Affdex SDK for Android

Developer Guide
For SDK version 2.01

Introduction

The Affdex SDK is the culmination of years of scientific research into emotion detection, validated across thousands of tests worldwide on PC platforms, and now made available on Android and Apple iOS. Affdex SDK turns your ordinary app into an extraordinary app by emotion-enabling your app to respond in real-time to user emotions.

In this document, you will become familiar with integrating the Affdex SDK into your Android app. Please take time to read this document and feel free to give us feedback at sdk@affectiva.com.

What's in the SDK

The Affdex SDK package consists of the following:

- **SDK Developer Guide.pdf** (this document)
- **docs**, the folder containing documentation. In the javadoc subfolder, start with index.html.
- **libs**, the folder containing Affdex SDK libraries that your app will link against
- **assets**, the folder containing files needed by the SDK

Requirements

The Affdex SDK requires a device running Android API 16 or above.

Java 1.6 or above is required on your development machine.

Licensing

After you request the SDK, Affectiva will provide to you an Affectiva license file. Copy this file into your Android app project under the folder /assets/Affdex, and specify its relative path under that folder when invoking the setLicensePath method (described in more detail below).

Outline

This document will guide you through the following:

- Adding the SDK to your Android project
- Using the SDK
- Options

- Interpreting the data
- Where to Go From Here

Add the SDK to your Project

In order to use this SDK in one of your Android apps, you will need to copy some files from the SDK into your Android project. In your Android project, alongside your “src and “res” folders, you may have the optional folders “assets” and “libs”. Copy the SDK’s “assets” folder into your project. If you already have an “assets” folder, copy the contents of the SDK’s “assets” folder into your “assets” folder. In a similar way, copy the SDK's “libs” folder into your project.

The provided sample app does not have “assets” or “libs” folders. In this case, simply copy the entire “assets and “libs” folders into the app’s project, at the same level as the “res” and “src” folders.

We do not recommend adding any of your own files to the “assets/Affdex” folder.

Using the SDK

The following code snippets demonstrate how easy it is to obtain facial expression results using your device’s camera, a video file, or from images.

SDK Operating Modes

The Affdex SDK has the following operating modes:

- Camera mode: the SDK turns connects to the camera and processes the frames it records. Sample app: AffdexMe.
- Video file mode: provide to the SDK a path to a video file.
- Pushed frame mode: provide to the SDK individual frames of video and their timestamps.
- Photo mode: provide discrete images to the SDK (unrelated to any other image).

The SDK provides mode-specific Detector classes for each of these modes: CameraDetector, VideoFileDetector, FrameDetector, and PhotoDetector.

SDK calling overview

In general, calls to the SDK are made in the following order:

- Construct a Detector corresponding to the operating mode that you want. The following methods are called on a Detector instance.
- Call `setLicensePath()` with the path to the license file provided by Affectiva.
- Set options for the Detector. In particular, enable detection of at least one expression or emotion metric (e.g. call setDetectSmile() to detect smiles). See the "Options" section below for more information on the different options available.
- Note that there are no methods to enable measurement metrics such as Yaw, Pitch, and Roll, because these are enabled automatically.
- Call `start()` to start processing. Note the types of exceptions that can be thrown and handle them as desired.
- If you are pushing your own images (pushed frame mode or photo mode), call `process()` with each image.
- When you are done processing, call `stop()`.

To receive results from the SDK, implement the `Detector.ImageListener` and/or `Detector.FaceListener` interfaces, and register your listener object(s) with the Detector via `setImageListener()` and/or `setFaceListener()`. These interfaces provide results of the SDK's processing of each frame. The `ImageListener` interface provides information about facial expressions and face points for a face found in a given image via its `onImageResults` callback. The `FaceListener` interface notifies its listener when a face appears or disappears via its `onFaceDetectionStarted()` and `onFaceDetectionStopped()` callbacks. For an example of using these callbacks to show and hide the results from the SDK, see the sample app AffdexMe.

To check to see if the Detector is running (`start()` has been called, but not `stop()`), call `isRunning()`.

Note: Be sure to always call `stop()` following a successful call to `start()` (including for example, in circumstances where you abort processing, such as in exception catch blocks). This ensures that resources held by the Detector instance are released.

Camera Mode

Using the built-in camera is a common way to obtain video for facial expression detection. Either the front or back camera of your Android device can be used to feed video directly to the Detector.

A demonstration of Camera Mode is the sample app AffdexMe.

To use Camera Mode, implement the `Detector.ImageListener` and/or `Detector.FaceListener` interface. Then follow this sequence of SDK calls:

- Construct a CameraDetector. The `cameraType` argument specifies whether to connect to the front or back camera, while the `cameraPreviewView` argument optionally specifics a SurfaceView onto which the SDK should display preview frames.

  ```
  public CameraDetector(Context context,
                        SurfaceView cameraPreviewView)
  ```

- Call `setLicensePath()` with the path to the license file provided by Affectiva.
- Set options for the Detector. In particular, turn on at least one facial expression metric to detect facial expressions, e.g.

  ```
  setDetectSmile(true);
  ```

- Call `start()` to start initialize the SDK and call `startCamera(CameraType)` to start the device Camera. If successful, you will start receiving calls to `onImageResults()`.
- When you are done, call `stopCamera()` to stop the Camera and `stop()` to release the resources used by the Affdex SDK.
- Remember to add the Camera permission to your AndroidManifest.xml file.

Sizing the SurfaceView

Aside from the convenience of managing the Android Camera, CameraDetector also takes care of choosing the frame rate and frame size that will work best with the SDK. Since it is the developer's responsibility to layout and size the SurfaceView passed into CameraDetector, you may want to resize the SurfaceView to match the aspect ratio of the returned frames. For this purpose, implement the CameraDetector.OnCameraEventListener interface to receive the onCameraSizeSelected event. Below is a block of sample code showing how to resize the SurfaceView to occupy as much space as its parent container while matching the aspect ratio of the incoming camera frames.

```
@Override
public void onCameraSizeSelected(int cameraWidth, int cameraHeight, ROTATE rotation) {
        int cameraPreviewWidth;
        int cameraPreviewHeight;

        //cameraWidth and cameraHeight report the unrotated dimensions of the camera
frames, so switch the width and height if necessary
        if (rotation == ROTATE.BY_90_CCW || rotation == ROTATE.BY_90_CW) {
                cameraPreviewWidth = cameraHeight;
                cameraPreviewHeight = cameraWidth;
        } else {
                cameraPreviewWidth = cameraWidth;
                cameraPreviewHeight = cameraHeight;
        }

        //retrieve the width and height of the ViewGroup object containing our
SurfaceView (in an actual application, we would want to consider the possibility that
the mainLayout object may not have been sized yet)
        int layoutWidth = mainLayout.getWidth();
        int layoutHeight = mainLayout.getHeight();

        //compute the aspect Ratio of the ViewGroup object and the cameraPreview
        float layoutAspectRatio = (float)layoutWidth/layoutHeight;
        float cameraPreviewAspectRatio = (float)cameraWidth/cameraHeight;

        int newWidth;
        int newHeight;

        if (cameraPreviewAspectRatio > layoutAspectRatio) {
                newWidth = layoutWidth;
                newHeight =(int) (layoutWidth / cameraPreviewAspectRatio);
        } else {
                newWidth = (int) (layoutHeight * cameraPreviewAspectRatio);
                newHeight = layoutHeight;
        }

        //size the SurfaceView
        ViewGroup.LayoutParams params = surfaceView.getLayoutParams();
        params.height = newHeight;
        params.width = newWidth;
        surfaceView.setLayoutParams(params);
}
```

Hiding the SurfaceView

Some applications may not wish to display the camera preview on screen. Since Android requires an active Surface for the camera to function, CameraDetector always requires a SurfaceView to be passed in. However, if you do not wish to display the preview, you can set the SurfaceView to be 1px by 1px and call SurfaceView.setAlpha(0) to hide it on-screen.

Video File Mode

Another way to feed video into the detector is via a video file that is stored on the file system of your device. Follow this sequence of SDK calls:

- Construct a VideoFileDetector. The `filePath` argument is the path to your video file.

```
public Detector(Context context, String filePath)
```

- Call `setLicensePath()` with the path to the license file provided by Affectiva.
- Set options for the Detector. In particular, turn on at least one facial expression metric to detect facial expressions, e.g.

  ```
  setDetectSmile(true);
  ```

- Call `start()` to start processing. You will start receiving calls to `onImageResults()`.
- When you are done processing, call `stop()`.

Pushed Frame Mode

If your app is processing video and has access to video frames, you can push those video frames to the Affdex SDK for processing. Each video frame has an associated timestamp that increases with each frame in the video. Your app may have access to video frames because your app is interfacing to the device's camera, or because your app is reading a video file, or perhaps by some other method.

- Construct a FrameDetector.

  ```
  public FrameDetector(Context context)
  ```

- Call `setLicensePath()` with the path to the license file provided by Affectiva.
- Set options for the Detector. In particular, turn on at least one facial expression metric to detect facial expressions, e.g.

  ```
  setDetectSmile(true);
  ```

- Call `start()` to start processing.
- For each video frame, create an Affdex Frame (Bitmap, RGBA, and YUV420sp/NV21 formats are supported). Note: Frame is an abstract base class with two concrete subclasses: BitmapFrame and ByteArrayFrame; you should construct one of these concrete subclasses.
- Call `process` with the Affdex Frame and timestamp of the frame:

  ```
  public abstract void process(Frame frame, float timestamp);
  ```

- For each call to `process`, the SDK will call `onImageResults()`.

- When you are done processing, call `stop()`.

Photo Mode

Use Photo Mode for processing images that are unrelated to each other (that is, they are not sequential frames of a video). Discrete images are processed by the SDK independently, without regard to the content of the preceding images, using different algorithms and data than are used with the other modes involving sequences of frames from a video source.

- Construct a PhotoDetector.

  ```
  public Detector(Context context)
  ```

- Call `setLicensePath()` with the path to the license file provided by Affectiva.
- Set options for the Detector. In particular, turn on at least one facial expression metric to detect facial expressions, e.g.

  ```
  setDetectSmile(true);
  ```

- Call `start()` to initialize the PhotoDetector.
- For each photo to be processed, create an Affdex Frame from your frame (Bitmap, RGBA, and YUV420sp/NV21 formats are supported). Note: Frame is an abstract base class with two concrete subclasses: BitmapFrame and ByteArrayFrame; you should construct one of these concrete subclasses.
- Call `process` with the Affdex Frame:

  ```
  public abstract void process(Frame frame);
  ```

- For each call to `process`, the SDK will call `onImageResults()`.
- When you are done processing, call `stop()`.

Options

This section describes various options for operating the SDK.

Detecting emotions and expressions

The Affdex SDK can detect a variety of facial emotions and expressions, yielding metric scores for the metrics you configure. By default, no emotions or expressions are detected. Detection

can be enabled or disabled via setDetectXXX methods defined on the Detector class. For example:

```
setDetectSmile(true)
```

See the Detector class Javadoc for a complete list of the methods available.

Processing Rate

In Camera Mode, you can specify the maximum number of frames per second that the SDK should process. This can improve performance if your requirements do not require every frame in the video stream from the camera to be processed. The default (and recommended) rate is 5 frames per second, but you may also set it lower if you are using a slower device, and need additional performance. Here is an example of setting the processing rate to 20 FPS:

```
setMaxProcessRate(20);
```

Face Detection Statistics

To get the percentage of time a face was detected during a run (between `start()` and `stop()`), call:

```
getPercentFaceDetected();
```

This can only be called after `stop()`.

Interpreting the Data

To receive the results of the SDK's processing of a frame, implement the `Detector.ImageListener` and/or `Detector.FaceListener` interfaces.

For the ImageListener interface, implement the callback `onImageResults()`, which is called by the SDK for every frame (except those that the CameraDetector skips in order to honor the maximum processing rate, unless `setSendUnprocessFrames(true)` has been called).

This method receives these parameters:

1. A list of Face objects. In this release, this will be an empty list if no face was found in the frame, or a list of one Face object if there was a face found in the frame. In Camera Mode, if `setSendUnprocessedFrames(true)` has been called, then this

parameter will be null for any frame that has been skipped in order to honor the maximum processing rate.

2. The image processed, as an Affdex Frame (a wrapper type for images, including Bitmaps, for example).
3. The timestamp of the frame. In Photo Mode, this will be zero.

The returned Face object provides getter methods for retrieving emotion, expression, and measurement scores. Scores are generally values from 0-100, representing the expression as a percent, with the exception of Valence and the measurement metrics, which can be positive or negative.

The follow code sample shows an example of how to retrieve metric values from the Face object in onImageResults:

```
@Override
public void onImageResults(List<Face> faces, Frame frame,float timestamp) {

    if (faces == null)
        return; //frame was not processed

    if (faces.size() == 0)
        return; //no face found

    Face face = faces.get(0); //Currently, the SDK only detects one face at a time

    //Some Emotions
    float joy = face.emotions.getJoy();
    float anger = face.emotions.getAnger();
    float disgust = face.emotions.getDisgust();

    //Some Expressions
    float smile = face.expressions.getSmile();
    float brow_furrow = face.expressions.getBrowFurrow();
    float brow_raise = face.expressions.getBrowRaise();

    //Measurements
    float interocular_distance = face.measurements.getInterocularDistance();
    float yaw = face.measurements.orientation.getYaw();
    float roll = face.measurements.orientation.getRoll();
    float pitch = face.measurements.orientation.getPitch();
}
```

The Face object also provides a getFacePoints() method, which returns an array of face point coordinates.

In the following code sample, the face point coordinates are retrieved from the face object and logged.

```
@Override
public void onImageResults(List<Face> faces, Frame frame,float timestamp) {

    if (faces == null)
        return; //frame was not processed

    if (faces.size() == 0)
        return; //no face found

    Face face = faces.get(0); //Currently, the SDK only detects one face at a time

    PointF[] points = face.getFacePoints();

    for (int n = 0; n < points.length; n++) {
        Log.i(LOG_TAG, String.format("Point %d is located at %.2f,%.2f
\n",n,points[n].x,points[n].y));
    }
}
```

Face Point indices

The indices of the elements in the face points array correspond to specific locations on a face. Please see the table below for an explanation of the locations corresponding to each index.

Index	Point on face	Index	Point on face
0	Right Top Jaw	17	Inner Right Eye
1	Right Jaw Angle	18	Inner Left Eye
2	Gnathion	19	Outer Left Eye
3	Left Jaw Angle	20	Right Lip Corner
4	Left Top Jaw	21	Right Apex Upper Lip
5	Outer Right Brow Corner	22	Upper Lip Center
6	Right Brow Center	23	Left Apex Upper Lip
7	Inner Right Brow Corner	24	Left Lip Corner
8	Inner Left Brow Corner	25	Left Edge Lower Lip
9	Left Brow Center	26	Lower Lip Center
10	Outer Left Brow Corner	27	Right Edge Lower Lip
11	Nose Root	28	Bottom Upper Lip
12	Nose Tip	29	Top Lower Lip
13	Nose Lower Right Boundary	30	Upper Corner Right Eye
14	Nose Bottom Boundary	31	Lower Corner Right Eye
15	Nose Lower Left Boundary	32	Upper Corner Left Eye
16	Outer Right Eye	33	Lower Corner Left Eye

Reference documentation

The SDK comes with detailed Javadoc in both JAR and HTML formats, describing all of the SDK's classes and methods.

Viewing the Javadoc in a browser:

Open the file `docs/javadoc/index.html` in the location where you installed the SDK.

Viewing the Javadoc in your IDE:

Eclipse:

In your project's libs folder, create a file called Affdex-sdk-1.1.jar.properties. Edit that file in a text editor and enter a line like "doc=path/to/the/html/javadoc". The path specified should point to the docs/javadoc folder in your SDK installation folder, and can be an absolute or relative path. On Windows, use double backslashes to separate the folder names.

Android Studio:

At the time of this writing, Android Studio does not yet support attaching javadoc to library dependencies.

Getting started with the AffdexMe sample app

The SDK comes with a sample application called AffdexMe which demonstrates how to integrate the SDK into an app. In this section, we'll walk through the steps to build this app.

Step 1: Download the AffdexMe sample app source code

Download the public repository at https://github.com/Affectiva/affdexme-android If you do not use Git, you can simply click the 'Download ZIP' button to download the repository as a .zip file

Step 2: Copy assets and libraries packaged with the SDK into the AffdexMe project

-Copy the jars in the "libs" folder of the SDK into the "libs" folder of the AffdexMe project. The "libs" folder in the AfffdexMe project should be located in AffdexMe\app\libs

-Copy the "libs\armeabi-v7a" folder of the SDK into a new folder named "jniLibs" in the AffdexMe project. The "armeabi-v7a" folder in the AffdexMe project should thus be located in AffdexMe\app\src\main\jniLibs\armeabi-v7a

-Copy the contents of the "assets" folder of the SDK into the location AffdexMe\app\src\main\assets

Step 3: Open AffdexMe in Android Studio

Android Studio:

- File->Open
- Browse to and select the AffdexMe project. Android Studio will usually display a valid Android Studio project with a distinct green icon. Select 'OK'.

Step 4: Add your license file to the project

- Copy your Affectiva-provided license file to your project's assets/Affdex folder.
- In your IDE, edit the source file MainActivity.java, and in the initializeCameraDetector() method, edit the following line to refer to your license file:

```
detector.setLicensePath("Affectiva.license");
```

That's it! You should now be able to build and run the AffdexMe app.

Where to go from here

We're excited to help you get the most of our SDK in your application. Please use the following ways to contact us with questions, comments, suggestions ... or even praise!

Email: sdk@affectiva.com

Web: http://www.affdex.com/mobile-sdk