Affdex SDK for Android

Developer Guide
For SDK version 1.2

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:

- **Introducing the SDK.pdf**
- **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.

The SDK requires access to external storage on the Android device, and Internet access for collecting anonymous analytics (see "A Note about Privacy" in "Introducing the SDK"). Include the following in your app's AndroidManifest.xml:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

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
- A note about SDK analytics (Flurry)
- 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: MeasureUp.
- 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 facial expression metric (e.g. call setDetectSmile() to detect smiles). Several facial expressions can be detected by the SDK, as described in "Introducing the SDK". See the "Options" section below for more information on the different options available.
- Call `start()` to start processing. Note the types of exceptions that can be thrown and handle them as desired.
- Call `reset()` to reset the baselines used to determine facial expressions. If you are pushing your own images, you may reset the sequence of the timestamps after this call.
- 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 MeasureUp.

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 MeasureUp.

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 (specify null for this argument if you don't care about previewing the frames).

  ```
  public CameraDetector(Context context, CameraType cameraType,
                        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);
  ```

- If a non-null `SurfaceView` was passed into the constructor, it will be automatically resized to match the aspect ratio of the camera images. The `SurfaceView` resizes itself to fit as much of its parent `ViewGroup` as possible. Subscribe to the `CameraDetector.CameraSurfaceViewListener` interface to be notified when the `SurfaceView` size has been changed.

- Call `start()` to start processing. Note that if the camera is already in use, an exception will be thrown. If successful, you will start receiving calls to `onImageResults()`.
- When you are done, call `stop()`.

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 facial expressions

The Affdex SDK can detect a variety of facial expressions, yielding metric scores for the expressions you configure. Affdex expression metrics are described in detail in the "Introducing the SDK" document. By default, no 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 2 FPS:

```
setMaxProcessRate(2);
```

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 maximimum 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 facial expression scores corresponding to the expressions previously configured on the Detector. Scores are generally values from 0-100, representing the expression as a percent, with the exception of “valence” which ranges between -100 and +100.

The follow excerpt from MeasureUp displays the expression scores using TextViews:

```
public void onImageResults(List<Face> faces, Frame image, float
  timeStamp) {
  [some code omitted for brevity…]

  Face face = faces.get(0);
  updateMetricView(smilePct, face.getSmileScore());
  updateMetricView(browFurrowPct, face.getBrowFurrowScore());
  updateMetricView(browRaisePct, face.getBrowRaiseScore());
  updateMetricView(engagementPct, face.getEngagementScore());
  updateMetricView(lipDepressorPct, face.getLipCornerDepressorScore());
  updateMetricView(valencePct, face.getValenceScore());
}
private void updateMetricView(TextView view, float value) {
  String valAsPercent = String.format(Locale.US, "%.0f%%", value);
  view.setText(valAsPercent);
}
```

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

The following excerpt from MeasureUp gets the face points and displays them on the camera preview SurfaceView via a Canvas.

```
public void onImageResults(List<Face> faces, Frame image, float
  timeStamp) {
  [some code omitted for brevity…]

  Face face = faces.get(0);
  if (showPoints) {
    Bitmap facePointsBitmap = Bitmap.createBitmap(width, height,
      Bitmap.Config.ARGB_8888);
    Canvas canvas = new Canvas(facePointsBitmap);
    PointF[] points = face.getFacePoints();

    float radius = ((float) width) / 160;
    for (int i = 0; i < points.length; i++) {
      // The camera preview is displayed as a mirror, so X pts
      // have to be reversed
      canvas.drawCircle(width - points[i].x - 1, points[i].y,
        radius, facePointPaint)
    }
    pointsView.setImageBitmap(facePointsBitmap);
  }
}
```

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 MeasureUp sample app

The SDK comes with a sample application called MeasureUp 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 MeasureUp sample app

In a browser, open http://affdex-sdist.s3.amazonaws.com/Android/AndroidMeasureUp-1.1.zip, save it locally, and unzip it to wherever you normally put your Android projects.

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

Copy the "libs" and "assets" folders from the Affdex SDK installation folder to the folder where you unzipped the MeasureUp project.

Step 3: Import the MeasureUp sample project into your IDE

Eclipse:

- File->Import, choose General->Existing Projects into Workspace, click Next

- Browse to and select the folder where you unzipped the MeasureUp project, then click Finish

Note: you will see an error related to an unresolved resource ('@integer/google_play_services_version') in AndroidManifest.xml, which we will resolve in the next step.

Android Studio:

- File->Import Project
- Browse to and select the folder where you unzipped the MeasureUp project, then click OK.
- On the Import Project from ADT (EclipseAndroid) dialog, specify a folder for the imported project location, and click Next, then click Finish.

Note: you will see an error related to an unresolved resource ('@integer/google_play_services_version') in AndroidManifest.xml, which we will resolve in the next step.

Step 4: Add project dependences for Google Play Services and Android v4 Support:

Projects using the SDK need to include two additional libraries that are packaged with the Android SDK: the Google Play Services library and the Android v4 Support library.

To add the Google Play Services library to the project, follow instructions at: http://developer.android.com/google/play-services/setup.html

To add the Android v4 Support library to the project, follow the instructions at: https://developer.android.com/tools/support-library/setup.html

Step 5: 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 onCreate 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 MeasureUp app.

A Note about SDK Analytics (Flurry)

The Affdex SDK for Android, and therefore by extension, any application that uses it, leverages the Flurry Analytics service to log events. Due to a limitation in Flurry, an app cannot have two Flurry sessions open simultaneously, each logging to different Flurry accounts. Therefore, apps that use the Affdex SDK for Android cannot also use the Flurry Analytics service themselves, as doing so could result in the app's analytics events being logged to the Affectiva Flurry account, or vice versa. We are exploring ways to address this limitation in future releases.

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