Affdex SDK for Android

Developer Guide
For SDK version 1.0

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 on the camera and collects images. Sample app: MeasureUp.
- Video file mode: provide to the SDK a path to a video file.

- Push mode: provide to the SDK individual frames of video and their timestamps.
- Photo mode: provide a discrete image to the SDK (unrelated to any other image).

To enable the mode you want, call the appropriate constructor of the Detector class.

SDK calling overview

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

- Call the Detector constructor for 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, turn on at least one Classifier to detect facial expressions. Several facial expressions can be detected by the SDK, as described in "Introducing the SDK". Each expression is detected by a Classifier, which is represented by a number (e.g. 0 - 100). See the "Options" section below for more information on the different options available.
- Call `start()` to start processing. Be sure to check for returned errors.
- If you are pushing your own images (push mode or photo mode), call `processImage()` with each image.
- When you are done processing, call `stop()`.

To receive results from the SDK, implement the `Detector.Detection` interface. The interface provides results of the expressions detected for each image frame. A detailed example is provided later in this document.

If you are interested in simply when a face appears and disappears, this interface will notify you with calls to `onFaceDetectionStarted()` and `onFaceDetectionStopped()`. 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, as always, implement the interface `Detector.Detection`. Then follow this sequence of SDK calls:

- Call the Camera Mode Detector constructor. `CameraType` specifies the front or back camera. The `cameraPreviewView` is optional (you may set this argument to null).

```
public Detector(Context context, Detection detectorCallback,
                CameraType camera, 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 Classifier to detect facial expressions.

```
setClassifierOn(Classifier.SMILE, true)
```

- Call `start()` to start processing. If the camera is already in use, an error will be returned. 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:

- Call the Video File Mode Detector constructor. The last argument, filePath, is the path to your video file.

```
public Detector(Context context, Detection detectorCallback,
                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 Classifier to detect facial expressions.

```
setClassifierOn(Classifier.SMILE, true)
```

- Call `start()` to start processing. You will start receiving calls to `onImageResults()`.
- When the video file has finished processing, you will receive a call to `onProcessingFinished()`.
- When you are done processing, call `stop()`.

Push 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.

- Call the following Detector constructor, with the last argument, `discreteImages`, set to false.

```
public Detector(Context context, Detection detectorCallback,
      boolean discreteImages)
```

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

```
setClassifierOn(Classifier.SMILE, true)
```

- Call `start()` to start processing.
- For each video frame, create an Affdex Frame from your frame (Bitmap, RGBA, and YUV420sp/NV21 formats are supported).
- Call `processImage` with the Affdex Frame and timestamp of the frame:

```
public abstract void processImage(Frame facePicture, float
timestampSec);
```

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

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

Photo Mode (a.k.a. Discrete Image Mode)

Use Photo Mode for processing discrete images that are unrelated to each other (that is, they are not sequential frames of a video). Discrete images are processed by the SDK without the benefit of the history of previous images, with different algorithms and data than are used with video frames that have a timestamp.

- Call the following Detector constructor, with the last argument, `discreteImages`, set to true.

```
public Detector(Context context, Detection detectorCallback,
      boolean discreteImages)
```

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

```
setClassifierOn(Classifier.SMILE, true)
```

- Call `start()` to start processing. You will start receiving calls to `onImageResults()`.
- For each video frame, create an Affdex Frame from your frame (Bitmap, RGBA, and YUV420sp/NV21 formats are supported).
- Call `processImage` with the Affdex Frame:

```
public abstract void processImage(Frame facePicture);
```

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

Options

This section describes various options for operating the SDK.

Classifiers

Expressions are detected by classifiers. Classifiers employ the machine learning algorithms that yield expression metrics for the facial expressions you specify. Affdex expression metrics are described in detail in the “Introducing the SDK” document. By default, all classifiers are

disabled. Classifiers can be enabled or disabled with this function, whose second argument determines the enable/disable status:

```
setClassifierOn(Classifier.SMILE, true)
```

See the Classifier class in the Javadoc to see which classifiers are available to be used.

Face Points

The SDK uses facial feature points on the eyes, mouth and other parts of the face as part of detecting expressions. To receive the location of these points relative to the image or video frame being processed, call:

```
setReturnFacePoints(true);
```

Processing Rate

If you plan to use the camera to process facial frames using the Affdex SDK, you can specify the maximum number of frames per second that the SDK will process. This is helpful to balance battery life with your processing requirements. 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

Once you implement the `Detector.Detection` interface, you will have to implement the callback `onImageResults()`, which is called by the SDK for every frame.

This method receives these parameters:

1. A list of Metrics, one for each enabled classifier and one for the face points, if enabled.

2. The image processed, as an Affdex Frame (a wrapper type for images, including Bitmaps, for example).
3. The time of occurrence of the image.

A Metric can be either a numeric value describing the facial expression (ClassifierMetric), or an array of facial feature points (PointsMetric), which are points on the face used to detect facial expressions.

The following example from MeasureUp iterates through the list of returned Metrics. MeasureUp draws facial feature points on top of the camera view via a Canvas. In this code snippet we show the facial feature points extracted from the PointsMetric, and drawn on a Canvas.

The following code checks to see if each Metric is a PointsMetric or a ClassifierMetric. ClassifierMetrics are generally values from 0-100, representing the expression as a percent. In this example, they are formatted as text and printed to the TextView smilePct.

```
@Override
public void onImageResults(List<Metric> metrics, Frame image, double timeOfOccurence)
{
  for (Metric metric : metrics) {
    if (metric instanceof PointsMetric) {
      // Draw the facial feature points
      PointF[] points = ((PointsMetric) metric).getPoints();
      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, 4, facePointPaint);
      }
    }else{
      // Display the value of the classifier
      ClassifierMetric classification = (ClassifierMetric)metric;
      float value = classification.getValue();
      if (Float.isNaN(value))  { // NaN means no face found, so no value
        return; // none of the metrics will have values, so return
      }
      String valAsPercent = String.format(Locale.US, "%.0f%%", value);
      switch (classification.getType()) {
      case SMILE: smilePct.setText(valAsPercent);
        break;
      default:
        break;
      }
    }
  }
}
```

If you have called `setMaxProcessRate()` (which only applies in Camera Mode), you may receive unprocessed frames. Unprocessed frames occur when you set the SDK frame rate lower than the camera frame rate. The `metrics` parameter will be null. If you don't want to receive unprocessed frames, call the following method:

```
setSendUnprocessedFrames(false);
```

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