Affdex SDK for Windows

SDK Developer Guide

# Introduction

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 as a software development kit for Windows. Affdex SDK turns your ordinary app into an extraordinary app by emotion-enabling it to respond in real-time to user emotions.

In this document, you will become familiar with integrating the Affdex SDK into your application. There is a section devoted to each platform supported. Please take time to read this document and feel free to give us feedback at sdk@affectiva.com.

# Affdex SDK for Windows

Affdex SDK for Windows enables the integration of Affectiva's emotion recognition engine into Windows applications. The SDK exposes APIs that can be used from C++ applications and .NET languages. This developer guide is a general overview of the APIs included in the SDK; please consult the API class documentation for the syntax specifics for using the different classes.

## Getting Started

The following are included in the SDK installer:

* + **docs,** documentation files for both the C++ and .NET APIs and licenses.
  + **bin,** packaged native dynamic linked library and .NET assemblies.
  + **lib**, packaged native library
  + **data,** data files required for the SDK runtime.

Affectiva makes source available for sample applications that use the SDK. You can find these source examples on our site: <https://github.com/Affectiva/win-sdk-samples>.

## Requirements & Dependencies

*Hardware requirements (recommended)*

* + Processor, 2 GHz
  + RAM, 1 GB
  + Disk Space (min) : 950 MB

*Runtime Requirements*

* + Visual C++ Redistributable runtime for VS 2013
  + Microsoft .NET framework v 4.0 (.NET SDK only)

The software runtime requirements are installed automatically by the SDK installer.

*Supported operating systems*

* + Windows 7 and above

# Using the SDK

Since the purpose of the SDK is to detect facial expressions and their underlying emotions from facial images. Facial images can be captured from different sources:

* + Camera: a webcam that is connected to the device.
  + Video: a video file on a device's local storage.
  + Frames: a sequence of timed images.
  + Photos: a facial photo

For each of the different sources, the SDK defines a detector class that can handle processing images acquired from that source. There are a set of comon steps needed to start using a detector.

## Creating a Detector

First step is to instantiate a detector that matches the source. Each detector expects a different set of parameters in their constructor that is dependent on their functionality, for example the FrameDetector constructor expects two parameters, a buffer size, which is necessary for setting the capacity (number of frames) of the internal FrameBuffer and a process frame rate, which can be used to throttle the maximum number of frames that get processed per second. By default, the process frame rate is set to 30.

FrameDetector(int bufferSize, int processFrameRate);

**Configuring a Detector**

In order to succesfully initialize the detector, a valid license file must be provided. Each license file issued by Affectiva is time bound and will only work for a fixed period of time shown in the license file, after which the SDK will throw an AffdexLicenseException. The location of the license file must be indicated by calling the following method with the fully qualified path to it:

void setLicensePath( String licensePath);

The Affdex classifier data files are used in frame analysis processing. These files are supplied as part of the SDK. The location of the data files on the phyiscal storage needs to be passed to a detector in order to initalize it by calling the following with the fully qualified path to the folder containing them:

void setClassifierPath(String classifierPath);

The Detectors use callback or interface classes to communicate events and results. The event listeners need to be initialized before the detector is started:

The FaceListener is a client callback interface which sends notification when the detector has started or stopped tracking a face. Call the following method to set the FaceListener:

void setFaceListener(FaceListener listener);

The ImageListener is a client callback interface which delivers information about an image which has been handled by the Detector. Call the following methods to set the ImageListener:

void setImageListener(ImageListener listener);

The ProcessStatusListener is a callback interface which provides information regarding the processing state of the detector. Call the following methods to set the ProcessStatusListener:

void setProcessStatusListener(ProcessStatusListener listener);

## Setting the Classifiers

The following methods are available to turn on or off the detection of various classifiers.  
By default, all classifiers are turned off (set to false).

bool getDetectAttention();

void setDetectAttention(bool detectAttention);

bool getDetectSmile();

void setDetectSmile(bool detectSmile);

bool getDetectBrowRaise();

void setDetectBrowRaise(bool detectBrowRaise);

bool getDetectBrowFurrow();

void setDetectBrowFurrow(bool detectBrowFurrow);

bool getLipCornerDepressor();

void setDetectLipCornerDepressor(bool lipCornerDepressor);

bool getValence();

void setDetectValence(bool detectValence);

bool getDetectEngagement();

void setDetectEngagement(bool detectEngagement);

## Starting a Detector

After a detector is configured using the methods above, the detector initialization can be triggered by calling the start method:

void start();

Likewise, stopping the detector can be done as follows:

void stop();

The processing state can be reset. This method resets the context of the video frames. Additionally Face IDs and Timestamps are set to zero (0):

void reset();

In order to determine whether the detector is currently running, call the following:

bool isRunning();

This returns the state of the detector. If detector is running; it returns **true**, else **false.**

Detectors

## For each of the possible sources of facial frames, the SDK defines a detector class to consume and process images from these sources.

## FrameDetector

The FrameDetector tracks expressions in a sequence of real-time frames. It expects each frame to have a timestamp that indicates the the time the frame was captured. The timestamps arrive in an increasing order. The FrameDetector will detect a face in an frame and deliver information on it to you, including the facial expressions.

The FrameDetector constructor expects two parameters, a buffer size (which is necessary for setting the number of frames of the internal frame buffer), and a process frame rate (useful for throttling the maximum number of frames processed per second). By default, the process frame rate is set to 30. If the buffer becomes full because processing cannot keep up with the supply of frames, the oldest unprocessed frame is dropped.

FrameDetector(int bufferSize, int processFrameRate);

After successfully initializing the detector using the start method. The frames can be passed to the detector by calling the process method.

void process(Frame frame);

## CameraDetector

Using a webcam is a common way to obtain video for facial expression detection. The CameraDetector can access a webcam connected to the device to capture frames and feed them directly to the facial expression engine.

The constructor of the CameraDetector class expects the camera ID, the number of frames to capture per second and the number of frames to process per second.

CameraDetector(int cameraId=0, double cameraFPS=15, double processFPS);

An instance of the CameraDetector can also be created without any parameters. In this case, the detector connects to the first camera on the device list and assumes the capture frame rate to be 15 frames per second.

CameraDetector();

In addition to all of the methods common between all of the detectors, methods are available to set the camera ID. The camera ID must be a positive number:

void setCameraId(int cameraId);

The capture frame rate can also be set or reset. The frame rate must be a positive number greater than zero (0):

void setCameraFPS(double cameraFPS);

## VideoDetector

Another common use of the SDK is to process previously captured video files. The VideoDetector helps streamline this effort by decoding and processing frames from a video file. Like the FrameDetector, the constructor accepts a parameter for processing frames per second. This parameter regulates how many frames from the video stream get processed. During processing, the VideoDetector decodes and processes frames as fast as possible and actual processing times will depend on CPU speed. Appendix I includes a list of recomended video codecs that are compatible with the detector.

VideoDetector(double processFPS);

Once the detector is started, the processing begins by calling the process function, the path to video file you are processing is passed in as a parameter:

void process(String path);

To stop the processing, the stop method can be used, however it is best to only call this method once video processing has completed.

void stop();

## PhotoDetector

The PhotoDetector class is used for streamlining the processing of still images. Since photos lack any continuity over time, the expression and emotion detection is performed independently on each frame and the timestamp is ignored. Due to this fact, the underlying emotion detection may return different results than the video based detectors.

Like the FrameDetector, the PhotoDetector must be started:

void start();

and stopped:

void stop();

Photos are processed using the following method:

void process(Frame frame);

Unlike other detectors, photo processing is done **synchronously**. Calls to process will not return until processing is complete and the ImageListener callback methods have complete.

## Data Structures

## Frame

The Frame is used for passing images to and from the detectors. To initialize a new instance of a frame, you must call the frame constructor. The frame constructor requires the width and height of the frame and a pointer to the pixel array representing the image. Additionally, the color format of the incoming image must be supplied. (See below for supported color formats.)

Frame(int frameWidth, int frameHeight, ref byte[] pixels, COLOR\_FORMAT frameColorFormat);

A timestamp can be optionally set. It is required when passing the frame to the FrameDetector, and is not when using the PhotoDetector. The timestamp is automatically generated by querying the system time when using the CameraDetector, and is decoded from the video file in the case of the VideoDetector.

Frame(int frameWidth, int frameHeight, ref byte[] pixels, COLOR\_FORMAT frameColorFormat, float timestamp);

The following color formats are supported by the Frame class:

enum class COLOR\_FORMAT

{

RGB,

BGR

};

To retrieve the color format used to create the frame, call:

COLOR\_FORMAT getColorFormat();

To get the Frame image's underlying byte array of pixels, call this method:

byte[] getBGRByteArray();

To retrieve the length of the frame's byte array in addition to the image's width and height in pixels, call the following methods:

int getBGRByteArrayLength();

int getWidth() const;

int getHeight() const;

Client applications have the ability to get and set the Frame's timestamp through the following:

float getTimestamp() const;

void setTimestamp(float value);

**Face**

The Face class represents a face found with a processed frame. The following methods are available on the Face class to retrieve classifiers' values:

float getAttentionScore();

float getBrowFurrowScore();

float getBrowRaiseScore();

float getEngagementScore();

float getLipCornerDepressorScore();

float getSmileScore();

float getValenceScore();

Interocular distance, the distance between the two outer eye corners can be retrieved by calling:

float getInterocularDistance();

The head orientation represented by three angles (pitch, yaw, roll) can be retrieved by calling:

Orientation getOrientation();

The Face object also enables users to retrieve the feature points associated with a face. To retrieve the number of points in the collection, call:

int getFeaturePointCount();

To access the actual points call:

FeaturePoint[] getFeaturePoints();

This returns a vector of FeaturePoint objects.

Note:

In the .NET SDK, the values can be retrieved by inspecting the values of the properties of the class. Please refer to the SDK technical documentation for more details.

## Orientation

An Orientation is a representation of the orientation of the head in a 3-D space using Euler angles (pitch, yaw, roll):

struct Orientation

{

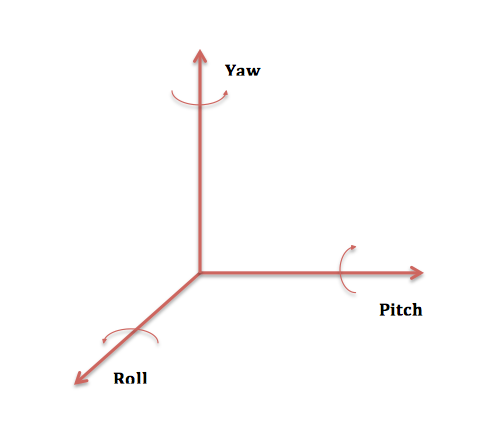
float pitch;

float yaw;

float roll;

};

## FeaturePoint

A FeaturePoint is the cartesian coordinates of a facial feature on the source image and is defined as the following:

struct FeaturePoint

{

int id;

float x;

float y;

};

The following are the IDs and facial feature names tracked by the SDK.

| ID | Facial feature name | ID | Facial feature name |
| --- | --- | --- | --- |
| 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 |

## Listeners

**ImageListener**

This interface delivers information about the images and faces captured by a detector. The ImageListener contains two client callback methods:

onImageResults returns the processed frame and a dictionary of the faces found. An individual entry in the dictionary is comprised of a face ID and a Face object which contains metrics about the face. If the image was processed but no face was found, the returned dictionary will be empty. The detectors track a single face, the face that occupies the largest area in the image. A Future release of the SDK will allow tracking mutiple faces in an image.

virtual void onImageResults(Dictionary<int, Face> faces, Frame image);

onImageCapture returns all the frames passed to the detector.

virtual void onImageCapture(Frame image);

**FaceListener**

This interface provides methods that the Detector uses to communicate to users of the class. The following method indicates that the face detector has detected a face and has begun tracking it. The receiver should expect that tracking continues until detection has stopped.

virtual void onFaceFound(float timestamp, int faceId);

The following method indicates that the face detector has stopped tracking a face, and is called when a face is no longer detected. The receiver should expect that there is no face tracking until the detector is started again.

virtual void onFaceLost(float timestamp, int faceId);

**ProcessStatusListener**

This is a client listener interface which delivers information on the state of the processing.

The ProcessStatusListener contains callbacks to inform about the status of the processing. onProcessingFinished is called when a video file has completed processing. onProcessingException is called if an AffdexException is encountered during the processing. If either of those callbacks is triggered, no further calls to any registered ImageListeners should be expected and it is safe to stop the detector.

virtual void onProcessingFinished();

virtual void onProcessingException(AffdexException exception);

# Where to Go From Here

For detailed class documentation, see the documentation folder.

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

**Email:** [sdk@affectiva.com](mailto:sdk@affectiva.com)

**Web:** [www.affdex.com](http://www.affdex.com)

## Appendix I

**Supported File Types for Video Processing**

This software uses FFmpeg code licensed under the LGPLv2.1 for video decoding. FFmpeg supports decoding many video codecs. The following video codecs were tested and are known to work:

**Video Containers**

.MOV, .WMV, .FLV, .AVI, .MP4, .WEBM

**Video Codecs**

FOURCC Description

CVID Cinepak

FMP4 FFMPEG

FLV1 FLV / Sorenson Spark / Sorenson H.263 (Flash Video)

H264 H.264 / AVC / MPEG-4 AVC / MPEG-4 part 10

IV50 Intel Indeo Video Interactive 5

MPG1 MPEG-1 video

MP43 MPEG-4 part 2 Microsoft variant version 3

MJPG Motion JPEG

SVQ1 Sorenson Video 1

WMV1 Windows Media Video 7

WMV2 Windows Media Video 8

WMV3 Windows Media Video 9

VP80 On2 VP8