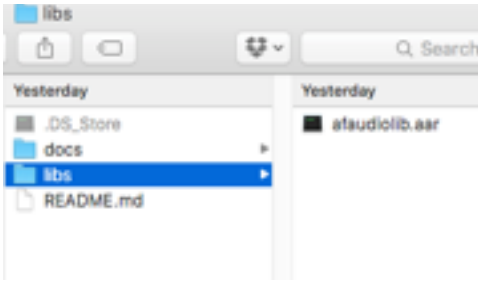
AudioFetch SDK for Android

The AudioFetch SDK allows developers to add the AudioFetch system channel discovery and audio management features directly to their third-party native Android applications.

Documentation Date: July 27, 2016
Documentation Version: 1.0

The following is a break down of how to add and integrate the AudioFetch SDK to your Android app.

1. Download, unzip, and add the SDK package to your project
   1.1. Download the zip archive here: https://github.com/audiofetch/audiofetch-android-sdk-public-library/archive/master.zip
   1.2. Extract the zip file, and copy the libs/afaudiolib.aar Android library package to your project's libs folder.



1.3. Add the Maven Central maven repositories, and the flatDir directive to the repositories section of your project's build.gradle file.

1.4. Add the dependency for the AudioFetch Android SDK library, as well as the dependencies required to use the library. (See README.md to copy/paste code)

```
repositories {
    mavenCentral()
    flatDir {
        dirs 'libs'
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])

    // audiofetch android SDK lib
    compile(name: 'afaudiolib', ext: 'aar')

    // required by above AudioFetch SDK lib
    compile 'com.android.support:support-annotations:23.+'
    compile 'com.squareup:otto:1.3.8'
    compile 'com.google.guava:guava:19.0'
}
```

1.5. Initialize and create your AudioController reference. This will kick off discovery of AudioFetch Devices as well as start the audio stream. The ChannelsReceivedEvent will occur once discovery has finished. Additionally, AudioStateEvents will occur repeatedly, throughout the app lifecycle.

```
protected AudioController mAudioController;
```

```
AudioController.init(this);
mAudioController = AudioController.get();
```

1.6. Create your Otto Bus Events (http://square.github.io/otto/) that are required. See the JavaDoc (https://github.com/audiofetch/audiofetch-android-sdk-public-library/tree/master/docs) for more information about these events and when they're triggered.

1.6.1. Import the Bus library's annotation:

```
import com.squareup.otto.Subscribe;
```

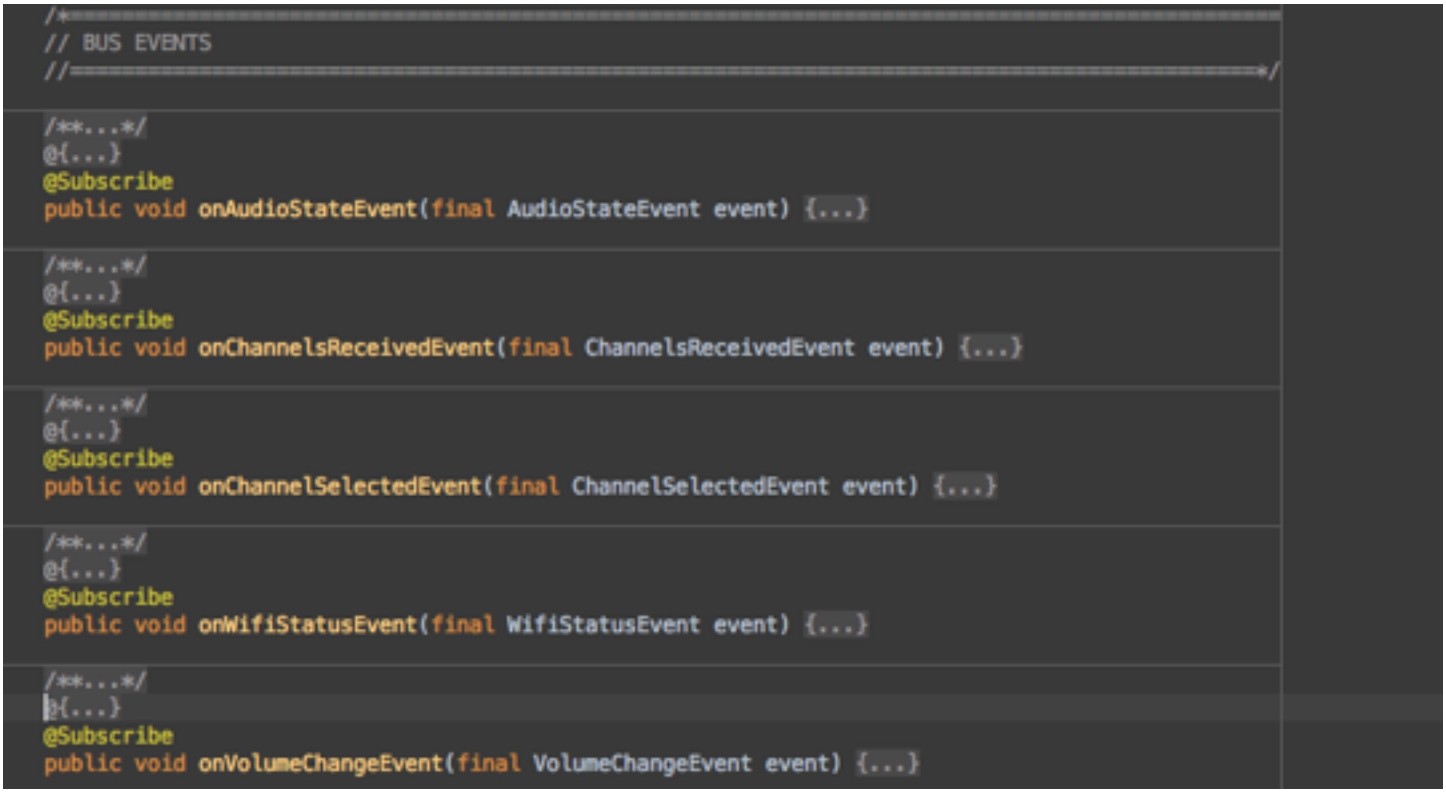
1.6.2. Register/Unregister to receive Bus events in the appropriate lifecycle methods.

```
@Override
public void onStart() {
    super.onStart();
    final int loadCount = mLoadCount.incrementAndGet();
    final boolean isFirstLoad = (1 == loadCount);

    if (!mIsBusRegistered) {
        try {
            MainActivity.getBus().register(this);
```

```
@Override
public void onStop() {
    if (mIsBusRegistered) {
        // remove any possible pending callbacks
        MainActivity.getBus().unregister(this);
        mIsBusRegistered = false;
    }
    super.onStop();
}
```

1.6.3. Create event handlers to receive the events.
  1.6.3.1. The minimum events that should be handled are:
    1.6.3.1.1. AudioStateEvent
    1.6.3.1.2. ChannelsReceivedEvent
    1.6.3.1.3. WifiStatusEvent
  1.6.3.2. Other optional events that may be needed include:
    1.6.3.2.1. VolumeChangeEvent
    1.6.3.2.2. ChannelSelectedEvent



```
/*=============================================================================
// BUS EVENTS
//=============================================================================*/

/**...*/
@{...}
@Subscribe
public void onAudioStateEvent(final AudioStateEvent event) {...}

/**...*/
@{...}
@Subscribe
public void onChannelsReceivedEvent(final ChannelsReceivedEvent event) {...}

/**...*/
@{...}
@Subscribe
public void onChannelSelectedEvent(final ChannelSelectedEvent event) {...}

/**...*/
@{...}
@Subscribe
public void onWifiStatusEvent(final WifiStatusEvent event) {...}

/**...*/
@{...}
@Subscribe
public void onVolumeChangeEvent(final VolumeChangeEvent event) {...}
```

* See SDK Sample Application for method details, and an example of how to populate your UI with channel information.

1.6.4. Implement other optional features:

1.6.4.1. Pass onKeyDown event from Activity through to the AudioController, to handle hardware button volume changes.

```
/**
 * Should be called by calling activity to set volume by passing event from the activity's onKeyDown event handler
 *
 * @param keyCode
 * @param event
 * @return
 */
public boolean onKeyDown(final int keyCode, final KeyEvent event) {
    boolean success = false;
    AtomicInteger currentVolume = new AtomicInteger(-1);
    if (null != getMainActivity() && null != mAudioController) {
        if (mAudioController.onKeyDown(keyCode, event, currentVolume) && currentVolume.intValue() >= 0) {
            success = true;
        }
    }
    return success;
}
```

1.6.4.2. Set a SeekBar as a volume control to allow in-app volume changes.

```
mVolumeControl = (SeekBar) mView.findViewById(R.id.volume_slider);
mAudioController.setVolumeControl(mVolumeControl);
```

See the sample SDK application (https://github.com/audiofetch/audiofetch-android-sdk-sample), for a full reference architecture of how to implement the AudioFetch Android SDK.

Sample SDK Application may be downloaded here:
https://github.com/audiofetch/audiofetch-android-sdk-sample/archive/master.zip