



Luxand, Inc.
<https://www.luxand.com>

Luxand FaceSDK

8.0.1

Face Detection and Recognition Library

Developer's Guide

Table of Contents

Overview	6
Requirements.....	6
Technical Specifications.....	7
Face Detection.....	7
Face Matching	7
Live Video Recognition with Tracker API	7
Facial Feature Detection	8
Eye Centers Detection	8
Gender Recognition	8
Age Recognition	8
Facial Expression Recognition.....	8
Liveness detection	8
Multi-Core Support.....	9
Library Size	9
Installation	9
Windows	9
Linux/macOS	9
Directory Structure	9
Sample Applications.....	10
Using FaceSDK with Programming Languages.....	12
Using with .NET (C# and VB).....	12
Using CImage class in .NET	13
CImage();	14
CImage(Int);	14
CImage.ReloadFromHandle();.....	14
Using with C/C++	14
Using with Delphi	15
Using with Java	15
Using with Cocoa	15
Using with VisualBasic 6.0.....	15
Using with iOS.....	16
Using with Android.....	16
Using with Python.....	16
Unicode support	16
Redistributables.....	16
Usage Scenarios	17
Library Activation.....	18
FSDK_GetHardware_ID Function.....	18
FSDK_ActivateLibrary Function	19
FSDK_GetLicenseInfo Function.....	19
Initialization	20
FSDK_Initialize Function	20
FSDK_Finalize Function.....	20
Configuration.....	21
FSDK_SetParameter Function	21
FSDK_SetParameters Function.....	22

FaceSDK Parameters.....	22
<i>Face detection parameters</i>	23
Working with Images.....	23
FSDK_CreateEmptyImage Function.....	24
FSDK_LoadImageFromFile Function.....	24
FSDK_LoadImageFromFileW Function.....	25
FSDK_SaveImageToFile Function.....	26
FSDK_SaveImageToFileW Function.....	26
FSDK_LoadImageFromBuffer Function.....	27
FSDK_LoadImageFromJpegBuffer Function.....	28
FSDK_LoadImageFromPngBuffer Function.....	29
FSDK_GetImageBufferSize Function.....	29
FSDK_SaveImageToBuffer Function.....	30
FSDK_LoadImageFromHBitmap Function.....	31
FSDK_SaveImageToHBitmap Function.....	31
FSDK_LoadImageFromCLRImage Function.....	32
FSDK_SaveImageToCLRImage Function.....	32
FSDK_LoadImageFromAWTImage Function.....	33
FSDK_SaveImageToAWTImage Function.....	33
FSDK_SetJpegCompressionQuality.....	34
FSDK_GetImageWidth Function.....	34
FSDK_GetImageHeight Function.....	35
FSDK_CopyImage Function.....	35
FSDK_ResizeImage Function.....	36
FSDK_RotateImage Function.....	37
FSDK_RotateImageCenter Function.....	38
FSDK_RotateImage90 Function.....	38
FSDK_CopyRect Function.....	39
FSDK_CopyRectReplicateBorder Function.....	40
FSDK_MirrorImage Function.....	41
FSDK_FreeImage Function.....	42
Face Detection.....	42
Face Detection Models.....	42
Face Detection on Thermal Images.....	43
Data types.....	44
FSDK_DetectFace Function.....	45
FSDK_DetectMultipleFaces Function.....	46
FSDK_SetFaceDetectionParameters Function.....	47
FSDK_SetFaceDetectionThreshold Function.....	49
Facial Feature Detection.....	49
FSDK_DetectFacialFeatures Function.....	50
FSDK_DetectFacialFeaturesInRegion Function.....	51
FSDK_DetectEyes Function.....	53
FSDK_DetectEyesInRegion Function.....	54
Detected Facial Features.....	55
Mask-on Face Detection.....	58
Face Matching.....	59
FSDK_GetFaceTemplate Function.....	60
FSDK_GetFaceTemplateInRegion Function.....	61
FSDK_GetFaceTemplateUsingEyes Function.....	62
FSDK_GetFaceTemplateUsingFeatures Function.....	63
FSDK_MatchFaces Function.....	64
FSDK_GetMatchingThresholdAtFAR Function.....	65
FSDK_GetMatchingThresholdAtFRR Function.....	66

Gender, Age and Facial Expression Recognition	67
FSDK_DetectFacialAttributeUsingFeatures Function	67
FSDK_GetValueConfidence Function	69
Liveness Detection	70
Passive Liveness	70
Active Liveness	70
Thermal Face Detection	71
Working with Cameras.....	71
Data Types	71
FSDK_InitializeCapturing Function	72
FSDK_FinalizeCapturing Function	73
FSDK_SetCameraNaming Function	73
FSDK_GetCameraList Function	74
FSDK_GetCameraListEx Function	75
FSDK_FreeCameraList Function	76
FSDK_GetVideoFormatList Function	76
FSDK_FreeVideoFormatList Function	77
FSDK_SetVideoFormat Function	78
FSDK_OpenVideoCamera Function	78
FSDK_OpenIPVideoCamera Function	79
FSDK_SetHTTPProxy Function	80
FSDK_GrabFrame Function	81
FSDK_CloseVideoCamera Function	82
Tracker API: Face Recognition and Tracking in Video Streams.....	82
What is Tracker API	82
Understanding Identifiers	85
<i>A subject can have several identifiers</i>	<i>85</i>
<i>Merger of identifiers</i>	<i>85</i>
<i>When identifiers are not merged</i>	<i>85</i>
<i>Similar identifiers</i>	<i>85</i>
Tracker Memory	86
<i>Memory available for each subject</i>	<i>86</i>
<i>Imposing memory limits</i>	<i>86</i>
<i>How to set the memory limit</i>	<i>87</i>
Tracker Parameters	87
<i>Face tracking parameters</i>	<i>87</i>
<i>Face recognition parameters</i>	<i>87</i>
<i>Facial feature tracking parameters</i>	<i>88</i>
Tuning for Optimal Performance	89
Using the API	89
<i>Locking identifiers</i>	<i>89</i>
<i>Multiple camera support</i>	<i>90</i>
<i>Storing original facial images</i>	<i>90</i>
<i>Usage Scenario</i>	<i>90</i>
User Interaction with the System	91
<i>Enrollment</i>	<i>91</i>
<i>Dealing with false acceptances</i>	<i>92</i>
Saving and Loading Tracker Memory	92
Recognition Performance	93
<i>Performance measures</i>	<i>93</i>
<i>Understanding storage events</i>	<i>93</i>
<i>How to measure your rate of storage events</i>	<i>93</i>
<i>Understanding FAR</i>	<i>94</i>
<i>Understanding R</i>	<i>94</i>
<i>Choosing Threshold value</i>	<i>94</i>
Gender, Age and Facial Expression Recognition	95

Face, Eye and Facial Feature Tracking	96
<i>Counting the number of people</i>	96
Thread Safety	97
FSDK_CreateTracker Function	97
FSDK_FreeTracker Function	97
FSDK_ClearTracker Function	98
FSDK_SetTrackerParameter Function	98
FSDK_SetTrackerMultipleParameters Function	99
FSDK_GetTrackerParameter Function	100
FSDK_FeedFrame Function	101
FSDK_GetTrackerEyes Function	102
FSDK_GetTrackerFacialFeatures Function	103
FSDK_GetTrackerFacePosition Function	104
FSDK_GetTrackerFacialAttribute Function	104
FSDK_LockID Function	106
FSDK_UnlockID Function	106
FSDK_PurgeID Function	107
FSDK_GetName Function	108
FSDK_SetName Function	108
FSDK_GetIDReassignment Function	109
FSDK_GetSimilarIDCount Function	110
FSDK_GetSimilarIDList Function	111
FSDK_GetAllNames Function	111
FSDK_SaveTrackerMemoryToFile Function	112
FSDK_LoadTrackerMemoryFromFile Function	113
FSDK_GetTrackerMemoryBufferSize Function	114
FSDK_SaveTrackerMemoryToBuffer Function	115
FSDK_LoadTrackerMemoryFromBuffer Function	115
Multi-Core Support.....	116
FSDK_GetNumThreads Function	117
FSDK_SetNumThreads Function	117
Thread Safety.....	118
Migration.....	119
Migration from FaceSDK 7.2, 7.2.1 to FaceSDK 8.0	119
Migration from FaceSDK 7.1 to FaceSDK 7.2, 7.2.1	119
Migration from FaceSDK 6.5.1 to FaceSDK 7.0, 7.1	119
<i>Face Detection</i>	119
<i>Template format changes</i>	120
<i>Removal of libstdc++ dependency on iOS</i>	120
Migration from FaceSDK 6.5 to FaceSDK 6.5.1	120
Migration from FaceSDK 6.3, 6.3.1, 6.4 to FaceSDK 6.5	121
<i>Template format changes</i>	121
<i>Template matching</i>	121
<i>Tracker API changes</i>	121
Migration from FaceSDK 6.2 to FaceSDK 6.3, 6.3.1, 6.4	122
Migration from FaceSDK 6.0, 6.0.1, 6.1 to FaceSDK 6.2	123
Migration from FaceSDK 5.0, 5.0.1 to FaceSDK 6.0, 6.0.1, 6.1	123
Migration from FaceSDK 4.0 to FaceSDK 5.0, 5.0.1	124
Migration from FaceSDK 3.0 to FaceSDK 4.0	124
Migration from FaceSDK 2.0 to FaceSDK 3.0	125
Error Codes	125
Library Information	127

Overview

Luxand FaceSDK is a cross-platform face detection and recognition library that can be easily integrated into the customer's application. FaceSDK offers the API (Application Programming Interface) to detect and track faces and facial features, to recognize gender, age and facial expressions (if a smile is present and if the eyes are open or closed), and to recognize faces on still images and videos. FaceSDK also allows detecting faces on thermal images.

FaceSDK is provided with Tracker API which allows tracking and recognizing faces in live video. Tracker API simplifies working with video streams, offering the functions to tag subjects with names and recognize them further.

The SDK provides the coordinates of [70 facial feature points](#) (including eyes, eyebrows, mouth, nose and face contours). Luxand FaceSDK uses multiple processor cores to speed up recognition. The library supports DirectShow-compatible web cameras on Windows, v4l2-compatible web cameras on Linux and IP cameras with an HTTP/MJPEG interface on every platform.

Luxand FaceSDK is a dynamic link library available for 32-bit and 64-bit versions of Windows and Linux, 64-bit macOS, iOS, Android. The SDK contains interface header files and sample applications for C++, Microsoft Visual C++ 6.0/2005/2008/2010/2012/2013/2015/2017+, Visual Basic .NET 2005/2008/2010+, Microsoft C# .NET 2005/2008/2010+, Borland Delphi 6.0+, Netbeans (Java), Xcode 7+ (iOS), Eclipse ADT (Android), Android Studio (Android), Visual Basic 6.0, C++Builder 6.0 and Python.

Requirements

The FaceSDK library supports the following platforms:

- Windows 2008R2/2012/2016/2019/2022, Windows 7, Windows 8, Windows 10, Windows 11
- Linux (RHEL 7+, CentOS 7+ and other)
- Linux armv7/arm64 (Raspberry Pi2+)
- macOS 10.13+, arm64/x86_64
- iOS 9.0+, arm64/ armv7/ x86_64/ x86(iPhone, iPad, simulator)
- Android 5.0+ (platform version 21+), arm64 (arm64-v8a)/ armv7 (armeabi-v7a)/ x86

An Intel Xeon or AMD Ryzen processor is recommended for better performance.

Minimum system requirements:

- 1 GHz processor
- 256 MB RAM

Recommended system requirements:

- Intel Core i7, i9, Xeon or AMD Ryzen processor
- 2 GB RAM
- DirectShow/v4l2-compatible webcam (on Windows or Linux)
- IP camera with MJPEG interface (like AXIS IP cameras)

Note that the web camera functions are available only on Windows and Linux. IP cameras are accessible on all platforms.

Technical Specifications

The FaceSDK library has the following technical specifications:

Face Detection

- Robust frontal face detection
- Detection of multiple faces in a photo
- Detection of faces on thermal images
- Head rotation support: $-30..30$ degrees of in-plane rotation and $-30..30$ degrees out-of-plane rotation
- Determines in-plane face rotation angle
- Detection speed:
 - Realtime detection (webcam resolution, $-15..15$ degrees of in-plane head rotation): 0.00154 seconds (649 FPS) (AMD*), 0.00863 seconds (116 FPS) (iOS*), 0.01414 seconds (71 FPS) (Android*)
 - Reliable detection (digital camera resolution, $-30..30$ degrees of in-plane head rotation): 0.0081 seconds (AMD), 0.05 seconds (iOS), 0.082 seconds (Android)
- Returned information for each detected face: (x,y) coordinates of face center, face width and rotation angle
- Easy configuration of face detection parameters

Face Matching

- Matching of two faces at given FAR (False Acceptance Rate) and FRR (False Rejection Rate)
- Enrollment time:
 - Webcam resolution, using FSDK_GetFaceTemplate: 0.01417 seconds (71 FPS) (AMD), 0.03725 seconds (27 FPS) (iOS), 0.0777 seconds (13 FPS) (Android)
- Template Size: 1040 bytes
- Matching speed:
 - Single thread, templates per second: 5000000 (AMD), 3205128 (iOS), 242600 (Android)
 - Multiple parallel threads, templates per second: 53763440 (AMD), 10101010 (iOS), 1096791 (Android)
- Returned information: facial similarity level

Live Video Recognition with Tracker API

- Assigns a unique ID to each subject detected in video
- Allows tagging any subject in video with a name, and recognizing it further
- No requirement for a subject to pose to be enrolled
- Constant learning of subjects' appearance
- Provides with estimates of false acceptance rate and recognition rate
- Tracks multiple faces and their facial features
- Recognizes male and female genders
- Recognizes age
- Recognizes facial expressions

- Detects whether a subject is live

Facial Feature Detection

- Detection of 70 facial feature points (eyes, eyebrows, mouth, nose, face contour)
- Detection time (using FSDK_DetectFacialFeaturesInRegion, not including face detection stage): 0.00027 seconds (3733 FPS) (AMD), 0.0003 seconds (3333 FPS) (iOS), 0.00188 seconds (531 FPS) (Android)
- Allowed head rotation: –30..30 degrees of in-plane rotation, –20..20 degrees out-of-plane rotation
- Returned information: array of 70 (x,y) coordinates of each facial feature point

Eye Centers Detection

- Detection of eye centers only, detection time (not including face detection stage): 0.00027 seconds (3752 FPS) (AMD), 0.00028 seconds (3571 FPS) (iOS), 0.00188 seconds (531 FPS) (Android)
- Returned information: two (x,y) coordinates of left eye center and right eye center

Gender Recognition

- Recognition of different genders
- Gender recognition time (not including face and facial feature detection stages): 0.0039 seconds (AMD), 0.0063 seconds (iOS), 0.0122 seconds (Android)
- Returned information: confidence level in each gender

Age Recognition

- Recognition of age
- Age recognition time (not including face and facial feature detection stages): 0.0051 seconds (AMD), 0.0075 seconds (iOS), 0.0131 seconds (Android)
- Returned information: age of a person

Facial Expression Recognition

- Recognizes if the subject smiles and if the eyes are open or closed
- Expression recognition time (not including face and facial feature detection stages): 0.0043 seconds (AMD), 0.0063 seconds (iOS), 0.0122 seconds (Android)
- Returned information: confidence level in each facial expression

Liveness detection

- Detects whether the subject is live (i.e. not a photo/video presented to the camera)
- Works with still images and videos
- Liveness detection time (not including face and facial feature detection stages): 0.017 seconds (AMD), 0.016 seconds (iOS), 0.034 seconds (Android)
- Returned information: the probability of the subject being live

Multi-Core Support

- The library uses all available processor cores when executing face detection or recognition functions to maximize the performance.

Library Size

- The size of the redistributables does not exceed 160 MB for each platform.

* Measured on AMD Ryzen 5 1600X processor with 12 threads, iPhone X with 6 threads, Google Pixel 2 (Snapdragon 835) with 8 threads.

Installation

Windows

To install Luxand FaceSDK, run the installation file:

Luxand_FaceSDK_Setup.exe

and follow the instructions.

FaceSDK is installed to the C:\Program Files\Luxand\FaceSDK {VER} directory on 32-bit machines and to the C:\Program Files (x86)\Luxand\FaceSDK {VER} directory on 64-bit machines by default, where {VER} is the version number of FaceSDK.

FaceSDK is a copy-protected library, and your application must activate the library on startup (see the Library Activation chapter).

Linux/macOS

Unpack the Luxand_FaceSDK.tar.bz2 archive into the desired directory.

Directory Structure

The FaceSDK directory contains the following directories and files:

bin\	FaceSDK binary files
bin\android	FaceSDK Android binaries
bin\iOS	FaceSDK iOS binaries
bin\linux_armv7	FaceSDK Linux/ARMv7 binaries
bin\linux_arm64	FaceSDK Linux/ARM64 binaries
bin\linux_x86	FaceSDK Linux 32-bit binaries
bin\linux_x86_64	FaceSDK Linux 64-bit binaries

bin\osx_x86_64	FaceSDK macOS Intel 64-bit binaries
bin\osx_arm64	FaceSDK macOS ARM64 binaries
bin\win32	FaceSDK Windows 32-bit binaries and stub library files
bin\win64	FaceSDK Windows 64-bit binaries and stub library files
demo\	Demo applications (win32)
include\	Header files
samples\	Sample applications

Sample Applications

FaceSDK is distributed with the following sample applications (they can be found in the FaceSDK `samples\` directory):

1. LiveRecognition

This application receives video from a camera, allows tagging any subject with a name, and then display the name (recognizing the subject). The application utilizes Tracker API. Source code is available on Microsoft C# 2005/2008, Microsoft C# 2010 and higher, iOS (Objective-C and Swift), Android (Eclipse and Android Studio), Borland Delphi 6.0 and higher, C++/GTK 3.0+, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017, Microsoft Visual Basic .NET 2005/2008, Microsoft Visual Basic .NET 2010 and higher, Java and Visual Basic 6.0. The iOS/Android versions are published in the Apple AppStore and in Google Play (“Luxand Face Recognition” application).

2. FaceTracking

This application receives video from a webcam and highlights all detected faces with rectangles. The application utilizes Tracker API. Source code is available on Microsoft C# 2010 and higher, Borland Delphi 6.0 and higher, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017, Microsoft Visual Basic .NET 2010 and higher, Java and Visual Basic 6.0.

3. Lookalikes

This application allows the user to create a database of faces and run a search for the best matches (the most similar face from the database is shown). Source code is available on Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017, Microsoft C# 2010 and higher, Java and Borland Delphi 6.0 and higher. There is an example of working with Microsoft SQL database on Microsoft C# 2010 and higher, and with and SQLite on Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017. To run the Microsoft SQL example, you need to attach the database (located in the DB folder of the sample) to the Microsoft SQL Server.

4. LiveFacialFeatures

This application tracks users’ facial features in real time using a web camera. The coordinates of facial features are smoothed by Tracker API to prevent jitter. Source code is available on Microsoft C# 2010 and higher, Borland Delphi 6.0 and higher,

Java, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017, iOS, Android (Eclipse and Android Studio) and Microsoft Visual Basic .NET 2010 and higher.

5. AgeGenderRecognition

Using Tracker API, this application recognizes the gender and age of a subject looking into a webcam. Source code is available on Microsoft C# 2010 and higher, Borland Delphi 6.0 and higher, Java, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017, iOS, Android (Eclipse and Android Studio) and Microsoft Visual Basic .NET 2010 and higher.

6. ExpressionRecognition

Using Tracker API, this application recognizes if a subject looking into a webcam is smiling, and if the subject's eyes are open or closed. Source code is available on Microsoft C# 2010 and higher, Borland Delphi 6.0 and higher, Java, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017, iOS, Android (Eclipse and Android Studio) and Microsoft Visual Basic .NET 2010 and higher.

7. FacialFeatures

This application opens a photo, detects a face in a photo (only one face, the one that can be detected best), detects facial features and draws a frame around the detected face and detected features. Source code is available on Microsoft C# 2010 and higher, Borland C++ Builder 6.0, Borland Delphi 6.0 and higher, Java, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017, iOS, Android (Eclipse and Android Studio), Microsoft Visual Basic .NET 2010 and higher, Visual Basic 6.0.

8. IPCamera

This application opens an IP camera (allowing the user to specify its address, user name and password), displays the image from the camera and tracks faces. The application utilizes Tracker API. Source code is available on Microsoft C# 2010 and higher, Borland Delphi 6.0 and higher, Java, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017 and higher, Microsoft Visual Basic .NET 2010 and higher.

9. Portrait

This application is for the command line. The application receives a picture, detects a face and, if the face is found, crops it and saves it to a file. Source code is available on C++.

10. Thermal

This application loads a thermal face detection model and allows you to open a grayscale thermal image (which you may have received from a thermal camera), detect faces on the image and draw frames around the detected faces. Source code is available on Microsoft C# 2010 and higher, Microsoft Visual C++ 2005/2008/2010/2012/2013/2015/2017 and higher, iOS, Android and C++.

11. ActiveLiveness

This application asks a subject looking into a camera to rotate their head and smile in a certain way to detect liveness. Source code is available on Microsoft C# 2010 and higher, Microsoft Visual C++ 2017 and higher, iOS, Android, Java and Python.

12. PassiveLiveness

This application automatically detects the liveness of a subject looking into a camera (without any assistance from the subject). Source code is available on Microsoft C# 2010 and higher, Microsoft Visual C++ 2017 and higher, iOS, Android, Java and Python.

13. Advanced

This sample provides source code for .NET wrapper that links facesdk.dll dynamically. Refer to [Using with .NET \(C# and VB\)](#) for details. The sample also provides source code for Java and Python wrappers.

Using FaceSDK with Programming Languages

To access the FaceSDK library functions, you need to use its binary file in your applications. The specific file depends on the platform:

- Windows applications use `facesdk.dll`
- Linux and Android applications use `libfacesdk.so`
- macOS applications use `libfacesdk.dylib`
- .NET applications use `facesdk.NET.dll` and the appropriate binary file (`facesdk.dll`, `libfacesdk.dylib` or `libfacesdk.so`)
- Java applications use `facesdk.jar`, `jna.jar` and the appropriate binary file (`facesdk.dll`, `libfacesdk.dylib` or `libfacesdk.so`)
- iOS applications use `libfacesdk-static.a`
- VB6 applications use `facesdk-vb.dll` in addition to `facesdk.dll`

On Windows, Linux and macOS it is usually recommended to store this file in the directory where the executable file of your application is located. Alternatively, you may keep the file in:

- the working directory of your application
- the directory specified in the path environment variable of your system: `PATH` (Windows), `LD_LIBRARY_PATH` (Linux), `DYLD_LIBRARY_PATH` (macOS).

You need to include interface header files into your application project in order to use FaceSDK.

Using with .NET (C# and VB)

You need to have .NET Framework 4.0+ on your system.

For Microsoft .NET applications, you need to add the .NET component into your project.

Follow these steps to add the component in Visual Studio 2010+:

- Select Project – Add Reference – Browse
- Choose the file `include\.NET\FaceSDK.NET.dll`
- Add the following statement to the beginning of your application:

```
using Luxand
```

After that you may use the methods of the `Luxand.FSDK` namespace for general FaceSDK functions, and `Luxand.FSDKCam` namespace for webcam-related functions. You may refer just to `FSDK` and `FSDKCam` namespaces if `using Luxand` is specified.

Once `FaceSDK.NET.dll` is added to the references, it will be redistributed automatically with your application, so no specific deployment actions are required. However you need to redistribute `facesdk.dll` (or `libfsdk.so` on Linux / `libfsdk.dylib` on macOS) with your application.

By default, the documentation refers to C/C++ declarations of FaceSDK functions. For example, the function to detect a face is referred to as `FSDK_DetectFace` function. To refer to this function in .NET, replace the `FSDK_` prefix with `FSDK.` namespace. Thus, the reference to this function becomes [FSDK.DetectFace](#) (note that webcam-specific functions are located in the `FSDKCam.` namespace; refer to [Working with Web Cameras](#) for details).

If you are using an older version of .NET (for example, 2.0, 3.0 or 3.5) or just need a component for a specific .NET version, you may use the source code available in the `samples\advanced\.NET wrapper` directory. The LiveRecognition sample includes projects for Microsoft C# 2005/2008 and Visual Basic .NET 2005/2008 that are using this wrapper.

Using CImage class in .NET

`CImage` is a class for Microsoft .NET for easy manipulation of images. `CImage` encapsulates an `HImage` handle and provides convenient methods for applying FaceSDK functions to that image.

To start working with `CImage`, just create an instance of it. You can pass a file path, `HImage` handle, `HBITMAP` handle or `System.Drawing.Image` object to the constructor to load the corresponding object into the image. Alternatively, call the constructor without parameters to create an empty image. Refer to the functions [FSDK_LoadImageFromFile](#), [FSDK_LoadImageFromHBitmap](#), [FSDK.LoadImageFromCLRImage](#) and [FSDK_CreateEmptyImage](#) for further details.

A `CImage` instance has three properties: `ImageHandle`, `Width` and `Height`. `ImageHandle` is a handle of the internal representation of the image encapsulated by the class. `Width` and `Height` properties are the width and height of an image in pixels (see [FSDK_GetImageWidth](#) and [FSDK_GetImageHeight](#)). If you alter the `ImageHandle` handle directly (for example, executing an `FSDK.` method applied to that image handle), you must update the `CImage` object by calling the `CImage.ReloadFromHandle()` method. `CImage` throws an exception if any FaceSDK function, called within the `CImage` method, has returned an error.

Most `CImage` methods operating with an image (for example, the `Resize()` method) return the processed image as the result.

The `CImage` destructor releases `ImageHandle`, so there is no need to call `FSDK.FreeImage` explicitly after the instance has been destroyed.

Note that when you pass an existing image handle to the constructor, it will be freed after the destruction of the `CImage` class instance, and become invalid. If you need the original image handle to be valid after the `CImage` class instance is destroyed, consider creating a copy of the image handle and passing the copy to the `CImage` constructor.

CImage();

Creates an empty CImage instance.

Syntax:

```
FSDK.CImage ();
```

CImage(Int);

Creates a CImage instance from image already loaded to FaceSDK.

Syntax:

```
FSDK.CImage (int ImageHandle);
```

Parameters:

ImageHandle – the internal handle of an image already loaded to FaceSDK. The destructor will free the ImageHandle handle.

CImage.ReloadFromHandle();

Updates the internal properties of a CImage instance in accordance with the ImageHandle.

Syntax:

```
FSDK.CImage.ReloadFromHandle ();
```

Using with C/C++

For Microsoft Visual C++ applications, you need to include the header file `include\C\LuxandFaceSDK.h`, and the stub library file `facesdk.lib` into your project.

Follow these steps to add the library to your project:

- Copy `include\C\LuxandFaceSDK.h` into the directory of your project
- For 32-bit applications, copy `bin\win32\facesdk.dll` and `bin\win32\facesdk.lib` into the output directory of your project
- For 64-bit applications, copy `bin\win64\facesdk.dll` and `bin\win64\facesdk.lib` into the output directory of your project
- Choose Project Properties – Linker – Input – Additional Dependencies, and add `facesdk.lib` string
- Choose Project Properties – Linker – General – Additional Library Directories Dependencies, and add `$(OutDir)` string (a reference to the output directory)
- Add the following statement to the beginning of your application:

```
include "LuxandFaceSDK.h"
```

The output directory `$(OutDir)` typically refers to `Debug\` or `Release\` in the directory of your solution. You may change it in the Configuration Properties – General of your project. You may also choose another directory to store the `.lib` file, but it is recommended to keep `facesdk.dll` in the directory where the executable file of your application is located.

You need to redistribute the file `facesdk.dll` with your application.

Using with Delphi

For Delphi applications, put `facesdk.dll` into the working directory of your application and use the `include\Delphi\LuxandFaceSDK.pas` unit in your project.

You need to redistribute the file `facesdk.dll` with your application.

Using with Java

You need JDK 1.6 (or OpenJDK 1.6) to use FaceSDK with Java. The FaceSDK Java wrapper uses JNA (all information about JNA and the actual version can be found at <https://github.com/twall/jna>, but `jna.jar` is included in the distribution for your convenience). The FaceSDK java wrapper works with any IDE, but only Netbeans sample projects are provided with the distribution.

To use FaceSDK in your Netbeans project, follow these steps:

- 1) Add `FaceSDK.jar` (`include\Java\FaceSDK.jar`) and `jna.jar` (`include\Java\jna.jar`) to the Libraries section of the project.
- 2) Add the following imports to your source code:

```
import Luxand.*;
import Luxand.FSDK.*;
import Luxand.FSDKCam.*;
```
- 3) Put the appropriate `facesdk` binaries (`facesdk.dll`, `libfdsk.so` or `libfdsk.dylib`) in the project directory (or to the `/usr/lib` directory if using OpenJDK).

You need to redistribute the FaceSDK binaries (`facesdk.dll`, `libfdsk.so` or `libfdsk.dylib`) as well as `FaceSDK.jar` and `jna.jar` with your application.

Using with Cocoa

If you are using Cocoa to write an application in Xcode, make sure that your source code files (which use FaceSDK) have the `.mm` extension, so they are compiled as Objective-C++. If the files have the `.m` extension, they are compiled as Objective-C and cannot use FaceSDK.

Using with VisualBasic 6.0

For Visual Basic 6.0 applications, put the Visual Basic wrapper (`bin\win32\FaceSDK-VB.dll`) into the project directory and add `LuxandFaceSDK.bas` (`include\VB6\LuxandFaceSDK.bas`) module to your project (Select Project – Add module – Existing and choose a module location). Also you need to put `facesdk.dll` into the application working directory.

Note that Tracker API functions employ the Currency data type to store 64-bit integer values. You need to redistribute both `FaceSDK-VB.dll` and `facesdk.dll` with your application.

Visual Basic 6.0 was released in 1998 and cannot fully support some modern operating systems. Hence we cannot guarantee that the samples written in Visual Basic 6.0 will work seamlessly on any Microsoft Windows versions newer than Windows XP.

Using with iOS

Refer to the [Using with Cocoa](#) section. In Objective-C you should use the C++ syntax within your program.

To enable FaceSDK support in your Xcode iOS project, add `bin/iOS/libfsdk-static.a` and `include/C/LuxandFaceSDK.h` to your project.

Using with Android

For Android Studio you need to copy the directories `armeabi-v7a` and `arm64-v8a` contained in `bin/android/` to the `app/src/main/jniLibs` directory of your project. You also need to add the `include/android/FSDK.java` file to the `app/src/main/java/com/luxand` directory.

For Eclipse ADT you need to copy the subdirectories contained in `bin/android/` to the `libs/` directory of your project. You also need to add the `include/android/FSDK.java` file to the `src/com/luxand/` directory.

The syntax of some functions on Android is different from the corresponding Java syntax due to the usage of JNI instead of JNA.

Note: Only `arm64` (`arm64-v8a`), `armv7` (`armeabi-v7a`), `x86` and `x86_64` architectures are supported by FaceSDK on the Android platform.

The FSDK class is provided in the binary code form only. Therefore, the “com.luxand” package name of this class cannot be changed.

Using with Python

Please refer to the samples in the `samples\advanced\Python` directory.

Unicode support

The library supports Unicode filenames on the Windows platform. If you work with file names in the Unicode character set, use functions [FSDK_LoadImageFromFileW](#) and [FSDK_SaveImageToFileW](#) to open and save files.

Redistributables

The following files may be redistributed with your application:

Windows	<code>bin\win32\facesdk.dll</code> (for 32-bit systems) <code>bin\win64\facesdk.dll</code> (for 64-bit systems) <code>bin\win32\FaceSDK-VB.dll</code> (for Visual Basic 6.0 applications)
Linux	<code>bin\linux_x86\libfsdk.so</code> (for 32-bit systems) <code>bin\linux_x86_64\libfsdk.so</code> (for 64-bit systems) <code>bin\linux_armv7</code> (for armv7 systems) <code>bin\linux_arm64</code> (for arm64 systems)
macOS	<code>bin\osx_x86_64\libfsdk.dylib</code> <code>bin\osx_arm64\libfsdk.dylib</code>

Android	bin\android\arm64-v8a\libfsdk.so bin\android\armeabi-v7a\libfsdk.so bin\android\x86\libfsdk.so bin\android\x86_64\libfsdk.so
iOS	NONE (static libraries)
.NET	include\.NET\FaceSDK.NET.dll
Java (on Windows / Linux / macOS)	include\Java\FaceSDK.jar include\Java\jna.jar
Thermal face detection	thermal.bin (the detection model used in Thermal samples)

Usage Scenarios

The library usage level depends on the functionality required from Luxand FaceSDK.

If you work with video, consider using [Tracker API](#), as the API provides high-level functions to recognize subjects and tag them with names, to track their faces and facial features, and to recognize gender, age and facial expressions. The usage scenario for Tracker API can be found in the [Usage Scenario](#) section of the [Tracker API](#) chapter.

Otherwise, the typical scenario is as follows:

1. Activate FaceSDK by calling up the [FSDK_ActivateLibrary](#) function with the key sent by Luxand, Inc.
2. Initialize FaceSDK by calling up the [FSDK_Initialize](#) function.
3. Load images either from a file, a buffer or the HBITMAP handle ([FSDK_LoadImageFromFile](#), [FSDK_LoadImageFromBuffer](#), [FSDK_LoadImageFromHBitmap](#) functions).
4. Set face detection parameters if needed ([FSDK_SetFaceDetectionParameters](#), [FSDK_SetFaceDetectionThreshold](#)).
5. Use FaceSDK functions:
 - Detect a face ([FSDK_DetectFace](#)) or multiple faces ([FSDK_DetectMultipleFaces](#)) in an image
 - Detect facial features if needed ([FSDK_DetectFacialFeatures](#), [FSDK_DetectFacialFeaturesInRegion](#))
 - Extract a face template from the image ([FSDK_GetFaceTemplate](#), [FSDK_GetFaceTemplateInRegion](#), [FSDK_GetFaceTemplateUsingFeatures](#))
 - Match the face templates ([FSDK_MatchFaces](#)) and acquire the facial similarity level
 - To find out if a face belongs to the same person, calculate the matching threshold at a given FAR or FRR rate ([FSDK_GetMatchingThresholdAtFAR](#) and [FSDK_GetMatchingThresholdAtFRR](#) functions).
6. Finalize the FaceSDK library ([FSDK_Finalize](#) function).

To work with a camera, follow these steps*:

1. Initialize camera capturing ([FSDK_InitializeCapturing](#)).

2. Get a list of web cameras available in the system ([FSDK_GetCameraList](#))
3. Get list of video formats supported by the web camera ([FSDK_GetVideoFormatList](#)).
4. Set the desired video format for the chosen web camera ([FSDK_SetVideoFormat](#)).
5. Open a web camera ([FSDK_OpenVideoCamera](#)) or an IP camera ([FSDK_OpenIPVideoCamera](#)).
6. Grab frames ([FSDK_GrabFrame](#)) in a loop, displaying them and detecting/recognizing faces.
7. Close video camera ([FSDK_CloseVideoCamera](#)).
8. Delete the list of video formats ([FSDK_FreeVideoFormatList](#)).
9. Delete the list of web cameras ([FSDK_FreeCameraList](#)).
10. Finalize capturing ([FSDK_FinalizeCapturing](#)).

*If you work with an IP camera, you should not follow steps 2, 3, 4, 8 and 9.

Library Activation

FaceSDK is a copy-protected library, and must be activated with a license key before its use. You need to pass the license key received from Luxand, Inc. to the [FSDK_ActivateLibrary](#) function before initializing Luxand FaceSDK. Almost all FaceSDK functions will return the FSDKKE_NOT_ACTIVATED error code in case the library is not activated. To retrieve your license information, call [FSDK_GetLicenseInfo](#). This function returns the name the library is licensed to. You may need to use the [FSDK_GetHardware_ID](#) function to obtain your hardware ID if your license is restricted to one machine only. Additionally, you can find out hardware ID by running the hardwareid program (ShowHardwareID.exe for Windows), which is located in the bin directory.

To get a temporary evaluation key from Luxand, Inc., run License Key Wizard from the Start – Luxand – FaceSDK menu. You may also request this key at

<https://luxand.com/facesdk/requestkey/>.

FSDK_GetHardware_ID Function

Generates a Hardware ID code.

C++ Syntax:

```
int FSDK_GetHardware_ID(char* HardwareID);
```

Delphi Syntax:

```
function FSDK_GetHardware_ID(HardwareID: PChar): integer;
```

C# Syntax:

```
int FSDK.GetHardwareID(out string HardwareID);
```

VB Syntax:

```
Function FSDKVB_GetHardwareID(ByRef HardwareID As Byte) As Long
```

Java Syntax:

```
int FSDK.GetHardware_ID(String HardwareID[]);
```

iOS and Android: not implemented.

Parameters:

HardwareID – address of the null-terminated string for receiving the Hardware ID code.

Return Value:

Returns FSDK_OK if successful.

FSDK_ActivateLibrary Function

Activates the FaceSDK library.

C++ Syntax:

```
int FSDK_ActivateLibrary(char* LicenseKey);
```

Delphi Syntax:

```
function FSDK_ActivateLibrary(LicenseKey: PChar): integer;
```

C# Syntax:

```
int FSDK.ActivateLibrary(out string LicenseKey);
```

VB Syntax:

```
Function FSDKVB_ActivateLibrary(ByVal LicenseKey As String) As Long
```

Java and Android Syntax:

```
int FSDK.ActivateLibrary(String LicenseKey);
```

Parameters:

LicenseKey – License key you received from Luxand, Inc.

Return Value:

Returns FSDK_OK if the registration key is valid and not expired.

FSDK_GetLicenseInfo Function

Retrieves license information.

C++ Syntax:

```
int FSDK_GetLicenseInfo(char* LicenseInfo);
```

Delphi Syntax:

```
function FSDK_GetLicenseInfo(LicenseInfo: PAnsiChar): integer;
```

C# Syntax:

```
int FSDK.GetLicenseInfo(out string LicenseInfo);
```

VB Syntax:

```
Function FSDKVB_GetLicenseInfo(ByRef LicenseInfo As Byte) As Long
```

Java and Android Syntax:

```
int FSDK.GetLicenseInfo(String LicenseInfo[]);
```

Parameters:

LicenseInfo – address of the null-terminated string for receiving the license information. This variable should be allocated no less than 256 bytes of memory.

Return Value:

Returns FSDKE_OK if successful.

Initialization

FSDK_Initialize Function

Initializes the FaceSDK library. Should be called before using of any face detection functions.

C++ Syntax:

```
int FSDK_Initialize(char* DataFilePath);
```

Delphi Syntax:

```
function FSDK_Initialize(DataFilePath: PChar): integer;
```

C# Syntax:

```
int FSDK.InitializeLibrary();
```

VB Syntax:

```
Function FSDKVB_Initialize(ByRef DataFilePath As Byte) As  
Long
```

Java and Android Syntax:

```
int FSDK.Initialize();
```

Parameters:

DataFilePath – pointer to the null-terminated string specifying the path where facesdk.dll is stored. An empty string means the current directory. (Note: the parameter is not used since FaceSDK 1.8; an empty string might be passed as this parameter.)

Return Value:

Returns FSDKE_OK if successful or FSDK_IO_ERROR if an I/O error occurs.

FSDK_Finalize Function

Finalizes the FaceSDK library. Should be called when the application is exited.

C++ Syntax:

```
int FSDK_Finalize();
```

Delphi Syntax:

```
function FSDK_Finalize: integer;
```

C# Syntax:

```
int FSDK.FinalizeLibrary();
```

VB Syntax:

```
Function FSDKVB_Finalize() As Long
```

Java and Android Syntax:

```
int FSDK.Finalize();
```

Return Value:

Returns FSDKE_OK if successful.

Configuration

FSDK_SetParameter Function

Sets a parameter for FaceSDK. See the [FaceSDK Parameters](#) section for details.

Note that, to set Tracker parameters, the [FSDK_SetTrackerParameter](#) or [FSDK_SetTrackerMultipleParameters](#) function should be used instead.

C++ Syntax:

```
int FSDK_SetParameter(const char * ParameterName, const char *  
ParameterValue);
```

Delphi Syntax:

```
function FSDK_SetParameter(ParameterName, ParameterValue:  
PAnsiChar): integer;
```

C# Syntax:

```
int FSDK.SetParameter(string ParameterName, string  
ParameterValue);
```

VB Syntax:

```
Function FSDKVB_SetParameter(ByVal ParameterName As String,  
ByVal ParameterValue As String) As Long
```

Java and Android Syntax:

```
int FSDK.SetParameter(String ParameterName, String  
ParameterValue);
```

Parameters:

ParameterName – name of the parameter to be set.

ParameterValue – value of the parameter.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetParameters Function

Sets multiple parameters for FaceSDK. The parameters and their values are specified in the following format:

```
"Parameter1=Value1[;Parameter2=Value2[;...]]"
```

See the [FaceSDK Parameters](#) section for details.

C++ Syntax:

```
int FSDK_SetParameters(const char * Parameters, int *  
ErrorPosition);
```

Delphi Syntax:

```
function FSDK_SetParameters(Parameters: PAnsiChar;  
ErrorPosition: PInteger): integer;
```

C# Syntax:

```
int FSDK.SetParameters(string Parameters, ref int  
ErrorPosition);
```

VB Syntax:

```
Function FSDKVB_SetParameters(ByVal Parameters As String,  
ByRef ErrorPosition As Long) As Long
```

Java and Android Syntax:

```
int FSDK.SetParameters(String Parameters, IntByReference  
ErrorPosition);
```

Parameters:

Parameters – string containing the parameters and the corresponding values to be set.

ErrorPosition – pointer to the integer variable that will receive the position of the character that caused the syntax error in the string.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_SYNTAX_ERROR and sets the value of the ErrorPosition variable in case of syntax error.

Example:

```
int err = 0;  
FSDK_SetParameters("FaceDetectionModel=thermal.bin;TrimOutOfSc  
reenFaces=false;TrimFacesWithUncertainFacialFeatures=false",  
&err);
```

FaceSDK Parameters

FaceSDK allows for setting a number of parameters with the [FSDK_SetParameter](#) or [FSDK_SetParameters](#) function.

Face detection parameters

Note that the Tracker API does not use the face detection parameters set with [FSDK_SetParameter](#) or [FSDK_SetParameters](#). Instead, you should use [FSDK_SetTrackerParameter](#) or [FSDK_SetTrackerMultipleParameters](#).

Also note that additional face detection parameters can be set by calling the [FSDK_SetFaceDetectionParameters](#) and [FSDK_SetFaceDetectionThreshold](#) functions.

FaceDetectionModel – a path to the face detection model file to load. You can use it to load thermal face detection model (see the Thermal sample application). The value “default” can be passed to switch back to the default visual face detection model.

TrimOutOfScreenFaces – determines whether faces that go beyond the edges of the image should be excluded from face detection. The default value is True (such faces aren’t detected). Use True when you extract face templates from the detected faces and match them. Setting the value to False allows you to detect faces in a larger number of cases, but such faces may yield higher false acceptance rates when matching faces.

TrimFacesWithUncertainFacialFeatures – determines whether faces with uncertain facial features should not be detected. The default value is True (faces with uncertain facial features aren’t detected). Should be set to False for a thermal face detection model. Use True when you extract face templates from the detected faces and match them. Setting the value to False allows you to detect faces in a larger number of cases, but such faces may yield higher false acceptance rates when matching faces.

Working with Images

Images are represented as the HImage data type.

C++ Declaration:

```
typedef int HImage;
```

C# Declaration:

```
int Image
```

Delphi Declaration:

```
HImage = integer;  
PHImage = ^HImage;
```

Java and Android Declaration:

```
class HImage {  
    protected int himage;  
};
```

FaceSDK provides a number of functions to load images to the internal representation from files, buffers or HBITMAP handles and to save images from the internal representation to

files, buffers and HBITMAP handles. Each FSDK_LoadImageFromXXXX function creates a new HImage handle, which can be deleted using the [FSDK_FreeImage](#) function.

Note that when you perform multiple stages of recognition on large images (for example, when you first detect a face, and then create its template using [FSDK_GetFaceTemplateInRegion](#)), consider first resizing the image to a smaller size to speed up operations. Note that you must resize the image to dimensions no smaller than the InternalResizeWidth parameter of the [FSDK_SetFaceDetectionParameters](#) function if you perform face detection, in order to keep the same accuracy of face detection.

FSDK_CreateEmptyImage Function

Creates a handle of an empty image. You don't need to call this function before calling FSDK_LoadImageFromXXXX since these functions already create the HImage handle. Should be called before using the [FSDK_CopyImage](#), [FSDK_ResizeImage](#), [FSDK_RotateImage](#), [FSDK_RotateImageCenter](#), [FSDK_RotateImage90](#), [FSDK_MirrorImage](#), [FSDK_CopyRect](#), [FSDK_CopyRectReplicateBorder](#) functions to create the handle of the destination image.

C++ Syntax:

```
int FSDK_CreateEmptyImage(HImage* Image);
```

Delphi Syntax:

```
function FSDK_CreateEmptyImage (Image: PHImage): integer;
```

C# Syntax:

```
int FSDK.CreateEmptyImage(refint Image);
```

VB Syntax:

```
Function FSDKVB_CreateEmptyImage(ByRef Image As Long) As Long
```

Java and Android Syntax:

```
int FSDK.CreateEmptyImage(HImage Image);
```

Parameters:

Image – pointer to HImage for creating the image handle.

Return Value:

Returns FSDKE_OK if successful.

FSDK_LoadImageFromFile Function

Loads the image from a file and provides the internal handle of this image.

C++ Syntax:

```
int FSDK_LoadImageFromFile(HImage* Image, char* FileName);
```

Delphi Syntax:

```
function FSDK_LoadImageFromFile(Image: PHImage; FileName: PAnsiChar): integer;
```


C# Syntax:

```
int FSDK.LoadImageFromFile(ref int Image, string FileName)
```

VB Syntax:

```
Function FSDKVB_LoadImageFromFile(ByRef Image As Long, ByVal  
FileName As String) As Long
```

Java and Android Syntax:

```
int FSDK.LoadImageFromFile(HImage Image, String FileName);
```

CImage Syntax:

```
int FSDK.CImage(String FileName);
```

Parameters:

Image – pointer to HImage for receiving the loaded image handle.

FileName – filename of the image to be loaded. FaceSDK supports the JPG, PNG and BMP file formats.

Return Value:

Returns FSDKE_OK if successful.

FSDK_LoadImageFromFileW Function

Loads the image from a file path in the Unicode character set and provides the internal handle of this image. The function is available only on Windows platforms.

C++ Syntax:

```
int FSDK_LoadImageFromFileW(HImage* Image, wchar_t* FileName);
```

Delphi Syntax:

```
function FSDK_LoadImageFromFileW(Image: PHImage; FileName:  
PWideChar): integer;
```

C# Syntax:

```
int FSDK.LoadImageFromFileW(ref int Image, string FileName)
```

Java Syntax:

```
int FSDK.LoadImageFromFileW(HImage Image, String FileName);
```

Parameters:

Image – pointer to HImage for receiving the loaded image handle.

FileName – filename of the image to be loaded. FaceSDK supports the JPG, PNG and BMP file formats.

Return Value:

Returns FSDKE_OK if successful.

This function is not available in Visual Basic 6.0.

FSDK_SaveImageToFile Function

Saves an image to a file. When saving to .jpg files, you can set the quality of JPEG compression using the [FSDK_SetJpegCompressionQuality](#) function.

C++ Syntax:

```
int FSDK_SaveImageToFile(HImage Image, char* FileName);
```

Delphi Syntax:

```
function FSDK_SaveImageToFile(Image: HImage; FileName:
PAnsiChar): integer;
```

C# Syntax:

```
int FSDK.SaveImageToFile(int Image, string FileName);
```

VB Syntax:

```
Function FSDKVB_SaveImageToFile(ByVal Image As Long, ByVal
FileName As String) As Long
```

Java and Android Syntax:

```
int FSDK.SaveImageToFile(HImage Image, String FileName);
```

CImage Syntax:

```
void FSDK.CImage.Save(string FileName);
```

Parameters:

Image – internal handle of an image to be saved.

FileName – name of file the image will be saved to. FaceSDK saves images in the BMP, PNG or JPG file format. The format to use is recognized by the extension specified in the *FileName* parameter.

Return Value:

Returns FSDKE_OK if successful.

Example

```
int img1;

FSDK_Initialize("");
FSDK_LoadImageFromFile(&img1, "test.bmp"); // load .bmp file
FSDK_SaveImageToFile(img1, "test.jpg"); // save as .jpg
```

FSDK_SaveImageToFileW Function

Saves an image to a file path in the Unicode character set. The function is available only on Windows platforms. When saving to .jpg files, you can set the quality of the JPEG compression using the [FSDK_SetJpegCompressionQuality](#) function.

C++ Syntax:

```
int FSDK_SaveImageToFileW(HImage Image, wchar_t* FileName);
```

Delphi Syntax:

```
function FSDK_SaveImageToFileW(Image: HImage; FileName: PWideChar): integer;
```

C# Syntax:

```
int FSDK.SaveImageToFileW(int Image, string FileName);
```

Java Syntax:

```
int FSDK.SaveImageToFileW(HImage Image, String FileName);
```

Parameters:

Image – internal handle of an image to be saved.

FileName – name of file the image will be saved to. FaceSDK saves images in the BMP, PNG or JPG file format. The format to use is recognized by the extension specified in the *FileName* parameter.

Return Value:

Returns FSDKE_OK if successful.

The function is not available in Visual Basic 6.0

FSDK_LoadImageFromBuffer Function

Loads an image from a buffer and provides the internal handle of this image. The function suggests that the image data is organized in a top-to-bottom order, and the distance between adjacent rows is *ScanLine* bytes (for example, in the 24-bit image, the *ScanLine* value might be 3**Width* bytes if there is no spacing between adjacent rows). The following image modes are supported:

Mode name	Meaning
FSDK_IMAGE_GRAYSCALE_8BIT	8-bit grayscale image
FSDK_IMAGE_COLOR_24BIT	24-bit color image (R, G, B order)
FSDK_IMAGE_COLOR_32BIT	32-bit color image with alpha channel (R, G, B, A order)

C++ Syntax:

```
int FSDK_LoadImageFromBuffer(HImage* Image, unsigned char* Buffer, int Width, int Height, int ScanLine, FSDK_IMAGEMODE ImageMode);
```

Delphi Syntax:

```
function FSDK_LoadImageFromBuffer(Image: PHImage; var Buffer; Width, Height: integer; ScanLine: integer; ImageMode: FSDK_IMAGEMODE): integer;
```

C# Syntax:

```
int FSDK.LoadImageFromBuffer(ref int Image, byte[] Buffer, int
Width, int Height, int ScanLine, FSDK_IMAGE_MODE ImageMode);
```

VB Syntax:

```
Function FSDKVB_LoadImageFromBuffer(ByRef Image As Long, ByRef
Buffer As Byte, ByVal Width As Long, ByVal Height As Long,
ByVal ScanLine As Long, ByVal ImageMode As FSDK_IMAGE_MODE) As
Long
```

Android Syntax:

```
int FSDK.LoadImageFromBuffer(HImage Image, byte Buffer[], int
Width, int Height, int ScanLine, FSDK_IMAGE_MODE ImageMode);
```

Java Syntax:

```
int FSDK.LoadImageFromBuffer(HImage Image, byte Buffer[], int
Width, int Height, int ScanLine, int ImageMode);
```

Parameters:

Image – pointer to HImage for receiving the loaded image handle.

Buffer – pointer to buffer containing image data.

Width – width of an image in pixels.

Height – height of an image in pixels.

ScanLine – distance between adjacent rows in bytes.

ImageMode – mode of an image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_LoadImageFromJpegBuffer Function

Loads an image from a buffer containing JPEG data, and provides the handle of this image.

C++ Syntax:

```
int FSDK_LoadImageFromJpegBuffer (HImage* Image, unsigned
char* Buffer, unsigned int BufferLength);
```

Delphi Syntax:

```
function FSDK_LoadImageFromJpegBuffer (Image: PHImage; var
Buffer; BufferLength: integer): integer;
```

VB Syntax:

```
Function FSDKVB_LoadImageFromJpegBuffer (ByRef Image As Long,
ByRef Buffer As Byte, ByVal BufferLength As Long) As Long
```

Android Syntax:

```
int FSDK.LoadImageFromJpegBuffer(HImage Image, byte Buffer[],
int BufferLength);
```

Parameters:

Image – pointer to HImage for receiving the loaded image handle.

Buffer – pointer to the buffer containing the image data in JPEG format (usually loaded from a JPEG file).

BufferLength – size of buffer in bytes.

Return Value:

Returns FSDK_OK if successful.

This function is not available in .NET and Java.

FSDK_LoadImageFromPngBuffer Function

Loads an image from a buffer containing PNG data and provides the handle of this image.

C++ Syntax:

```
int FSDK_LoadImageFromPngBuffer (HImage* Image, unsigned char* Buffer, unsigned int BufferLength);
```

Delphi Syntax:

```
function FSDK_LoadImageFromPngBuffer (Image: PHImage; var Buffer; BufferLength: integer): integer;
```

VB Syntax:

```
Function FSDKVB_LoadImageFromPngBuffer (ByRef Image As Long, ByRef Buffer As Byte, ByVal BufferLength As Long) As Long
```

Android Syntax:

```
int FSDK.LoadImageFromPngBuffer(HImage Image, byte Buffer[], int BufferLength);
```

Parameters:

Image – pointer to HImage for receiving the loaded image handle.

Buffer – pointer to the buffer containing the image data in PNG format (usually loaded from a PNG file).

BufferLength – size of buffer in bytes.

Return Value:

Returns FSDK_OK if successful.

The function is not available in .NET and Java.

FSDK_GetImageBufferSize Function

Returns the size of the buffer required to store an image.

C++ Syntax:

```
int FSDK_GetImageBufferSize(HImage Image, int * BufSize, FSDK_IMAGEMODE ImageMode);
```

Delphi Syntax:

```
function FSDK_GetImageBufferSize(Image: HImage; BufSize:
PInteger; ImageMode: FSDK__IMAGEMODE): integer;
```

C# Syntax:

```
int FSDK.GetImageBufferSize(int Image, ref int BufSize,
FSDK__IMAGEMODE ImageMode);
```

VB Syntax:

```
Function FSDKVB_GetImageBufferSize(ByVal Image As Long, ByRef
BufSize As Long, ByVal ImageMode As FSDK__IMAGEMODE) As Long
```

Android Syntax:

```
int FSDK.GetImageBufferSize(HImage Image, int BufSize[],
FSDK__IMAGEMODE ImageMode);
```

Java Syntax:

```
int FSDK.GetImageBufferSize(HImage Image, int BufSize[],
int ImageMode);
```

Parameters:

Image – internal handle of an image.

BufSize – pointer to an integer variable to store the calculated buffer size.

ImageMode – desired image mode of a buffer.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SaveImageToBuffer Function

Saves an image to a buffer in the desired image mode. Refer to the [FSDK_LoadImageFromBuffer](#) function description to read more about image modes.

C++ Syntax:

```
int FSDK_SaveImageToBuffer(HImage Image, unsigned char*
Buffer, FSDK__IMAGEMODE ImageMode);
```

Delphi Syntax:

```
function FSDK_SaveImageToBuffer(Image: HImage; var Buffer;
ImageMode: FSDK__IMAGEMODE): integer;
```

C# Syntax:

```
int FSDK.SaveImageToBuffer(int Image, out byte[] Buffer,
FSDK__IMAGEMODE ImageMode);
```

VB Syntax:

```
Function FSDKVB_SaveImageToBuffer(ByVal Image As Long, ByRef
Buffer As Byte, ByVal ImageMode As FSDK__IMAGEMODE) As Long
```

Android Syntax:

```
int FSDK.SaveImageToBuffer(HImage Image, byte Buffer[],  
FSDK_IMAGEMODE ImageMode);
```

Java Syntax:

```
int FSDK.SaveImageToBuffer(HImage Image, byte Buffer[],  
int ImageMode);
```

Parameters:

Image – internal handle of an image to be saved.

Buffer – pointer to the buffer containing the image data.

ImageMode – desired mode an image will be saved in.

Return Value:

Returns FSDKE_OK if successful.

FSDK_LoadImageFromHBitmap Function

Loads the image from an HBITMAP handle and provides the internal handle of this image.

C++ Syntax:

```
int FSDK_LoadImageFromHBitmap(HImage* Image, HBITMAP*  
BitmapHandle);
```

Delphi Syntax:

```
function FSDK_LoadImageFromHBitmap(Image: PHImage;  
BitmapHandle: HBitmap): integer;
```

C# Syntax:

```
int FSDK.LoadImageFromHBitmap(ref int Image, IntPtr  
BitmapHandle);
```

VB Syntax:

```
Function FSDKVB_LoadImageFromHBitmap(ByRef Image As Long,  
ByVal BitmapHandle As Integer) As Long
```

CImage Syntax:

```
FSDK.CImage(IntPtr BitmapHandle);
```

Parameters:

Image – pointer to HImage for receiving the loaded image handle.

BitmapHandle – handle of the image to be loaded.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SaveImageToHBitmap Function

Creates an HBITMAP handle containing the image.

C++ Syntax:

```
int FSDK_SaveImageToHBitmap(HImage Image, HBITMAP*
BitmapHandle);
```

Delphi Syntax:

```
function FSDK_SaveImageToHBitmap(Image: HImage; BitmapHandle:
PHBitmap): integer;
```

C# Syntax:

```
int FSDK.SaveImageToHBitmap(int Image, ref IntPtr
BitmapHandle);
```

VB Syntax:

```
Function FSDKVB_SaveImageToHBitmap(ByVal Image As Long, ByRef
BitmapHandle As Integer) As Long
```

CImage Syntax:

```
IntPtr FSDK.CImage.GetHbitmap();
```

Parameters:

Image – internal handle of the image to be saved to HBITMAP.

BitmapHandle – pointer to HBITMAP the created HBITMAP handle will be saved to.

Return Value:

Returns FSDKE_OK if successful.

FSDK.LoadImageFromCLRImage Function

Loads the image from the System.Drawing.Image object and provides the internal handle of this image.

C# Syntax:

```
int FSDK.LoadImageFromCLRImage(ref int Image,
System.Drawing.Image ImageObject);
```

CImage Syntax:

```
FSDK.CImage(System.Drawing.Image ImageObject);
```

Parameters:

Image – reference to HImage for receiving the loaded image handle.

ImageObject – object of the image to be loaded.

Return Value:

Returns FSDKE_OK if successful.

FSDK.SaveImageToCLRImage Function

Creates a System.Drawing.Image object containing the image.

C# Syntax:

```
int FSDK.SaveImageToCLRImage(int Image, ref  
System.Drawing.Image ImageObject);
```

CImage Syntax:

```
System.Drawing.Image FSDK.CImage.ToCLRImage();
```

Parameters:

Image – internal handle of the image to be saved to System.Drawing.Image.

ImageObject – reference to System.Drawing.Image object the image will be saved to.

Return Value:

Returns FSDKE_OK if successful.

FSDK.LoadImageFromAWTImage Function

Loads the image from the java.awt.Image object and provides the internal handle of this image.

Java Syntax:

```
int FSDK.LoadImageFromAWTImage(HImage Image, java.awt.Image  
SourceImage, int ImageMode);
```

Parameters:

Image – HImage for receiving the loaded image.

SourceImage – java.awt.Image object of the image to be loaded.

ImageMode – mode of an image. (See [FSDK LoadImageFromBuffer](#) for more information about image modes.)

Return Value:

Returns FSDKE_OK if successful.

FSDK.SaveImageToAWTImage Function

Creates a java.awt.Image object containing the image.

Java Syntax:

```
int FSDK.SaveImageToAWTImage(HImage Image, java.awt.Image  
DestImage[], int ImageMode);
```

Parameters:

Image – internal handle of the image to be saved to System.Drawing.Image.

DestImage[] – java.awt.Image object the image will be saved to.

ImageMode – desired mode an image will be saved in.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetJpegCompressionQuality

Sets the quality of the JPEG compression to use in the [FSDK_SaveImageToFile](#) function.

C++ Syntax:

```
int FSDK_SetJpegCompressionQuality(int Quality);
```

Delphi Syntax:

```
function FSDK_SetJpegCompressionQuality(Quality: integer):  
integer;
```

C# Syntax:

```
int FSDK.SetJpegCompressionQuality(int Quality);
```

VB Syntax:

```
Function FSDKVB_SetJpegCompressionQuality(ByVal Quality As  
Long) As Long
```

Java and Android Syntax:

```
int FSDK.SetJpegCompressionQuality(int Quality);
```

Parameters:

Quality – quality of JPEG compression. Varies from 0 to 100.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GetImageWidth Function

Returns the width of an image.

C++ Syntax:

```
int FSDK_GetImageWidth(HImage SourceImage, int* Width);
```

Delphi Syntax:

```
function FSDK_GetImageWidth(SourceImage: HImage; Width:  
PInteger): integer;
```

C# Syntax:

```
int FSDK.GetImageWidth(int SourceImage, ref int Width);
```

VB Syntax:

```
Function FSDKVB_GetImageWidth(ByVal SourceImage As Long, ByRef  
Width As Long) As Long
```

Java and Android Syntax:

```
int FSDK.GetImageWidth(HImage SourceImage, int Width[]);
```

CImage Syntax:

```
int FSDK.CImage.Width;
```

Parameters:

SourceImage – internal handle of an image.

Width – pointer to an integer variable to store the width of an image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GetImageHeight Function

Returns the height of an image.

C++ Syntax:

```
int FSDK_GetImageHeight(HImage SourceImage, int* Height);
```

Delphi Syntax:

```
function FSDK_GetImageHeight(SourceImage: HImage; Height: PInteger): integer;
```

C# Syntax:

```
int FSDK.GetImageHeight(int SourceImage, ref int Height);
```

VB Syntax:

```
Function FSDKVB_GetImageHeight(ByVal SourceImage As Long, ByRef Height As Long) As Long
```

Java and Android Syntax:

```
int FSDK.GetImageHeight(HImage SourceImage, int Height[]);
```

CImage Syntax:

```
int FSDK.CImage.Height;
```

Parameters:

SourceImage – internal handle of an image.

Height – pointer to an integer variable to store the height of an image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_CopyImage Function

Creates a copy of an image. The handle of the destination image should be created with the FSDK_CreateEmptyImage function.

C++ Syntax:

```
int FSDK_CopyImage(HImage SourceImage, HImage DestImage);
```

Delphi Syntax:

```
function FSDK_CopyImage(SourceImage: HImage; DestImage:  
HImage): integer;
```

C# Syntax:

```
int FSDK.CopyImage(int SourceImage, int DestImage);
```

VB Syntax:

```
Function FSDKVB_CopyImage (ByVal SourceImage As Long, ByVal  
DestImage As Long) As Long
```

Java and Android Syntax:

```
int FSDK.CopyImage(HImage SourceImage, HImage DestImage);
```

CImage Syntax:

```
FSDK.CImage FSDK.CImage.Copy();
```

Parameters:

SourceImage – handle of an image to be copied.

DestImage – handle of the destination image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_ResizeImage Function

Changes the size of an image. The handle of the destination image should be created with the [FSDK_CreateEmptyImage](#) function.

C++ Syntax:

```
int FSDK_ResizeImage(HImage SourceImage, double ratio, HImage  
DestImage);
```

Delphi Syntax:

```
function FSDK_ResizeImage (SourceImage: HImage; ratio: double;  
DestImage: HImage): integer;
```

C# Syntax:

```
int FSDK.ResizeImage(int SourceImage, double ratio, int  
DestImage);
```

VB Syntax:

```
Function FSDKVB_ResizeImage (ByVal SourceImage As Long, ByVal  
ratio As Double, ByVal DestImage As Long) As Long
```

Java and Android Syntax:

```
int FSDK.ResizeImage(HImage SourceImage, double ratio, HImage  
DestImage);
```

CImage Syntax:

```
FSDK.CImage FSDK.CImage.Resize(double ratio);
```

Parameters:

SourceImage – handle of an image to be resized.

Ratio – factor by which the x and y dimensions of the source image are changed. A factor value greater than 1 corresponds to increasing the image size.

DestImage – handle of the destination image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_RotateImage Function

Rotates an image around its center. The handle of the destination image should be created with the [FSDK_CreateEmptyImage](#) function.

C++ Syntax:

```
int FSDK_RotateImage(HImage SourceImage, double angle, HImage DestImage);
```

Delphi Syntax:

```
function FSDK_RotateImage (SourceImage: HImage; angle: double; DestImage: HImage): integer;
```

C# Syntax:

```
int FSDK.RotateImage (int SourceImage, double angle, int DestImage);
```

VB Syntax:

```
Function FSDKVB_RotateImage (ByVal SourceImage As Long, ByVal angle As Double, ByVal DestImage As Long) As Long
```

Java and Android Syntax:

```
int FSDK.RotateImage(HImage SourceImage, double angle, HImage DestImage);
```

CImage Syntax:

```
FSDK.CImage FSDK.CImage.Rotate(double angle);
```

Parameters:

SourceImage – handle of an image to be rotated.

Angle – rotation angle in degrees.

DestImage – handle of the destination image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_RotateImageCenter Function

Rotates an image around an arbitrary center. The handle of the destination image should be created with the [FSDK_CreateEmptyImage](#) function.

C++ Syntax:

```
int FSDK_RotateImageCenter(HImage SourceImage, double angle,  
double xCenter, double yCenter, HImage DestImage);
```

Delphi Syntax:

```
function FSDK_RotateImageCenter (SourceImage: HImage; angle:  
double; xCenter: double; yCenter: double; DestImage:  
HImage): integer;
```

C# Syntax:

```
int FSDK.RotateImageCenter (int SourceImage, double angle,  
double xCenter, double yCenter, int DestImage);
```

VB Syntax:

```
Function FSDKVB_RotateImageCenter (ByVal SourceImage As Long,  
ByVal angle As Double, ByVal xCenter As Double, ByVal yCenter  
As Double, ByVal DestImage As Long) As Long
```

Java and Android Syntax:

```
int FSDK.RotateImageCenter(HImage SourceImage, double angle,  
double xCenter, double yCenter, HImage DestImage);
```

Parameters:

SourceImage – handle of an image to be rotated.
Angle – rotation angle in degrees.
xCenter – the X coordinate of the rotation center.
yCenter – the Y coordinate of the rotation center.
DestImage – handle of the destination image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_RotateImage90 Function

Rotates the image by 90 or 180 degrees clockwise or counter-clockwise. The handle of the destination image should be created with the [FSDK_CreateEmptyImage](#) function.

C++ Syntax:

```
int FSDK_RotateImage90(HImage SourceImage, int Multiplier,  
HImage DestImage);
```

Delphi Syntax:

```
function FSDK_RotateImage90(SourceImage: HImage; Multiplier:  
integer; DestImage: HImage): integer;
```

C# Syntax:

```
int FSDK.RotateImage90(int SourceImage, int Multiplier, int DestImage);
```

VB Syntax:

```
Function FSDKVB_RotateImage90(ByVal SourceImage As Long, ByVal Multiplier As Long, ByVal DestImage As Long) As Long
```

Java and Android Syntax:

```
int FSDK.RotateImage90(HImage SourceImage, int Multiplier, HImage DestImage);
```

CImage Syntax:

```
FSDK.CImage FSDK.CImage.Rotate90(int Multiplier);
```

Parameters:

SourceImage – handle of an image to be rotated.

Multiplier – an integer multiplier of 90 degrees defining the rotation angle. Specify 1 for 90 degrees clockwise, 2 for 180 degrees clockwise; specify -1 for 90 degrees counter-clockwise.

DestImage – handle of the destination image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_CopyRect Function

Creates a copy of a rectangular area of an image. The handle of the destination image should be created with the [FSDK_CreateEmptyImage](#) function. If some apex of a rectangle is located outside the source image, rectangular areas that do not contain the source image will be black.

C++ Syntax:

```
int FSDK_CopyRect(HImage SourceImage, int x1, int y1, int x2, int y2, HImage DestImage);
```

Delphi Syntax:

```
function FSDK_CopyRect (SourceImage: HImage; x1, y1, x2, y2: integer; DestImage: HImage): integer;
```

C# Syntax:

```
int FSDK.CopyRect (int SourceImage, int x1, int y1, int x2, int y2, int DestImage);
```

VB Syntax:

```
Function FSDKVB_CopyRect (ByVal SourceImage As Long, ByVal x1 As Long, ByVal y1 As Long, ByVal x2 As Long, ByVal y2 As Long, ByVal DestImage As Long) As Long
```

Java and Android Syntax:

```
int FSDK.CopyRect(HImage SourceImage, int x1, int y1, int x2,
int y2, HImage DestImage);
```

CImage Syntax:

```
FSDK.CImage FSDK.CImage.CopyRect(int x1, int y1, int x2, int
y2);
```

Parameters:

SourceImage – handle of an image to copy the rectangle from.

x1 – the X coordinate of the bottom left corner of the copied rectangle.

y1 – the Y coordinate of the bottom left corner of the copied rectangle.

x2 – the X coordinate of the top right corner of the copied rectangle.

y2 – the Y coordinate of the top right corner of the copied rectangle.

DestImage – handle of the destination image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_CopyRectReplicateBorder Function

Creates a copy of a rectangular area of an image and adds replicated border pixels. The handle of the destination image should be created with the [FSDK_CreateEmptyImage](#) function.

This function copies the source image to the destination image and fills pixels ("border") outside the copied area in the destination image with the values of the nearest source image pixels.

C++ Syntax:

```
int FSDK_CopyRectReplicateBorder(HImage SourceImage, int x1,
int y1, int x2, int y2, HImage DestImage);
```

Delphi Syntax:

```
function FSDK_CopyRectReplicateBorder(SourceImage: HImage; x1,
y1, x2, y2: integer; DestImage: HImage): integer;
```

C# Syntax:

```
int FSDK.CopyRectReplicateBorder(int SourceImage, int x1, int
y1, int x2, int y2, int DestImage);
```

VB Syntax:

```
Function FSDKVB_CopyRectReplicateBorder(ByVal SourceImage As
Long, ByVal x1 As Long, ByVal y1 As Long, ByVal x2 As Long,
ByVal y2 As Long, ByVal DestImage As Long) As Long
```

Java and Android Syntax:

```
int FSDK.CopyRectReplicateBorder(HImage SourceImage, int x1,
int y1, int x2, int y2, HImage DestImage);
```


CImage Syntax:

```
FSDK.CImage FSDK.CImage.CopyRectReplicateBorder(int x1, int y1, int x2, int y2);
```

Parameters:

SourceImage – handle of an image to copy the rectangle from.
x1 – the X coordinate of the left bottom corner of the copied rectangle.
y1 – the Y coordinate of the left bottom corner of the copied rectangle.
x2 – the X coordinate of the right top corner of the copied rectangle.
y2 – the Y coordinate of the right top corner of the copied rectangle.
DestImage – handle of the destination image.

Return Value:

Returns FSDKE_OK if successful.

FSDK_MirrorImage Function

Mirrors an image. The function can mirror images horizontally or vertically.

C++ Syntax:

```
int FSDK_MirrorImage(HImage Image, bool UseVerticalMirroringInsteadOfHorizontal);
```

Delphi Syntax:

```
function FSDK_MirrorImage(Image: HImage; UseVerticalMirroringInsteadOfHorizontal: boolean): integer;
```

C# Syntax:

```
int FSDK.MirrorImage(int Image, bool UseVerticalMirroringInsteadOfHorizontal);
```

VB Syntax:

```
Function FSDKVB_MirrorImage(ByVal Image As Long, ByVal UseVerticalMirroringInsteadOfHorizontal As Boolean) As Long
```

Java and Android Syntax:

```
int FSDK.MirrorImage(HImage Image, boolean UseVerticalMirroringInsteadOfHorizontal);
```

CImage Syntax:

```
FSDK.CImage FSDK.CImage.MirrorVertical();  
FSDK.CImage FSDK.CImage.MirrorHorizontal();
```

Parameters:

Image – handle of the image to be mirrored.
UseVerticalMirroringInsteadOfHorizontal – sets the mirror direction.

TRUE: left-to-right swap;

FALSE: top-to-bottom swap;

Return Value:

Returns FSDK_OK if successful.

FSDK_FreeImage Function

Frees the internal representation of an image.

C++ Syntax:

```
int FSDK_FreeImage(HImage Image);
```

Delphi Syntax:

```
function FSDK_FreeImage(Image: HImage): integer;
```

C# Syntax:

```
int FSDK.FreeImage(int Image);
```

VB Syntax:

```
Function FSDKVB_FreeImage(ByVal Image As Long) As Long
```

Java and Android Syntax:

```
int FSDK.FreeImage(HImage Image);
```

Parameters:

Image – handle of the image to be freed.

Return Value:

Returns FSDK_OK if successful.

Face Detection

You can use the [FSDK_DetectFace](#) function to detect a frontal face in an image. The function returns the position of the face in the image. The performance and reliability of face detection is controlled by the [FSDK_SetFaceDetectionParameters](#) and [FSDK_SetFaceDetectionThreshold](#) functions.

Typical parameters for face detection are:

- To detect faces from a webcam in real time, call:

```
FSDK_SetFaceDetectionParameters(false, false, 100);
```

- To reliably detect faces in digital camera photos, call

```
FSDK_SetFaceDetectionParameters(true, false, 500);
```

Face Detection Models

Luxand FaceSDK allows you to switch the internal models used for face detection. You may use the switching to load improved face detection models when made available by Luxand or to switch to a thermal face detection model.

Use the `FSDK_SetParameter` or `FSDK_SetParameters` function to load a face detection model from a file. When using Tracker API, use the `FSDK_SetTrackerParameter` or `FSDK_SetTrackerMultipleParameters` function instead. Set the `FaceDetectionModel` parameter to specify the file to load a model from. Be sure to check the return value of `FSDK_SetParameter` or `FSDK_SetParameters` to confirm the file loaded correctly.

To load a thermal face detection model, set `FaceDetectionModel` to `thermal.bin`. Confirm this file is available in the current directory and, if not, specify the full path to the file.

To switch back to the default model (i.e., the model for visual face detection), set `FaceDetectionModel` to `default`. See the FaceSDK Parameters section for more information.

Face Detection on Thermal Images

Luxand FaceSDK allows for the detection of faces on 8-bit grayscale thermal images. You typically receive such images from a thermal camera, with each pixel representing temperature.

To pass the thermal image to FaceSDK, you may need to convert the temperature values of the image (which may be float or 14-bit values, for example) into 8-bit values (from 0 to 255). The absolute temperature itself is not taken into account by FaceSDK when detecting faces, only the relative difference in temperature between facial features and the background.

Typically, you may normalize an image so that coldest pixel is represented by 0 and the hottest by 255. However, if you have very hot or very cold images in the background, this may lead to faces having a low contrast. Therefore, it is recommended to normalize the image so that 0 would represent the temperature of about 20 degrees Celsius and 255 the temperature of about 40 degrees Celsius (the usual range of temperatures for human faces). After this normalization, any pixels colder than 20 degrees Celsius will have the value 0, and any pixels hotter than 40 degrees Celsius will have the value of 255.

If your thermal camera returns a noisy picture, you may get lower detection rates. In such cases, it is recommended to de-noise the image with a median or a Gaussian filter before passing it to FaceSDK. More information can be found on the links below:

https://en.wikipedia.org/wiki/Median_filter

https://en.wikipedia.org/wiki/Gaussian_blur

Try varying the face detection threshold if you get a high number of false positives or low detection rates on your camera.

FaceSDK itself does not communicate with thermal cameras except when the camera is available as a standard Windows web camera. When working with a thermal camera, you need to use the camera manufacturer's API to receive images. To pass thermal images to FaceSDK, you may use the `FSDK_LoadImageFromBuffer` function after converting the images to an 8-bit format (and possibly normalizing the pixel values, as described above).

To detect faces on thermal images, follow these steps:

1. Load a face detection model by setting the `FaceDetectionModel` with the `FSDK_SetParameter` or `FSDK_SetParameters` function. If using Tracker API, use the

- FSDK_SetTrackerParameter or FSDK_SetTrackerMultipleParameters function instead.
2. Check the return value of the above functions for error to be sure the thermal model was loaded.
 3. Set the TrimOutOfScreenFaces and TrimFacesWithUncertainFacialFeatures parameters to False using the same functions.
 4. Pass thermal images to FSDK_DetectFaces, FSDK_DetectMultipleFaces or FSDK_FeedFrame (when using Tracker API).

Example:

```
FSDK_SetParameters("FaceDetectionModel=thermal.bin;TrimOutOfScreenFaces=false;TrimFacesWithUncertainFacialFeatures=false",
&err);
```

Refer to the Thermal sample application to see how faces on thermal images can be detected.

Data types

Luxand FaceSDK introduces the TFacePosition data type that stores the information about the position of the face. The xc and yc fields specifies the X and Y coordinates of the center of the face, w specifies the width of the face, and angle specifies the in-plane rotation angle of the face in degrees.

C++ Declaration:

```
typedef struct {
    int xc, yc, w;
    int padding;
    double angle;
} TFacePosition;
```

C# Declaration:

```
public struct TFacePosition {
    public int xc, yc, w;
    public double angle;
}
```

Delphi Declaration:

```
TFacePosition = record
    xc, yc, w: integer;
    padding: integer;
    angle: double;
end;
PFacePosition = ^TFacePosition;
```

VB Declaration:

```
Type TFacePosition
    xc As Long
    yc As Long
    w As Long
```

```
padding as Long
angle As Double
End Type
```

Java Declaration:

The class TFacePosition contains the following fields:

```
public int xc, yc, w;
public double angle;
```

The class TFaces encapsulates an array of TFacePosition classes. It has the following properties:

```
public TFacePosition faces[];
int maxFaces;
```

FSDK_DetectFace Function

Detects a frontal face in an image and stores information about the face position into the TFacePosition structure.

C++ Syntax:

```
int FSDK_DetectFace(HImage Image, TFacePosition*
FacePosition);
```

Delphi Syntax:

```
function FSDK_DetectFace(Image: HImage; FacePosition:
PFacePosition): integer;
```

C# Syntax:

```
int FSDK.DetectFace(int Image, ref FSDK.TFacePosition
FacePosition);
```

VB Syntax:

```
Function FSDKVB_DetectFace(ByVal Image As Long, ByRef
FacePosition As TFacePosition) As Long
```

Java Syntax:

```
int FSDK.DetectFace(HImage Image, TFacePosition.ByReference
FacePosition);
```

Android Syntax:

```
int FSDK.DetectFace(HImage Image, TFacePosition FacePosition);
```

CImage Syntax:

```
FSDK.TFacePosition FSDK.CImage.DetectFace();
```

Parameters:

Image – handle of the image to detect the face in.

FacePosition – pointer to the TFacePosition structure to store information about the face position.

Return Value:

Returns FSDK_OK if successful. If a face is not found, the function returns the FSDK_FACE_NOT_FOUND code. If the input image is too small (less than 20x20 pixels), the functions returns FSDK_IMAGE_TOO_SMALL.

Example

```
int img1;
TFacePosition FacePosition;

FSDK_Initialize("");
FSDK_LoadImageFromFile(&img1, "test.jpg");
FSDK_DetectFace(img1, &FacePosition);

printf("face position: %d %d %d", FacePosition.xc,
FacePosition.yc, FacePosition.angle);
```

FSDK_DetectMultipleFaces Function

Detects multiple faces in an image.

C++ Syntax:

```
int FSDK_DetectMultipleFaces(HImage Image, int* DetectedCount,
TFacePosition* FaceArray, int MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_DetectMultipleFaces(Image: HImage;
DetectedCount: PInteger; FaceArray: PFacePositionArray;
MaxSizeInBytes: integer): integer;
```

C# Syntax:

```
int FSDK.DetectMultipleFaces(int Image, ref int DetectedCount,
out FSDK.TFacePosition[] FaceArray, int MaxSizeInBytes);
```

VB Syntax:

```
Function FSDKVB_DetectMultipleFaces(ByVal Image As Long, ByRef
DetectedCount As Long, ByRef FaceArray As TFacePosition, ByVal
MaxSizeInBytes As Long) As Long
```

Java and Android Syntax:

```
int FSDK.DetectMultipleFaces(HImage Image, TFaces FaceArray);
```

CImage Syntax:

```
FSDK.TFacePosition[] FSDK.CImage.DetectMultipleFaces();
```

Parameters:

Image – handle of the image to detect faces in.

DetectedCount – count of the faces found in the image.

FaceArray – pointer to the array of TFacePosition structure to store the information about the detected faces.

MaxSizeInBytes – size of the FaceArray buffer in bytes. The function will not store more than MaxSize bytes in the buffer.

Return Value:

Returns FSDK_OK if successful. If no faces are found, the function returns the FSDK_FACE_NOT_FOUND code. If the input image is too small (less than 20x20 pixels), the functions returns FSDK_IMAGE_TOO_SMALL.

Example

```
int img1;
int DetectedCount;
TFacePosition FaceArray[50];

FSDK_Initialize("");
FSDK_LoadImageFromFile(&img1, "test.jpg");
FSDK_DetectMultipleFaces(img1, &DetectedCount , FaceArray,
sizeof(FaceArray));

for (i = 0; i < DetectedCount; i++) {
    printf("face position: %d %d %d\n", FaceArray[i].xc,
FaceArray[i].yc, FaceArray[i].angle);
}
```

FSDK_SetFaceDetectionParameters Function

Allows setting a number of face detection parameters to control the performance and reliability of face detector.

The function allows configuring the following parameters: HandleArbitraryRotations, DetermineFaceRotationAngle and InternalResizeWidth.

HandleArbitraryRotations, DetermineFaceRotationAngle can be TRUE or FALSE, while InternalResizeWidth is an integer.

Other face detection parameters that can also be set using the FSDK_SetParameter or FSDK_SetParameters function.

C++ Syntax:

```
int FSDK_SetFaceDetectionParameters(bool
HandleArbitraryRotations, bool DetermineFaceRotationAngle, int
InternalResizeWidth);
```

Delphi Syntax:

```
function
FSDK_SetFaceDetectionParameters(HandleArbitraryRotations:
boolean; DetermineFaceRotationAngle: boolean;
InternalResizeWidth: integer): integer;
```

C# Syntax:

```
int FSDK.SetFaceDetectionParameters(bool
HandleArbitraryRotations, bool DetermineFaceRotationAngle, int
InternalResizeWidth);
```

VB Syntax:

```
Function FSDKVB_SetFaceDetectionParameters (ByVal  
HandleArbitraryRotations As Boolean, ByVal  
DetermineFaceRotationAngle As Boolean, ByVal  
InternalResizeWidth As Long) As Long
```

Java and Android Syntax:

```
int FSDK.SetFaceDetectionParameters(boolean  
HandleArbitraryRotations, boolean DetermineFaceRotationAngle,  
int InternalResizeWidth);
```

Parameters:

HandleArbitraryRotations— extends default in-plane face rotation angle from -15..15 degrees to -30..30 degrees.

TRUE: extended in-plane rotation support is enabled at the cost of detection speed (3 times performance hit).

FALSE: default fast detection -15..15 degrees.

DetermineFaceRotationAngle— enables or disables the detection of in-plane face rotation angle.

TRUE: detects in-plane rotation angle when detecting faces. The angle is recorded into the Angle field of the TFacePosition structure (TFacePosition is a structure returned by [FSDK_DetectFace](#) and [FSDK_DetectMultipleFaces](#)).

FALSE: disables the detection of rotation angle.

Note: Enabling face rotation angle detection slows down the detection process slightly. Set this parameter to TRUE if you are planning to call [FSDK_DetectFacialFeatures](#) or [FSDK_DetectFacialFeaturesInRegion](#).

InternalResizeWidth – controls the detection speed by setting the size of the image the detection functions will work with. Choose higher value to increase detection quality, or lower value to improve the performance.

Note: By default, all images are internally resized to the width of 384 pixels. 384 pixels are a reasonable compromise between performance and detection quality. While large images are down-sized, the smaller ones are up-sized to the specified Resize Width in order to maintain constant detection speed.

Choosing the right value for InternalResizeWidth

Choosing the correct value for the InternalResizeWidth parameter is essential for the correct operation of face detection functions of the SDK. The face detection functions can only detect faces as small as 20x20 pixels. Even if the source image is a large 1000x1000 dots one, the face on that image can be as small as 100x100 pixels. If you set InternalResizeWidth to 200, then the source image will be resized to 200x200 pixels, thus the face will only occupy 20x20 pixels. This is still enough for the SDK functions to work. If, however, you set InternalResizeWidth to 100, then the original image will become 100x100 pixels, and the face on it will only occupy 10x10 dots, which is NOT enough for the SDK functions to work with. Be extra careful when changing the default value of InternalResizeWidth. For example, webcam images can be usually detected with InternalResizeWidth set to 100, while images from multi-megapixel digital cameras require values of at least 384 or 512 pixels to work with.

Return Value:

Returns FSDK_OK if successful.

FSDK_SetFaceDetectionThreshold Function

Sets a threshold value for face detection. The default value is 5. The lowest possible value is 1.

The function allows adjusting the sensitivity of the detection. If the threshold value is set to a higher value, the detector will only recognize faces with sharp, clearly defined details, thus reducing the number of false positive detections. Setting the threshold lower allows detecting more faces with less clearly defined features at the expense of increased number of false positives.

C++ Syntax:

```
int FSDK_SetFaceDetectionThreshold(int Threshold);
```

Delphi Syntax:

```
function FSDK_SetFaceDetectionThreshold(Threshold: integer): integer;
```

C# Syntax:

```
int FSDK.SetFaceDetectionThreshold(int Threshold);
```

VB Syntax:

```
Function FSDKVB_SetFaceDetectionThreshold(ByVal Threshold As Long) As Long
```

Java and Android Syntax:

```
int FSDK.SetFaceDetectionThreshold(int Threshold);
```

Parameters:

Threshold – Threshold value.

Return Value:

Returns FSDK_OK if successful.

Facial Feature Detection

FaceSDK provides the [FSDK_DetectFacialFeatures](#) function to detect facial features in an image and the [FSDK_DetectEyes](#) function to detect just eye centers in an image. First, these functions detect a frontal face in an image, and then detect its facial features or only eye centers. The [FSDK_DetectFacialFeaturesInRegion](#) and [FSDK_DetectEyesInRegion](#) functions do not perform the face detection step and detect facial features or eye centers in a region returned by [FSDK_DetectFace](#) or [FSDK_DetectMultipleFaces](#).

In the current version of Luxand FaceSDK the performance of [FSDK_DetectEyes](#) and [FSDK_DetectFacialFeatures](#) is the same, so there is no advantage in calling [FSDK_DetectEyes](#) instead of [FSDK_DetectFacialFeatures](#).

The facial features are stored in the FSDK_Features data structure. FSDK_Features is an array data type containing FSDK_FACIAL_FEATURE_COUNT points. The list of facial features recognized by FaceSDK is available in the [Detected Facial Features](#) chapter.

Eye centers are saved to FSDK_Features[0] and FSDK_Features[1]. The [FSDK_DetectEyes](#) and [FSDK_DetectEyesInRegion](#) functions do not change other elements of the FSDK_Features array.

C++ Declaration:

```
typedef struct { int x,y; } TPoint;
typedef TPoint FSDK_Features [FSDK_FACIAL_FEATURE_COUNT];
```

C# Declaration:

```
public struct TPoint {
    public int x, y;
}
```

Delphi Declaration:

```
TPoint = record
    x, y: integer;
end;
FSDK_Features = array[0..FSDK_FACIAL_FEATURE_COUNT - 1] of
TPoint;
PFSDK_Features = ^FSDK_Features;
```

VB Declaration:

```
Type TPoint
    x As Long
    y As Long
End Type
```

Java and Android Declaration:

The class TPoint has the following properties:

```
public int x, y;
```

The class FSDK_Features has the following property:

```
public TPoint features[];
```

FSDK_DetectFacialFeatures Function

Detects a frontal face in an image and detects its facial features.

C++ Syntax:

```
int FSDK_DetectFacialFeatures(HImage Image, FSDK_Features*
FacialFeatures);
```

Delphi Syntax:

```
function FSDK_DetectFacialFeatures(Image: HImage;
FacialFeatures: PFSDK_Features): integer;
```

C# Syntax:

```
int FSDK.DetectFacialFeatures(int Image, out FSDK.TPoint[] FacialFeatures);
```

VB Syntax:

```
Function FSDKVB_DetectFacialFeatures(ByVal Image As Long, ByRef FacialFeatures As TPoint) As Long
```

Java Syntax:

```
int FSDK.DetectFacialFeatures(HImage Image, FSDK_Features.ByReference FacialFeatures);
```

Android Syntax:

```
int FSDK.DetectFacialFeatures(HImage Image, FSDK_Features FacialFeatures);
```

CImage Syntax:

```
FSDK.TPoint[] FSDK.CImage.DetectFacialFeatures();
```

Parameters:

Image— handle of the image facial features should be detected in.

FacialFeatures— pointer to the FSDK_Features array for receiving the detected facial features.

Return Value:

Returns FSDKE_OK if successful.

Example

```
int img1;
FSDK_Features Features;

FSDK_Initialize("");
FSDK_LoadImageFromFile(&img1, "test.jpg");
FSDK_DetectFacialFeatures(img1, Features);

printf("Left eye location: (%d, %d)\n",
Features[FSDKP_LEFT_EYE].x, Features[FSDKP_LEFT_EYE].y);
printf("Right eye location: (%d, %d)\n",
Features[FSDKP_RIGHT_EYE].x, Features[FSDKP_RIGHT_EYE].y);
```

FSDK_DetectFacialFeaturesInRegion Function

Detects facial features in an image region returned by [FSDK_DetectFace](#) or [FSDK_DetectMultipleFaces](#). This function can be useful if an approximate face size is known, or to detect facial features of a specific face returned by [FSDK_DetectMultipleFaces](#).

C++ Syntax:

```
int FSDK_DetectFacialFeaturesInRegion(HImage Image, TFacePosition* FacePosition, FSDK_Features* FacialFeatures);
```

Delphi Syntax:

```
function FSDK_DetectFacialFeaturesInRegion(Image: HImage;
FacePosition: PFacePosition; FacialFeatures: PFSDK_Features):
integer;
```

C# Syntax:

```
int FSDK.DetectFacialFeaturesInRegion(int Image, ref
FSDK.TFacePosition FacePosition, out FSDK.TPoint[]
FacialFeatures);
```

VB Syntax:

```
Function FSDKVB_DetectFacialFeaturesInRegion(ByVal Image As
Long, ByRef FacePosition As TFacePosition, ByRef
FacialFeatures As TPoint) As Long
```

Java Syntax:

```
int FSDK.DetectFacialFeaturesInRegion(HImage Image,
TFacePosition FacePosition, FSDK_Features.ByReference
FacialFeatures);
```

Android Syntax:

```
int FSDK.DetectFacialFeaturesInRegion(HImage Image,
TFacePosition FacePosition, FSDK_Features FacialFeatures);
```

C# Syntax:

```
FSDK.TPoint[] FSDK.CImage.DetectFacialFeaturesInRegion(ref
FSDK.TFacePosition FacePosition);
```

Parameters:

Image – handle of the image facial features should be detected in.

FacePosition – pointer to the face position structure.

FacialFeatures – pointer to the FSDK_Features array for receiving the detected facial features.

Return Value:

Returns FSDKE_OK if successful.

Example

```
int i, DetectedCount, img1;
FSDK_Features Features;
TFacePosition FaceArray[50];

FSDK_Initialize("");
FSDK_LoadImageFromFile(&img1, "test.jpg");

FSDK_DetectMultipleFaces(img1, &DetectedCount , FaceArray,
sizeof(FaceArray));

for (i = 0; i < DetectedCount; i++) {
```

```

    FSDK_DetectFacialFeaturesInRegion(img1, FaceArray[i],
    Features);
    printf("Left eye location: (%d, %d)\n",
    Features[FSDKP_LEFT_EYE].x, Features[FSDKP_LEFT_EYE].y);
    printf("Right eye location: (%d, %d)\n",
    Features[FSDKP_RIGHT_EYE].x, Features[FSDKP_RIGHT_EYE].y);
}

```

FSDK_DetectEyes Function

Detects a frontal face in an image and detects its eye centers.

C++ Syntax:

```

int FSDK_DetectEyes(HImage Image, FSDK_Features*
FacialFeatures);

```

Delphi Syntax:

```

function FSDK_DetectEyes(Image: HImage; FacialFeatures:
PFSDK_Features): integer;

```

C# Syntax:

```

int FSDK.DetectEyes(int Image, out FSDK.TPoint[]
FacialFeatures);

```

VB Syntax:

```

Function FSDKVB_DetectEyes(ByVal Image As Long, ByRef
FacialFeatures As TPoint) As Long

```

Java Syntax:

```

int FSDK.DetectEyes(HImage Image, FSDK_Features.ByReference
FacialFeatures);

```

Android Syntax:

```

int FSDK.DetectEyes(HImage Image, FSDK_Features
FacialFeatures);

```

CImage Syntax:

```

FSDK.TPoint[] FSDK.CImage.DetectEyes();

```

Parameters:

Image – handle of the image eye centers should be detected in.

FacialFeatures – pointer to the FSDK_Features array for receiving the detected eye centers.

Return Value:

Returns FSDKE_OK if successful.

FSDK_DetectEyesInRegion Function

Detects eye centers in an image region returned by [FSDK_DetectFace](#) or [FSDK_DetectMultipleFaces](#).

C++ Syntax:

```
int FSDK_DetectEyesInRegion(HImage Image, TFacePosition*  
FacePosition, FSDK_Features* FacialFeatures);
```

Delphi Syntax:

```
function FSDK_DetectEyesInRegion(Image: HImage; FacePosition:  
PFacePosition; FacialFeatures: PFSDK_Features): integer;
```

C# Syntax:

```
int FSDK.DetectEyesInRegion(int Image, ref FSDK.TFacePosition  
FacePosition, out FSDK.TPoint[] FacialFeatures);
```

VB Syntax:

```
Function FSDKVB_DetectEyesInRegion(ByVal Image As Long, ByRef  
FacePosition As TFacePosition, ByRef FacialFeatures As TPoint)  
As Long
```

Java Syntax:

```
int FSDK.DetectEyesInRegion(HImage Image, TFacePosition  
FacePosition, FSDK_Features.ByReference FacialFeatures);
```

Android Syntax:

```
int FSDK.DetectEyesInRegion(HImage Image, TFacePosition  
FacePosition, FSDK_Features FacialFeatures);
```

CImage Syntax:

```
FSDK.TPoint[] FSDK.CImage.DetectEyesInRegion(ref  
FSDK.TFacePosition FacePosition);
```

Parameters:

Image – handle of the image eye centers should be detected in.

FacePosition – pointer to the face position structure.

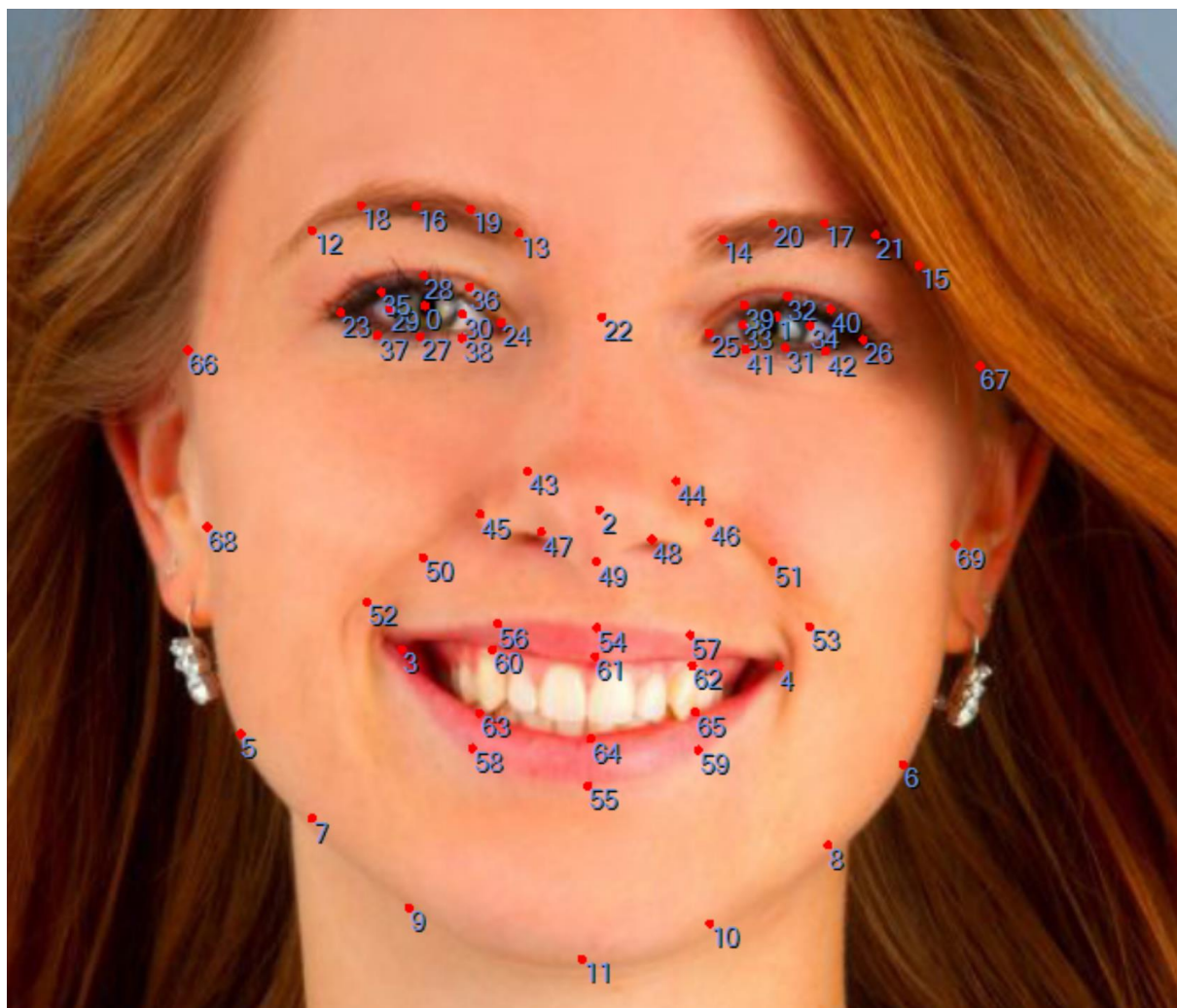
FacialFeatures – pointer to the FSDK_Features array for receiving the detected eye centers.

Return Value:

Returns FSDKE_OK if successful.

Detected Facial Features

Luxand FaceSDK detects 70 facial feature points. These facial feature points can be accessed by their names in the FSDK_Features array.



Facial Feature Name	Value
FSDKP_LEFT_EYE	0
FSDKP_RIGHT_EYE	1
FSDKP_LEFT_EYE_INNER_CORNER	24
FSDKP_LEFT_EYE_OUTER_CORNER	23
FSDKP_LEFT_EYE_LOWER_LINE1	38
FSDKP_LEFT_EYE_LOWER_LINE2	27
FSDKP_LEFT_EYE_LOWER_LINE3	37
FSDKP_LEFT_EYE_UPPER_LINE1	35
FSDKP_LEFT_EYE_UPPER_LINE2	28

FSDKP_LEFT_EYE_UPPER_LINE3	36
FSDKP_LEFT_EYE_LEFT_IRIS_CORNER	29
FSDKP_LEFT_EYE_RIGHT_IRIS_CORNER	30
FSDKP_RIGHT_EYE_INNER_CORNER	25
FSDKP_RIGHT_EYE_OUTER_CORNER	26
FSDKP_RIGHT_EYE_LOWER_LINE1	41
FSDKP_RIGHT_EYE_LOWER_LINE2	31
FSDKP_RIGHT_EYE_LOWER_LINE3	42
FSDKP_RIGHT_EYE_UPPER_LINE1	40
FSDKP_RIGHT_EYE_UPPER_LINE2	32
FSDKP_RIGHT_EYE_UPPER_LINE3	39
FSDKP_RIGHT_EYE_LEFT_IRIS_CORNER	33
FSDKP_RIGHT_EYE_RIGHT_IRIS_CORNER	34
FSDKP_LEFT_EYEBROW_INNER_CORNER	13
FSDKP_LEFT_EYEBROW_MIDDLE	16
FSDKP_LEFT_EYEBROW_MIDDLE_LEFT	18
FSDKP_LEFT_EYEBROW_MIDDLE_RIGHT	19
FSDKP_LEFT_EYEBROW_OUTER_CORNER	12
FSDKP_RIGHT_EYEBROW_INNER_CORNER	14
FSDKP_RIGHT_EYEBROW_MIDDLE	17
FSDKP_RIGHT_EYEBROW_MIDDLE_LEFT	20
FSDKP_RIGHT_EYEBROW_MIDDLE_RIGHT	21
FSDKP_RIGHT_EYEBROW_OUTER_CORNER	15
FSDKP_NOSE_TIP	2
FSDKP_NOSE_BOTTOM	49
FSDKP_NOSE_BRIDGE	22
FSDKP_NOSE_LEFT_WING	43
FSDKP_NOSE_LEFT_WING_OUTER	45
FSDKP_NOSE_LEFT_WING_LOWER	47
FSDKP_NOSE_RIGHT_WING	44

FSDKP_NOSE_RIGHT_WING_OUTER	46
FSDKP_NOSE_RIGHT_WING_LOWER	48
FSDKP_MOUTH_RIGHT_CORNER	3
FSDKP_MOUTH_LEFT_CORNER	4
FSDKP_MOUTH_TOP	54
FSDKP_MOUTH_TOP_INNER	61
FSDKP_MOUTH_BOTTOM	55
FSDKP_MOUTH_BOTTOM_INNER	64
FSDKP_MOUTH_LEFT_TOP	56
FSDKP_MOUTH_LEFT_TOP_INNER	60
FSDKP_MOUTH_RIGHT_TOP	57
FSDKP_MOUTH_RIGHT_TOP_INNER	62
FSDKP_MOUTH_LEFT_BOTTOM	58
FSDKP_MOUTH_LEFT_BOTTOM_INNER	63
FSDKP_MOUTH_RIGHT_BOTTOM	59
FSDKP_MOUTH_RIGHT_BOTTOM_INNER	65
FSDKP_NASOLABIAL_FOLD_LEFT_UPPER	50
FSDKP_NASOLABIAL_FOLD_LEFT_LOWER	52
FSDKP_NASOLABIAL_FOLD_RIGHT_UPPER	51
FSDKP_NASOLABIAL_FOLD_RIGHT_LOWER	53
FSDKP_CHIN_BOTTOM	11
FSDKP_CHIN_LEFT	9
FSDKP_CHIN_RIGHT	10
FSDKP_FACE_CONTOUR1	7
FSDKP_FACE_CONTOUR2	5
FSDKP_FACE_CONTOUR12	6
FSDKP_FACE_CONTOUR13	8
FSDKP_FACE_CONTOUR14	66
FSDKP_FACE_CONTOUR15	67
FSDKP_FACE_CONTOUR16	68

Mask-on Face Detection

To detect faces covered by masks, you need to adjust the settings of several parameters. You also need to download a model file trained specifically on masked faces (for visual face detection) and put it into the working directory of your application:

https://luxand.com/download/fd_masks1.bin

If you are using `FSDK_DetectFace` or `FSDK_DetectMultipleFaces` to detect faces (i.e. you don't use Tracker API), use the following code to set the recommended face detection parameters and use the latest model file:

```
int err = 0;
FSDK_SetFaceDetectionParameters(true, false, 1024);
FSDK_SetFaceDetectionThreshold(5);

if (FSDKE_OK !=
FSDK_SetParameters("FaceDetectionModel=fd_masks1.bin;TrimFaces
WithUncertainFacialFeatures=false",
&err))
{
    fprintf(stderr, "Error loading face detection model!\n");
    exit(3);
}
```

Alternatively, if you are using Tracker API, use the following calls:

```
int err = 0;
FSDK_SetTrackerMultipleParameters(tracker,
"RecognizeFaces=false; HandleArbitraryRotations=true;
DetermineFaceRotationAngle=false; InternalResizeWidth=1024;
FaceDetectionThreshold=5;", &err);

if (FSDKE_OK != FSDK_SetTrackerMultipleParameters(tracker,
"FaceDetectionModel=fd_masks1.bin;TrimFacesWithUncertainFacial
Features=false",
&err))
{
    fprintf(stderr, "Error loading face detection model!\n");
    exit(3);
}
```

In the code above, the `TrimFacesWithUncertainFacialFeatures` parameter is set to `false`. When it is set to `true`, faces with uncertain facial features are removed from the detection result. As masks may cover many facial features, this setting was preventing such faces from being detected. You can learn more about this parameter here:

[Configuration](#)

We don't recommend that you use face matching when this parameter is set to `false`, since it will give a higher amount of false acceptances. We plan to update our face recognition models so they support mask-on face matching soon.

We recommend setting the `InternalResizeWidth` parameter to 1024 so the faces that are far from the camera are detected. If you don't expect your faces to be that far, you can lower the parameter value to 512 or 256 to increase the speed of detection.

Face Matching

Luxand FaceSDK provides the API to extract face templates and match them. A template extracted from a face can be stored in a database and can then be used to match faces using the [FSDK_MatchFaces](#) function.

To extract a face template, use the [FSDK_GetFaceTemplate](#), [FSDK_GetFaceTemplateInRegion](#) or `FSDK_GetFaceTemplateUsingFeatures` functions. The [FSDK_MatchFaces](#) function returns the facial similarity level. You may consider similarity to be equal to the probability that templates belong to the same person.

More precisely: if the access control system provides access to a person when similarity is higher of threshold x , the possibility of providing erroneous access to another person is $1-x$. For example, if the decision to provide access to a person is based on the code

```
if (similarity > 0.99)
    AllowAccess();
```

the possibility of erroneous access to another person is 0.01, or 1%.

A facial template contains data that describes the face. There is no direct way to re-create the original face image from a template. However, when using Tracker API, it may store the original facial images in Tracker memory (see the [Storing original facial images](#) section).

To determine if the matched templates belong to the same person (with a specified error possibility), you can compare the facial similarity value with a threshold calculated by the [FSDK_GetMatchingThresholdAtFAR](#) or [FSDK_GetMatchingThresholdAtFRR](#) functions.

Note: it is recommended to retain both the original face images and their templates in the database. This is because future versions of Luxand FaceSDK may offer an improved template extraction algorithm, together with changes to the template format. If you are using Tracker API, there is an option to convert its memory automatically if the template format changes (see the [Storing original facial images](#) section).

A face template is stored in the `FSDK_FaceTemplate` data structure.

In .NET, there is no specific data type for a template. Instead, it is stored in an array of bytes of `FSDK.TemplateSize` length. Below is an example of retrieving facial template in C#.

C# Example:

```
templateData = new byte[FSDK.TemplateSize];
FSDK.GetFaceTemplate(imageHandle, out templateData);
```

C++ Declaration:

```
typedef struct {
    char ftemplate [1040];
} FSDK_FaceTemplate;
```

Delphi Declaration:

```
FSDK_FaceTemplate = record
    Template: array[0.. 1040-1] of byte;
end;
PFSDK_FaceTemplate = ^FSDK_FaceTemplate;
```

Java and Android Declaration:

The class `FSDK_FaceTemplate` has the following property:

```
public byte template[];
```

VB Declaration:

```
Public Type FSDK_FaceTemplate
    FaceTemplate(1040) as Byte
End Type
```

FSDK_GetFaceTemplate Function

This function is used to extract a template from a facial image. The function first detects a face, then detects its facial features and extracts the template.

If there is more than one face in the image, the template is extracted for the face with the most clearly visible details. If there is no clearly visible face, the function returns an error code. To set the threshold determining the accepted quality for faces, use the `FSDK_SetFaceDetectionThreshold` function.

If the face position or its features or eye centers are known, it is more efficient to use the [FSDK_GetFaceTemplateInRegion](#) or [FSDK_GetFaceTemplateUsingEyes](#) functions. To extract the template for a specific face, use the [FSDK_GetFaceTemplateInRegion](#) function.

C++ Syntax:

```
int FSDK_GetFaceTemplate(HImage Image, FSDK_FaceTemplate*
    FaceTemplate);
```

Delphi Syntax:

```
function FSDK_GetFaceTemplate(Image: HImage; FaceTemplate:
    PFSDK_FaceTemplate): integer;
```

C# Syntax:

```
int FSDK.GetFaceTemplate(int Image, out byte[] FaceTemplate);
```

VB Syntax:

```
Function FSDKVB_GetFaceTemplate(ByVal Image As Long, ByRef
    FaceTemplate As Byte) As Long
```

Java Syntax:

```
int FSDK.GetFaceTemplate(HImage Image,
    FSDK_FaceTemplate.ByReference FaceTemplate);
```

Android Syntax:

```
int FSDK.GetFaceTemplate(HImage Image, FSDK_FaceTemplate FaceTemplate);
```

CImage Syntax:

```
byte[] FSDK.CImage.GetFaceTemplate();
```

Parameters:

Image— handle of the image from which to extract the face template.

FaceTemplate – pointer to the FSDK_FaceTemplate structure, used to receive the face template.

Return Value:

Returns FSDKE_OK if successful. If no faces are found, or the quality of the image is not sufficient, the function returns the FSDKE_FACE_NOT_FOUND code.

FSDK_GetFaceTemplateInRegion Function

Extracts a template for a face located in a specific region returned by FSDK_DetectFace or FSDK_DetectMultipleFaces.

The function detects facial features in a specific region and extracts a template. The face detection stage is not performed. This function can be useful if an approximate face size and position is known, or to process a specific face returned by FSDK_DetectFace or FSDK_DetectMultipleFaces. The function returns no error if the face is not clearly visible. This is because it assumes that if face detection functions return a detected face position, the face is of sufficient quality.

If facial features or eye centers are known, it is more efficient to use the [FSDK_GetFaceTemplateUsingFeatures](#) or [FSDK_GetFaceTemplateUsingEyes](#) function.

C++ Syntax:

```
int FSDK_GetFaceTemplateInRegion(HImage Image, TFacePosition* FacePosition, FSDK_FaceTemplate* FaceTemplate);
```

Delphi Syntax:

```
function FSDK_GetFaceTemplateInRegion (Image: HImage;  
FacePosition: PFacePosition; FaceTemplate:  
PFSDK_FaceTemplate): integer;
```

C# Syntax:

```
int FSDK.GetFaceTemplateInRegion(int Image, ref  
FSDK.TFacePosition FacePosition, out byte[] FaceTemplate);
```

VB Syntax:

```
Function FSDKVB_GetFaceTemplateInRegion(ByVal Image As Long,  
ByRef FacePosition As TFacePosition, ByRef FaceTemplate As  
Byte) As Long
```

Java Syntax:

```
int FSDK.GetFaceTemplateInRegion(HImage Image, TFacePosition FacePosition, FSDK_FaceTemplate.ByReference FaceTemplate);
```

Android Syntax:

```
int FSDK.GetFaceTemplateInRegion(HImage Image, TFacePosition FacePosition, FSDK_FaceTemplate FaceTemplate);
```

CImage Syntax:

```
byte[] FSDK.CImage.GetFaceTemplateInRegion(ref FSDK.TFacePosition FacePosition);
```

Parameters:

Image – handle of the image from which to extract the face template.

FacePosition – pointer to the face position structure.

FaceTemplate – pointer to the FSDK_FaceTemplate structure, used to receive the face template.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GetFaceTemplateUsingEyes Function

Extracts a face template using the detected eye centers.

The function receives eye centers coordinates detected by the [FSDK_DetectFacialFeatures](#), [FSDK_DetectFacialFeaturesInRegion](#), [FSDK_DetectEyes](#) or [FSDK_DetectEyesInRegion](#) functions and extracts a face template. Face detection, facial feature detection, and eye centers detection are not performed. This function can be useful when facial features or eye centers for a specific face are already detected. The function returns no error if the face is not clearly visible, since it assumes that if the face and its facial features or eye centers are already detected, the face is of sufficient quality.

Note that the FSDK_GetFaceTemplate, FSDK_GetFaceTemplateInRegion and FSDK_GetFaceTemplateUsingFeatures functions return templates that could be matched with higher accuracy, so it is recommended to use these functions instead.

C++ Syntax:

```
int FSDK_GetFaceTemplateUsingEyes(HImage Image, FSDK_Features* eyeCoords, FSDK_FaceTemplate* FaceTemplate);
```

Delphi Syntax:

```
function FSDK_GetFaceTemplateUsingEyes(Image: HImage;  
eyeCoords: PFSDK_Features; FaceTemplate: PFSDK_FaceTemplate):  
integer;
```

C# Syntax:

```
int FSDK.GetFaceTemplateUsingEyes(int Image, ref FSDK.TPoint[] eyeCoords, out byte[] FaceTemplate);
```

VB Syntax:

```
Function FSDKVB_GetFaceTemplateUsingEyes(ByVal Image As Long,  
ByRef eyeCoords As TPoint, ByRef FaceTemplate As Byte) As Long
```

Java Syntax:

```
int FSDK.GetFaceTemplateUsingEyes(HImage Image, FSDK_Features  
eyeCoords, FSDK_FaceTemplate.ByReference FaceTemplate);
```

Android Syntax:

```
int FSDK.GetFaceTemplateUsingEyes(HImage Image, FSDK_Features  
eyeCoords, FSDK_FaceTemplate FaceTemplate);
```

CImage Syntax:

```
byte[] FSDK.CImage.GetFaceTemplateUsingEyes(ref FSDK.TPoint[]  
eyeCoords);
```

Parameters:

Image – handle of the image to extract the face template from.

eyeCoords – pointer to the FSDK_Features array containing eye centers coordinates.

FaceTemplate – pointer to the FSDK_FaceTemplate structure for receiving the face template.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GetFaceTemplateUsingFeatures Function

Extracts a face template using the detected facial feature coordinates.

The function receives facial feature coordinates detected by the [FSDK_DetectFacialFeatures](#) or [FSDK_DetectFacialFeaturesInRegion](#) functions and extracts a face template. Face detection, facial feature detection, and eye centers detection are not performed. This function can be useful when facial features for a specific face are already detected. The function produces no error if the face is not clearly visible, since it assumes that if the face and its facial features are already detected, the face is of sufficient quality.

The function determines if facial features, starting with the 2nd, are equal to zero or uninitialized. In this case, the functions calls [FSDK_GetFaceTemplateUsingEyes](#) instead.

C++ Syntax:

```
int FSDK_GetFaceTemplateUsingFeatures(HImage Image,  
FSDK_Features* FacialFeatures, FSDK_FaceTemplate*  
FaceTemplate);
```

Delphi Syntax:

```
function FSDK_GetFaceTemplateUsingFeatures(Image: HImage;  
FacialFeatures: PFSDK_Features; FaceTemplate:  
PFSDK_FaceTemplate): integer;
```

C# Syntax:

```
int FSDK.GetFaceTemplateUsingFeatures(int Image, ref
FSDK.TPoint[] FacialFeatures, out byte[] FaceTemplate);
```

VB Syntax:

```
Function FSDKVB_GetFaceTemplateUsingFeatures (ByVal Image As
Long, ByRef FacialFeaturesAs TPoint, ByRef FaceTemplate As
Byte) As Long
```

Java Syntax:

```
int FSDK.GetFaceTemplateUsingFeatures(HImage Image,
FSDK_Features FacialFeatures, FSDK_FaceTemplate.ByReference
FaceTemplate);
```

Android Syntax:

```
int FSDK.GetFaceTemplateUsingFeatures(HImage Image,
FSDK_Features FacialFeatures, FSDK_FaceTemplate FaceTemplate);
```

CImage Syntax:

```
byte[] FSDK.CImage.GetFaceTemplateUsingFeatures(ref
FSDK.TPoint[] FacialFeatures);
```

Parameters:

Image – handle of the image to extract the face template from.

FacialFeatures – pointer to the FSDK_Features array containing facial feature coordinates.

FaceTemplate – pointer to the FSDK_FaceTemplate structure for receiving the face template.

Return Value:

Returns FSDKE_OK if successful.

FSDK_MatchFaces Function

Match two face templates. The returned value determines the similarity of the faces.

C++ Syntax:

```
int FSDK_MatchFaces(FSDK_FaceTemplate* FaceTemplate1,
FSDK_FaceTemplate* FaceTemplate2, float* Similarity);
```

Delphi Syntax:

```
function FSDK_MatchFaces(FaceTemplate1, FaceTemplate2:
PFSDK_FaceTemplate; Similarity: PSingle): integer;
```

C# Syntax:

```
int FSDK.MatchFaces(ref byte[] FaceTemplate1, ref byte[]
FaceTemplate2, ref float Similarity);
```


VB Syntax:

```
Function FSDKVB_MatchFaces(ByRef FaceTemplate1 As Byte, ByRef FaceTemplate2 As Byte, ByRef Similarity As Single) As Long
```

Java Syntax:

```
int FSDK.MatchFaces(FSDK_FaceTemplate.ByReference FaceTemplate1, FSDK_FaceTemplate.ByReference FaceTemplate2, float Similarity[]);
```

Android Syntax:

```
int FSDK.MatchFaces(FSDK_FaceTemplate FaceTemplate1, FSDK_FaceTemplate FaceTemplate2, float Similarity[]);
```

Parameters:

FaceTemplate1 – pointer to the FSDK_FaceTemplate structure, using the first template for comparison.

FaceTemplate2 – pointer to the FSDK_FaceTemplate structure, using the second template for comparison.

Similarity – pointer to an integer value, used to receive the similarity of the face templates.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_INVALID_TEMPLATE if any of the face templates is in an invalid format.

Returns FSDKE_UNSUPPORTED_TEMPLATE_VERSION if any of the templates is created with an unsupported version of FaceSDK.

FSDK_GetMatchingThresholdAtFAR Function

This function returns the threshold value for similarity to determine if two matched templates belong to the same person at a given FAR (False Acceptance Rate) value. The FAR determines the acceptable error rate when two different people's templates are mistakenly recognized as the same person. Decreasing FAR leads to an increase in FRR – i.e. with low FAR it becomes more probable that two templates from the same person will be determined as belonging to different people.

C++ Syntax:

```
int FSDK_GetMatchingThresholdAtFAR(float FARValue, float* Threshold);
```

Delphi Syntax:

```
function FSDK_GetMatchingThresholdAtFAR(FARValue: single; var Threshold: single): integer;
```

C# Syntax:

```
int FSDK.GetMatchingThresholdAtFAR(float FARValue, ref float Threshold);
```

VB Syntax:

```
Function FSDKVB_GetMatchingThresholdAtFAR(ByVal FARValue As Single, ByRef Threshold As Single) As Long
```

Java and Android Syntax:

```
int FSDK.GetMatchingThresholdAtFAR(float FARValue, float Threshold[]);
```

Parameters:

FARValue – the desired FAR value. Varies from 0.0 (means 0%) to 1.0 (means 100%).
Threshold – pointer to a float variable to store the calculated Threshold value.

Return Value:

Returns FSDKE_OK if successful.

Example

```
FSDK_FaceTemplate template1, template2;

float MatchingThreshold, Similarity;
FSDK_GetMatchingThresholdAtFAR(0.02, &MatchingThreshold);

FSDK_GetFaceTemplate(img1, &template1);
FSDK_GetFaceTemplate(img2, &template2);
FSDK_MatchFaces(&template1, &template2, &Similarity);
if (Similarity > MatchingThreshold)
    printf("Same Person\n");
else
    printf("Different Person\n");
```

FSDK_GetMatchingThresholdAtFRR Function

This function returns the threshold value for similarity to determine if two matched templates belong to the same person at a given FRR (False Rejection Rate) value. The FRR determines the acceptable error rate when two templates of the same person are identified as belonging to different people. Decreasing FRR leads to an increase in FAR – i.e. with low FRR it becomes more probable that two different people's templates will be recognized as the same person.

C++ Syntax:

```
int FSDK_GetMatchingThresholdAtFRR(float FRRValue, float* Threshold);
```

Delphi Syntax:

```
function FSDK_GetMatchingThresholdAtFRR(FRRValue: single; var Threshold: single): integer;
```

C# Syntax:

```
int FSDK.GetMatchingThresholdAtFRR(float FRRValue, ref float Threshold);
```

VB Syntax:

```
Function FSDKVB_GetMatchingThresholdAtFRR (ByVal FRRValue As Single, ByRef Threshold As Single) As Long
```

Java and Android Syntax:

```
int FSDK.GetMatchingThresholdAtFRR(float FRRValue, float Threshold[]);
```

Parameters:

FRRValue – the desired FRR value. Varies from 0.0 (means 0%) to 1.0 (means 100%).

Threshold – pointer to a float variable, used to store the calculated Threshold value.

Return Value:

Returns FSDKE_OK if successful.

Gender, Age and Facial Expression Recognition

The SDK recognizes the gender, age and facial expressions of subjects. It recognizes if a smile is present and if the eyes are open or closed. To accomplish this, first you must to detect facial features in an image, and then pass these features to the [FSDK_DetectFacialAttributeUsingFeatures](#) function, specifying the "Gender", "Age" or the "Expression" attribute.

FSDK_DetectFacialAttributeUsingFeatures Function

Detects an attribute of a face, and returns the Values of a particular attribute, and Confidences in these Values.

Each facial attribute has a number of Values, and each Value has an associated Confidence. A Value is a string, and a Confidence is a float from 0 to 1, which represents confidence level of this particular value of the attribute.

The following attribute names are supported:

"Liveness" – to get the liveness probability (see the [Passive Liveness](#) section).

"Gender" – to detect the gender of a face. The attribute has "Male" and "Female" values.

"Age" – to detect the age of a face. The attribute has "Age" value.

"Expression" – to detect the expression of a face. The attribute has "Smile" and "EyesOpen" values. The Values and their Confidences are returned in a string of the following format:

```
"Value1=Confidence1[;Value2=Confidence2[;...]]"
```

For example, when calling the function with the "Gender" attribute, the following string may be returned:

```
"Male=0.95721;Female=0.04279"
```

It means that the subject has male gender with a confidence of 95.7%, and female gender with a confidence of 4.3%.

When calling the function with the "Age" attribute, the following string may be returned:

```
"Age=37 "
```

When calling the function with the "Expression" attribute, the following string may be returned:

```
"Smile=0.987;EyesOpen=0.9952 "
```

It means that the subject smiles with a confidence of 98.7%, and the eyes are open with a confidence of 99.5%. You may use several attributes in a single function call separated by "; ". For example, if *AttributeName* is "Gender; Age; Expression", the result may be the following:

```
"Male=0.95721;Female=0.04279;Age=37;Smile=0.987;EyesOpen=0.995 "
```

You may use the [FSDK_GetValueConfidence](#) to parse the returned string and retrieve the Confidences for individual Values.

C++ Syntax:

```
int FSDK_DetectFacialAttributeUsingFeatures(HImage Image,
const FSDK_Features * FacialFeatures, const char *
AttributeName, char * AttributeValues, long long
MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_DetectFacialAttributeUsingFeatures(Image:
HImage; FacialFeatures: PFSDK_Features; AttributeName,
AttributeValues: PAnsiChar; MaxSizeInBytes: int64): integer;
```

C# Syntax:

```
int FSDK.DetectFacialAttributeUsingFeatures(int Image, ref
TPoint [] FacialFeatures, string AttributeName, out string
AttributeValues, long MaxSizeInBytes);
```

VB Syntax:

```
Function FSDKVB_DetectFacialAttributeUsingFeatures(ByVal Image
As Long, ByRef FacialFeatures As TPoint, ByVal AttributeName
As String, ByRef AttributeValues As String, ByVal
MaxSizeInBytes As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.DetectFacialAttributeUsingFeatures(int Image,
FSDK_Features FacialFeatures, String AttributeName, String
AttributeValues[], long MaxSizeInBytes);
```

Parameters:

Image – HImage handle in which to detect the attribute.

FacialFeatures – pointer to the FSDK_Features array containing facial feature coordinates.

AttributeName – name of the attribute. You may specify several attributes separated by ";".

AttributeValues – pointer to the null-terminated string that will receive the attribute Values and their Confidences.

MaxSizeInBytes – amount of memory allocated for the output string.

Return Value:

Returns FSDK_OK if successful. Returns FSDK_INSUFFICIENT_BUFFER_SIZE if there is not enough room to store the output string; however, the output string still fills up all the space available.

FSDK_GetValueConfidence Function

Parses the string returned by [FSDK_DetectFacialAttributeUsingFeatures](#) or [FSDK_GetTrackerFacialAttribute](#), and returns the Confidence in an individual Value.

C++ Syntax:

```
int FSDK_GetValueConfidence(const char * AttributeValues, const char * Value, float * Confidence);
```

Delphi Syntax:

```
function FSDK_GetValueConfidence(AttributeValues, Value: PAnsiChar; Confidence: PSingle): integer;
```

C# Syntax:

```
int FSDK.GetValueConfidence(string AttributeValues, string Value, ref float Confidence);
```

VB Syntax:

```
Function FSDKVB_GetValueConfidence(ByVal AttributeValues As String, ByVal Value As String, ByRef Confidence As Single) As Long
```

Java and Android Syntax:

```
int FSDK.GetValueConfidence(String AttributeValues, String Value, float Confidence[]);
```

Parameters:

AttributeValues – pointer to the null-terminated string containing the attribute Values and their Confidences.

Value – pointer to the null-terminated string containing the desired Value.

Confidence – pointer to the float variable to store the Confidence in a Value.

Return Value:

Returns FSDK_OK if successful.

Example:

```
char AttributeValues[1024];
FSDK_DetectFacialAttributeUsingFeatures(image, features, "Gender", AttributeValues, sizeof(AttributeValues));

float ConfidenceMale = 0.0f;
```

```
float ConfidenceFemale = 0.0f;
FSDK_GetValueConfidence(AttributeValues, "Male",
&ConfidenceMale);
FSDK_GetValueConfidence(AttributeValues, "Female",
&ConfidenceFemale);
```

Liveness Detection

The SDK offers several approaches to detect spoofing attempts (when a photo or a video is presented to the camera instead of a real person). There are passive liveness detection, active liveness detection, and thermal face detection. A combination of several approaches provides the best reliability.

Passive Liveness

Passive liveness detection is the most sophisticated anti-spoofing technology. It does not require any special hardware, nor does it ask users to perform any actions to prove the liveness - it works just by analyzing images.

Passive liveness detection works with both still images and videos. The probability of a subject being live is available as the `Liveness` facial attribute. For still images, the attribute can be retrieved with the `FSDK_DetectFacialAttributeUsingFeatures` function. For videos, use the `FSDK_GetTrackerFacialAttribute` function of Tracker API. To enable passive liveness detection in Tracker API, make sure to pass the "`DetectLiveness=true`" parameter to Tracker API.

Using Tracker API for passive liveness detection usually improves the reliability of the liveness detection, as liveness is detected in a number of facial appearances on consecutive video frames. The resulting liveness probability p is calculated as a soft minimum:

$$p = \frac{\sum_{i=1}^n p_i e^{-\alpha p_i}}{\sum_{i=1}^n e^{-\alpha p_i}},$$

where p_i is the liveness probability detected on the frame i ; n is the number of frames.

The following Tracker parameters can be used to adjust passive liveness detection:

AttributeLivenessSmoothingAlpha – the α parameter. The default value is 1.

LivenessFramesCount – the minimum number of frames required before the liveness attribute is calculated. The default value is 15.

Note that an RGB color image is required to perform the passive liveness check, grayscale images aren't supported.

Active Liveness

Active liveness check requires Tracker API, as demonstrated in the `ActiveLiveness` samples. Liveness is verified by asking the user to perform a set of actions in front of the camera.

Thermal Face Detection

Thermal face detection can also be used to verify liveness. In a typical scenario, the system is equipped with a thermal camera and an ordinary "visual" camera. Both cameras should capture the same field of view. To ensure a face is live, it must be detected in the same place both by the visual and the thermal camera. For more information see the [Face Detection on Thermal Images](#) section.

Working with Cameras

The library offers a set of functions to work with DirectShow/v4l2-compatible web cameras and IP cameras with an MJPEG interface. The functions allow single frames to be retrieved from a camera one-by-one; they are stored in HImage descriptors. The application usually grabs frames in a loop, displaying each frame in its window and performing manipulations with images (such as face detection).

Web camera functions are available only for Windows and Linux platforms. IP camera functions are available for all platforms.

Android and iOS samples include platform-specific code working with cameras on phones and tablets.

Data Types

There are data types to store the information about video formats. Note that the names of video cameras are stored in wide char format (each char occupies two bytes).

C++ Declaration:

```
typedef struct {
    int Width;
    int Height;
    int BPP;
} FSDK_VideoFormatInfo;

typedef enum {
    FSDK_MJPEG
} FSDK_VIDEOCOMPRESSIONTYPE;
```

Delphi Declaration:

```
FSDK_VideoFormatInfo = record
    Width: integer;
    Height: integer;
    BPP: integer;
end;

PFSDK_VideoFormatInfo = ^FSDK_VideoFormatInfo;
FSDK_VideoFormatInfoArray =
    array[0..255] of FSDK_VideoFormatInfo;
PFSDK_VideoFormatInfoArray = ^FSDK_VideoFormatInfoArray;

FSDK_CameraList = array[0..255] of PWideChar;
PFSDK_CameraList = ^FSDK_CameraList;
```

```
FSDK_VIDEOCOMPRESSSIONTYPE = (
    FSDK_MJPEG
);
```

VB Declaration:

```
Type FSDK_VideoFormatInfo
    Width As Long
    Height As Long
    BPP As Long
End Type

Enum FSDK_VIDEOCOMPRESSSIONTYPE
    FSDK_MJPEG
End Enum
```

Java Declaration:

The class `FSDK_VideoFormatInfo` has the following properties:

```
    public int Width, Height, BPP;

class FSDK_VideoFormats {
    public FSDK_VideoFormatInfo.ByValue formats[];
}

class TCameras {
    public String cameras[];
}
```

Java and Android Declaration:

```
class HCamera {
    protected int hcamera;
}

class FSDK_VIDEOCOMPRESSSIONTYPE {
    public static final int FSDK_MJPEG = 0;
}
```

FSDK_InitializeCapturing Function

This function initializes the capturing process (but does not open a camera). This function should be called in a certain thread that works with cameras. Note that on Windows platforms this function initializes COM in the thread; if you already initialized COM, you must not call this function, and you must not call [FSDK_FinalizeCapturing](#).

C++ Syntax:

```
int FSDK_InitializeCapturing(void);
```

Delphi Syntax:

```
function FSDK_InitializeCapturing: integer;
```


C# Syntax:

```
int FSDKCam.InitializeCapturing();
```

VB Syntax:

```
Function FSDKVB_InitializeCapturing() As Long
```

Java Syntax:

```
int FSDKCam.InitializeCapturing();
```

Android Syntax:

```
int FSDK.InitializeCapturing();
```

Return Value:

Returns FSDKE_OK if successful.

FSDK_FinalizeCapturing Function

This function finalizes the capturing process, initialized by the [FSDK_InitializeCapturing](#) function (and finalizes COM on Windows platforms). If you already finalized COM, you must not call this function.

C++ Syntax:

```
int FSDK_FinalizeCapturing(void);
```

Delphi Syntax:

```
function FSDK_FinalizeCapturing: integer;
```

C# Syntax:

```
int FSDKCam.FinalizeCapturing();
```

VB Syntax:

```
Function FSDKVB_FinalizeCapturing() As Long
```

Java Syntax:

```
int FSDKCam.FinalizeCapturing();
```

Android Syntax:

```
int FSDK.FinalizeCapturing();
```

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetCameraNaming Function

Sets the retrieval format for the [FSDK_GetCameraList](#) function. Depending on the value of the argument, either web camera names (by default) or their unique IDs (Device Path) are returned. Device Path may be necessary if the system has several web cameras from the same manufacturer that have the same name. This function does not support IP cameras.

C++ Syntax:

```
int FSDK_SetCameraNaming(bool UseDevicePathAsName);
```

Delphi Syntax:

```
function FSDK_SetCameraNaming(UseDevicePathAsName: boolean): integer;
```

C# Syntax:

```
int FSDKCam.SetCameraNaming(bool UseDevicePathAsName)
```

VB Syntax:

```
Function FSDKVB_SetCameraNaming(ByVal UseDevicePathAsName As Boolean) As Long
```

Java Syntax:

```
int FSDKCam.SetCameraNaming (boolean UseDevicePathAsName);
```

Parameters:

UseDevicePathAsName – sets a retrieval format for the [FSDK_GetCameraList](#) function.

FALSE: [FSDK_GetCameraList](#) returns the list of names for cameras installed in the system;

TRUE: [FSDK_GetCameraList](#) returns the list of unique device paths of these cameras.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GetCameraList Function

This function retrieves the list of web cameras available in the system. The name of each camera is stored in wide char format (each character occupies two bytes). The function does not support IP cameras. The camera list must be destroyed by calling the [FSDK_FreeCameraList](#) function after the list is no longer needed.

C++ Syntax:

```
int FSDK_GetCameraList(wchar_t*** CameraList, int* CameraCount);
```

Delphi Syntax:

```
function FSDK_GetCameraList(CameraList: PWideChar; CameraCount: Pinteger): integer;
```

C# Syntax:

```
int FSDKCam.GetCameraList(out string[] CameraList, out int CameraCount)
```

VB Syntax:

```
Function FSDKVB_GetCameraList(ByRef CameraList As Variant, ByRef CameraCount As Long) As Long
```

Java Syntax:

```
int FSDKCam.GetCameraList(TCameras CameraList, int
CameraCount[]);
```

Parameters:

CameraList – pointer to `wchar_t**` variable to store the camera list.

CameraCount – pointer to integer variable to store the count of cameras in the system.

Return Value:

Returns `FSDKE_OK` if successful.

Example

```
wchar_t** CameraList;
int CameraCount;

FSDK_InitializeCapturing();
if (FSDK_GetCameraList(&CameraList, &CameraCount)==FSDKE_OK)
    for (int i=0; i<CameraCount; i++)
        wprintf(L"camera: %s\n", CameraList[i]);
printf("%d camera(s) found.\n", CameraCount);
FSDK_FinalizeCapturing();
```

FSDK_GetCameraListEx Function

This function retrieves the list of names and the device paths of the web cameras available in the system. The name and the device path of each camera are stored in wide char format (each character occupies two bytes) at the same indices at the corresponding arrays. The function does not support IP cameras. Both lists must be destroyed by calling the [FSDK_FreeCameraList](#) function after they are no longer needed.

C++ Syntax:

```
int FSDK_GetCameraListEx(wchar_t*** CameraNameList, wchar_t***
CameraDevicePathList, int* CameraCount);
```

Delphi Syntax:

```
function FSDK_GetCameraListEx(CameraNameList: PWideChar;
CameraDevicePathList: PWideChar; CameraCount: PInteger):
integer;
```

C# Syntax:

```
int FSDKCam.GetCameraListEx(out string[] CameraNameList, out
string[] CameraDevicePathList, out int CameraCount)
```

VB Syntax:

```
Function FSDKVB_GetCameraListEx(ByRef VCameraNameList As
Variant, ByRef VCameraDevicePathList As Variant, ByRef
CameraCount As Long) As Long
```

Java Syntax:

```
int FSDKCam.GetCameraListEx(TCameras CameraNameList, TCameras  
CameraDevicePathList, int CameraCount[]);
```

Parameters:

CameraNameList – pointer to `wchar_t**` variable to store the camera name list.

CameraDevicePathList – pointer to `wchar_t**` variable to store the camera device path list.

CameraCount – pointer to integer variable to store the number of cameras in the system.

Return Value:

Returns `FSDKE_OK` if successful.

FSDK_FreeCameraList Function

This function frees the list of web cameras obtained from the [FSDK_GetCameraList](#) or [FSDK_GetCameraListEx](#) function. The call of the function is not required in .NET, VB and Java.

C++ Syntax:

```
int FSDK_FreeCameraList(wchar_t*** CameraList, int  
CameraCount);
```

Delphi Syntax:

```
function FSDK_FreeCameraList(CameraList: Pointer; CameraCount:  
integer): integer;
```

Parameters:

CameraList – pointer to `wchar_t**` variable where the camera list is stored.

CameraCount – the count of cameras in the system, obtained from the [FSDK_GetCameraList](#) or [FSDK_GetCameraListEx](#) function.

Note:

You must call [FSDK_FreeCameraList](#) for *CameraNameList* and *CameraDevicePathList*, if you were using [FSDK_GetCameraListEx](#).

Return Value:

Returns `FSDKE_OK` if successful.

FSDK_GetVideoFormatList Function

This function returns the list of video formats supported by a given camera. This function does not support IP cameras.

C++ Syntax:

```
int FSDK_GetVideoFormatList(wchar_t* CameraName,  
FSDK_VideoFormatInfo** VideoFormatList, int*  
VideoFormatCount);
```

Delphi Syntax:

```
function FSDK_GetVideoFormatList(CameraName: PWideChar;
VideoFormatList: PFSDK_VideoFormatInfo; VideoFormatCount:
Pinteger): integer;
```

C# Syntax:

```
int FSDKCam.GetVideoFormatList(string CameraName, out
FSDKcam.VideoFormatInfo[] VideoFormatList, out int
VideoFormatCount)
```

VB Syntax:

```
Function FSDKVB_GetVideoFormatList(ByVal CameraName As String,
ByRef VideoFormatList As Variant, ByRef VideoFormatCount As
Long) As Long
```

Java Syntax:

```
int FSDKCam.GetVideoFormatList(String CameraName,
FSDK_VideoFormats VideoFormatList, int VideoFormatCount[]);
```

Parameters:

CameraName – pointer to name of desired video camera.

VideoFormatList – pointer to FSDK_VideoFormatInfo* variable to store the list of video formats.

VideoFormatCount – pointer to integer variable to store the count of video formats.

Return Value:

Returns FSDKE_OK if successful.

FSDK_FreeVideoFormatList Function

This function frees the list of video formats obtained from [FSDK_GetVideoFormatList](#). Calling this function is not required in .NET, VB and Java.

C++ Syntax:

```
int FSDK_FreeVideoFormatList(FSDK_VideoFormatInfo *
VideoFormatList);
```

Delphi Syntax:

```
function FSDK_FreeVideoFormatList(VideoFormatList: Pointer):
integer;
```

Parameters:

VideoFormatList – pointer to FSDK_VideoFormatInfo* variable where the list of video formats is stored.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetVideoFormat Function

The function sets the format of camera output. The function does not support IP cameras.

C++ Syntax:

```
int FSDK_SetVideoFormat(wchar_t* CameraName,  
FSDK_VideoFormatInfo VideoFormat);
```

Delphi Syntax:

```
function FSDK_SetVideoFormat(CameraName: PWideChar;  
VideoFormat: FSDK_VideoFormatInfo): integer;
```

C# Syntax:

```
int FSDKCam.SetVideoFormat(ref string CameraName,  
FSDKcam.VideoFormatInfo VideoFormat);
```

VB Syntax:

```
Function FSDKVB_SetVideoFormat(ByVal CameraName As String,  
ByRef VideoFormat As FSDK_VideoFormatInfo) As Long
```

Java Syntax:

```
int FSDKCam.SetVideoFormat(String CameraName,  
FSDK_VideoFormatInfo.ByValue VideoFormat);
```

Parameters:

CameraName – pointer to name of desired video camera.

VideoFormat – desired video format.

Return Value:

Returns FSDKE_OK if successful.

Example

```
wchar_t** CameraList;  
int CameraCount;  
FSDK_VideoFormatInfo* VideoFormatList;  
int VideoFormatCount;  
  
FSDK_GetCameraList(&CameraList, &CameraCount);  
  
FSDK_GetVideoFormatList(CameraList[0], &VideoFormatList,  
&VideoFormatCount);  
FSDK_SetVideoFormat(CameraList[0], VideoFormatList[0]);
```

FSDK_OpenVideoCamera Function

The function opens the web camera of a given name and returns its handle.

C++ Syntax:

```
int FSDK_OpenVideoCamera(wchar_t* CameraName, int*  
CameraHandle);
```

Delphi Syntax:

```
function FSDK_OpenVideoCamera(CameraName: PWideChar;
CameraHandle: PInteger): integer;
```

C# Syntax:

```
int FSDKCam.OpenVideoCamera(ref string CameraName, ref int
CameraHandle);
```

VB Syntax:

```
Function FSDKVB_OpenVideoCamera(ByVal CameraName As String,
ByRef CameraHandle As Long) As Long
```

Java Syntax:

```
int FSDKCam.OpenVideoCamera (String CameraName, HCamera
CameraHandle);
```

Parameters:

CameraName – pointer to name of web camera to open.

CameraHandle – pointer to integer variable to store the opened camera handle.

Return Value:

Returns FSDKE_OK if successful.

FSDK_OpenIPVideoCamera Function

This function opens the IP camera at a given URL and returns its handle. You may call the [FSDK_SetHTTPProxy](#) function to set an HTTP proxy for accessing the camera.

C++ Syntax:

```
int FSDK_OpenIPVideoCamera(FSDK_VIDEOCOMPRESSSIONTYPE
CompressionType, char * URL, char * Username, char * Password,
int TimeoutSeconds, int * CameraHandle);
```

Delphi Syntax:

```
function FSDK_OpenIPVideoCamera(CompressionType:
FSDK_VIDEOCOMPRESSSIONTYPE; URL: PAnsiChar; Username:
PAnsiChar; Password: PAnsiChar; TimeoutSeconds: integer;
CameraHandle: PInteger): integer;
```

C# Syntax:

```
int FSDKCam.OpenIPVideoCamera(FSDK_VIDEOCOMPRESSSIONTYPE
CompressionType, string URL, string Username, string Password,
int TimeoutSeconds, ref int CameraHandle);
```

VB Syntax:

```
Function FSDKVB_OpenIPVideoCamera(ByVal CompressionType As
FSDK_VIDEOCOMPRESSSIONTYPE, ByVal URL As String, ByVal Username
As String, ByVal Password As String, ByVal TimeoutSeconds As
Long, ByRef CameraHandle As Long) As Long
```

Java Syntax:

```
int FSDKCam.OpenIPVideoCamera(int CompressionType, String URL,
String Username, String Password, int TimeoutSeconds, HCamera
CameraHandle);
```

Android Syntax:

```
int
FSDK.OpenIPVideoCamera(FSDK_VIDEOCOMPRESSSIONTYPECompressionTyp
e, String URL, String Username, String Password, int
TimeoutSeconds, HCamera CameraHandle);
```

Parameters:

CompressionType – the type of video stream (MJPEG by default).

URL – URL of the IP camera to be opened.

Username – IP camera access username.

Password – IP camera access password.

TimeoutSeconds – connection timeout in seconds.

CameraHandle – pointer to integer variable to store the opened camera handle.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetHTTPProxy Function

This function sets an HTTP proxy to be used with an IP camera. If a proxy is required, the function should be called before the [FSDK_OpenIPVideoCamera](#) function.

C++ Syntax:

```
int FSDK_SetHTTPProxy(char* ServerNameOrIPAddress, unsigned
short Port, char* UserName, char* Password);
```

Delphi Syntax:

```
function FSDK_SetHTTPProxy(ServerNameOrIPAddress: PAnsiChar;
Port: Word; Username: PAnsiChar; Password: PAnsiChar):
integer;
```

C# Syntax:

```
int FSDK.SetHTTPProxy(string ServerNameOrIPAddress, UInt16
Port, string UserName, string Password);
```

VB Syntax:

```
Function FSDKVB_SetHTTPProxy(ByVal ServerNameOrIPAddress As
String, ByVal Port As Long, ByVal Username As String, ByVal
Password As String) As Long
```

Java Syntax:

```
int FSDKCam.SetHTTPProxy(String ServerNameOrIPAddress, int
Port, String UserName, String Password);
```


Android Syntax:

```
int FSDK.SetHTTPProxy(String ServerNameOrIPAddress, short Port,  
String UserName, String Password);
```

Parameters:

ServerNameOrIPAddress – proxy address.

Port – proxy port.

UserName – proxy username.

Password – proxy password.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GrabFrame Function

Retrieves the current frame from a web camera or an IP camera and stores the frame in the created HImage handle. If a camera returns an image, mirrored horizontally (it depends on the camera settings), then you can mirror it by using [FSDK_MirrorImage](#).

C++ Syntax:

```
int FSDK_GrabFrame(int CameraHandle, HImage* Image);
```

Delphi Syntax:

```
function FSDK_GrabFrame(CameraHandle: integer; var Image:  
PHImage): integer;
```

C# Syntax:

```
int FSDKCam.GrabFrame(int CameraHandle, ref int Image);
```

VB Syntax:

```
Function FSDKVB_GrabFrame(ByVal CameraHandle As Long, ByRef  
Image As Long) As Long
```

Java Syntax:

```
int FSDKCam.GrabFrame(HCamera CameraHandle, HImage Image);
```

Android Syntax:

```
int FSDK.GrabFrame(HCamera CameraHandle, HImage Image);
```

Parameters:

CameraHandle – handle of the opened camera to grab frame.

Image – pointer to HImage variable to store the frame. Note that the created HImage handle should be deleted once it is no longer needed using the [FSDK_FreeImage](#) function.

Return Value:

Returns FSDKE_OK if successful.

FSDK_CloseVideoCamera Function

This function closes the camera, opened by the [FSDK_OpenVideoCamera](#) or [FSDK_OpenIPVideoCamera](#) function.

C++ Syntax:

```
int FSDK_CloseVideoCamera(int CameraHandle);
```

Delphi Syntax:

```
function FSDK_CloseVideoCamera(CameraHandle: integer):  
integer;
```

C# Syntax:

```
int FSDKCam.CloseVideoCamera(int CameraHandle);
```

VB Syntax:

```
Function FSDKVB_CloseVideoCamera(ByVal CameraHandle As Long)  
As Long
```

Java Syntax:

```
int FSDKCam.CloseVideoCamera(HCamera CameraHandle);
```

Parameters:

CameraHandle – handle of opened video camera to close.

Return Value:

Returns FSDKE_OK if successful.

Tracker API: Face Recognition and Tracking in Video Streams

What is Tracker API

Tracker API is a set of functions that allows for recognizing subjects in live video streams. The API receives the video frame by frame, and assigns a unique identifier (ID) to each subject detected in the video. Thus, each subject can be determined by its ID across the video. You can attach a name tag to an identifier, and query any identifier for its name. The API also allows simple face tracking (without registering subjects); tracking of the coordinates of either all facial features or just eye centers; and recognition of subjects' gender, age and facial expression. The API provides an estimate of both recognition rate and false acceptance rate as the video progresses.

If your task is to track or recognize faces in video streams, consider using Tracker API instead of manually calling functions like [FSDK_DetectFace](#), [FSDK_DetectFacialFeatures](#) or [FSDK_GetFaceTemplate](#) for each frame (“manual handling”). The difference between Tracker API and manual handling is summarized in the table below.

	Tracker API	Manual handling
--	-------------	-----------------

	Tracker API	Manual handling
Development effort	A developer uses a few Tracker API functions to handle an incoming video frame and set and retrieve subjects' names. The API automatically learns the appearance of every detected subject.	A developer must implement different modes in the program: a mode to enroll subjects and a mode to recognize faces. In the enrollment mode, the program must store a certain (usually found experimentally) number of face templates in the database, while the subject is posing in front of the camera. In recognition mode, a template for each detected face is created and is matched against the database.
Performance	The API constantly learns how subjects appear. Thus, its recognition rate is usually higher than of a system that merely stores several templates of a subject.	If environmental conditions (such as lighting) change after the enrollment, the system may not recognize the subject, and a new enrollment will be required.
Recognition rates	Tracker API provides the estimate of recognition rate and false acceptance rate specifically for video streams.	FSDK MatchFaces provides FAR/FRR values for matching a pair of images. Typically, it is not easy to estimate how the storage of several templates per person affects recognition rate, how often false acceptances occur as the video progresses, and if false acceptance rate increases as more subjects are enrolled.
Enrollment	The subject is generally not required to pose. When the operator assigns a name to the subject, it is likely that Tracker API has already captured enough views of a subject to recognize it in later frames.	The subject is required to pose in front of the camera, for the system to capture the face in different views and environmental conditions, and with different facial expressions.

	Tracker API	Manual handling
Recognition without enrollment	<p>Every subject is recognized, regardless of whether it was already tagged with a name. The API assigns a unique ID to track the subject across the video.</p> <p>This allows for surveillance applications, when subjects cannot be required to participate willingly (that is, to pose) to be enrolled for recognition.</p>	Only enrolled subjects can be recognized. The requirement to participate actively in recognition makes surveillance applications difficult.
Tracking of multiple faces	The API tracks, recognizes, and allows assigning names to multiple faces simultaneously present in the video frame.	Usually only a single subject can pose in front of the camera when enrolling. If other subjects are visible, the system may mistakenly store their templates into the subject's database record. A separate tracking mechanism is required to decide whether the detected face belongs to the enrolled subject or not.
Facial feature detection	Tracker API allows tracking of facial feature coordinates of each subject in the video frame. Jitter is eliminated by smoothing.	The coordinates detected by FSDK_DetectFacialFeatures may jitter because of noise present in the video. If multiple faces are present, a tracking mechanism is required to implement smoothing.
Gender and age recognition	The API allows for identifying gender and age for each subject tracked in the video. The analysis of the video usually provides higher recognition rates than still image gender and age recognition.	When each video frame is treated as a still image, gender and age recognition rates are usually lower.
Facial expression recognition	The API allows for identifying if a smile is present and if the eyes are open or closed for each subject tracked in the video. The analysis of the video usually provides higher recognition rates than still image expression recognition.	When each video frame is treated as a still image, expression recognition rates are usually lower.

Understanding Identifiers

The API analyzes the video stream sequentially, frame by frame. For each frame, the function [FSDK_FeedFrame](#) returns the list of identifiers (integer numbers) of faces recognized in this frame. The purpose of an identifier is to assign a unique number to each subject in the video. If a face is similar to one recognized previously, it receives the same identifier. Otherwise, a new identifier (in ascending numeric order) is assigned. Thus, subjects recognized as different should get different identifiers.

It is important to note that the identifier value is meaningful only within a particular video stream. Identifiers of the same subject are not expected to be the same across different video streams.

A subject can have several identifiers

The same subject can get different identifiers in different frames (for example, ID1 in the first frame and ID2 in the second, $ID2 > ID1$), if the system was not able to match its face to ones previously seen (which might happen if the appearance of the subject on the second frame was notably or unexpectedly different).

Merger of identifiers

However, as the video progresses, the system learns more about the appearance of each person; at some point it may deduce that ID1 and ID2 actually represent the same person. In such a case (and if it is possible) it merges both identifiers into ID1, further returning ID1 for every novel recognized occurrence of this subject. The system retains the information of all merger events, so it is possible to receive the resulting value of an early assigned identifier (for example, receive the ID1 value when providing the ID2 value) by calling the [FSDK_GetIDReassignment](#) function. Note that if an identifier was tagged with a name, it can be merged only with other identifiers that are untagged; in such a case the tagged name is retained.

When calling Tracker API functions with identifiers received on earlier frames, it is always recommended to convert the identifier values with the [FSDK_GetIDReassignment](#) function first, and only then pass them to Tracker API. The reason is that they may have been merged on the subsequent frames, so the corresponding subjects are being represented with other identifier values.

When identifiers are not merged

The API supports tagging an identifier with a name, provided by the user. If identifiers are tagged with different names, they will not be merged.

The appearances of each subject are stored in the memory (see the [Memory](#) section). If a subject has been tagged with a name, and the memory for this subject is full, it will not be merged with any other identifier (because such a merger requires additional memory for the subject).

Similar identifiers

The identifier returned by the [FSDK_FeedFrame](#) function can be similar enough to other identifiers for the API to decide they represent the same person. Still, some reason (such as the one described above) may prevent them from merging. In such case, similar identifiers of an ID can be retrieved using the [FSDK_GetSimilarIDList](#) function.

You should always retrieve the list of similar identifiers when deciding if the recognized face belongs to a certain subject or not. Let us assume that you have a particular subject of interest and should respond when it is recognized. You may have stored an identifier of that subject, or assigned a name to it with [FSDK_SetName](#), and wait for such identifier (or name) to appear. (Keep in mind that you need to adjust the stored identifier with [FSDK_GetIDReassignment](#) after calling [FSDK_FeedFrame](#).) When the subject appears, however, there is no guarantee that the stored identifier will be returned by the [FSDK_FeedFrame](#) function. Instead, it may appear in the list of similar identifiers. Therefore, you should compare your identifier against the list of similar identifiers for each ID returned by [FSDK_FeedFrame](#). Accordingly, you need to retrieve the names of each similar identifier, for each ID returned by [FSDK_FeedFrame](#), to find if any of these names belong to the subject of interest. If you are not considering such lists of similar identifiers, your recognition rate will be lower (that is, you may miss the appearance of the subject of interest). Of course, your false acceptance rate will be lower as well. But the drop in recognition rate will be higher compared to when you set a higher recognition threshold (see the [Recognition Performance](#) section), and handle similar identifiers.

The function [FSDK_GetAllNames](#) implements the above functionality – it returns the name of an identifier, concatenated with the names (if any) of similar identifiers, separated by a semicolon.

Tracker Memory

The API allows limiting the memory used by a tracker. The memory size is measured in the total number of facial appearances stored (about 11 Kbytes per appearance when the `KeepFaceImages` parameter is set to true, and about 1.5 Kbytes when set to false). By default, the limit is 2150 appearances (about 24 Mbytes or 3 Mbytes depending on the value of the `KeepFaceImages` parameter). You can change the limit by setting the `MemoryLimit` parameter (see the [Tracker Parameters](#) section) to your desired value.

Memory available for each subject

For each subject tagged with a name, the amount of memory available is

$$\max(1, \text{memoryLimit}/(\text{subjectCount}+1))$$

where `subjectCount` is the total number of subjects tagged with a name. The remaining memory is dedicated to untagged identifiers.

If, when setting a name with [FSDK_SetName](#), there is not enough room for a new subject, the API will return the `FSDKE_INSUFFICIENT_TRACKER_MEMORY_LIMIT` error.

Imposing memory limits

If a memory limit for an identifier, tagged with a name, is approached, then no new appearances of that subject will be stored. That is, the system stops learning novel appearances of the subject. Furthermore, the identifier will not be merged with any other identifiers.

If a memory limit is approached for untagged identifiers, the earliest untagged facial appearance becomes purged when calling [FSDK_FeedFrame](#). Note that only a particular appearance of some untagged identifier becomes purged, not the identifier's entire record of appearances; identifiers that have only one occurrence are purged completely. To prevent purging, you may use the [FSDK_LockID](#) function.

Note that if an identifier is tagged, and does not occupy more memory than available per subject, its facial appearances are not purged.

How to set the memory limit

The higher the limit, the more identifiers you can tag, and the more facial appearances can be stored for each identifier (thus improving the recognition rate). However, the Threshold parameter should also be higher (but setting too high a Threshold has its downsides – see the [Recognition Performance](#) section), for the false acceptance rate to stay at an acceptable level.

When increasing MemoryLimit, the frame rate may decrease. Therefore, it is practical to choose a memory limit that will allow for a sufficient frame rate, will not require too high a threshold, and will consume only a certain amount of memory, while at the same time allowing for the storage of the desired number of subjects.

See the [Recognition performance](#) section to find which Threshold values should be chosen with different memory limits to achieve the desired recognition rates.

Tracker Parameters

Each HTracker instance allows setting a number of parameters with the [FSDK_SetTrackerParameter](#) or [FSDK_SetTrackerMultipleParameters](#) function.

Face tracking parameters

Note that the Tracker API does not use the parameters of face detection, set with [FSDK_SetFaceDetectionParameters](#) or [FSDK_SetFaceDetectionThresholds](#). Instead, you should use the Tracker API parameters below.

FaceDetectionModel, *TrimOutOfScreenFaces*,
TrimFacesWithUncertainFacialFeatures – the parameters analogous to ones described in the FaceSDK [Parameters](#) section. Their default values are (default, true, true).

HandleArbitraryRotations, *DetermineFaceRotationAngle*,
InternalResizeWidth – the parameters analogous to ones in [FSDK_SetFaceDetectionParameters](#). Their default values are (false, false, 256).

FaceDetectionThreshold – a parameter analogous to one in [FSDK_SetFaceDetectionThreshold](#). The default value is 5.

FaceTrackingDistance – specifies the maximum distance between faces of one person on consecutive frames, to consider an uninterrupted tracking sequence. The parameter is measured in width of the detected face. The default value is 0.5. You may decrease it when the frame rate is high to lower the probability of false acceptances, or increase it when the frame rate is low and the recognition rate is low due to interrupted tracking.

Face recognition parameters

RecognizeFaces – whether to recognize subject's identity. If set to true, the system attempts to assign each subject a unique id, while giving equal identifiers to the same subject across the video. If set to false, the system will return a unique ID value for every uninterrupted sequence of a detected face (that is, when a certain face is detected on every frame of the sequence), regardless of the identity of this face. The default value is true.

DetectGender – whether to recognize the gender of a subject. Gender recognition requires the detection of facial features, so when set to true, facial features are detected regardless of the *DetectFacialFeatures* parameter. The default value is false.

DetectAge – whether to recognize the age of a subject. Age recognition requires the detection of facial features, so when set to true, facial features are detected regardless of the *DetectFacialFeatures* parameter. The default value is false.

DetectExpression – whether to recognize facial expression of a subject. Expression recognition requires the detection of facial features, so when set to true, facial features are detected regardless of the *DetectFacialFeatures* parameter. The default value is false.

DetectLiveness – whether to perform passive liveness detection. See the [Passive Liveness](#) section for more details. The default value is false.

Learning – whether to learn subjects' appearances. If set to true, the API will learn the appearance of each subject, unless its memory is full, and add new subjects to the memory. If set to false, the system will return only the identifiers already present in the memory; no addition of novel subjects, novel facial appearances, or merger of identifiers will occur. If a subject does not match any appearance stored in the memory, [FSDK_FeedFrame](#) will return the -1 identifier for that subject. Set this flag to false if you have a reason not to alter the memory. The default value is true.

MemoryLimit – the amount of memory available for the storage of facial appearances. See the [Tracker Memory](#) section below. The default value is 2150.

Threshold – the threshold used when deciding if two facial appearances belong to the same subject. Each threshold value alters both the false acceptance rate and recognition rate. See the [Recognition Performance](#) below. The default value is 0.992.

KeepFaceImages – whether to store the original facial images in the Tracker memory. See the [Storing original facial images](#), section for details. The default value is true.

Facial feature tracking parameters

DetectEyes – whether to detect eyes. Eyes will be detected regardless of the value of this parameter when *RecognizeFaces* is set to 1. When eyes are detected, their coordinates can be retrieved with [FSDK_GetTrackerEyes](#). The default value is false.

DetectFacialFeatures – whether to detect facial features. Facial features are detected when *RecognizeFaces* is set to 1, regardless of the value of this parameter. They are also detected if *DetectGender*, *DetectAge* or *DetectExpression* are set to 1. The default value is false.

DetectAngles – whether to estimate out-of-plane face rotation angles by using the detected facial features. Pan and Tilt are returned as the *Angles* facial attribute. The default value is false.

FacialFeatureJitterSuppression – whether to suppress the jitter of facial features by employing more processor resources. If 0, such jitter suppression is not employed. Set to a higher value for better suppression. A non-zero setting takes effect only when *DetectFacialFeatures*=true, even if facial features are actually detected due to the setting of the *RecognitionPrecision*, *DetectGender*, *DetectAge* or *DetectExpression* parameters.

The default value depends on *NUM_THREADS*, the number of threads supported by the CPU, which can be obtained using the [FSDK_GetNumThreads](#) function. The default value is *NUM_THREADS* – 1 when *NUM_THREADS* <= 4; 3 when *NUM_THREADS* <= 8;

NUM_THREADS/2 - 1 when NUM_THREADS <= 40; and 19 otherwise. Note that the way the default value is selected may change the behavior of Tracker API from system to system. That is, systems supporting more threads will display smoother facial feature coordinates by default. This may also change the behavior of face recognition and attribute detection (although it will not change their average accuracy). If you need Tracker API to return the same output for the same input data regardless of the number of threads, set this parameter to a fixed value in your application.

SmoothFacialFeatures – whether to smooth facial features from frame to frame to prevent jitter. If set to false, the coordinates of facial features are detected independently of the previous frame, and may jitter because of the noise present in the video. If the parameter is set to true, the API will smooth the coordinates of facial features. The default value is true.

FacialFeatureSmoothingSpatial – a coefficient employed in facial feature smoothing. Controls spatial smoothing of facial features. The default value is 0.5.

FacialFeatureSmoothingTemporal – a coefficient employed in facial feature smoothing. Affects temporal smoothing of facial features (that is, how the smoothed coordinates relate to their coordinates on the previous frame). The default value is 250.

Tuning for Optimal Performance

The higher frame rates of [FSDK_FeedFrame](#) (i.e., fast processing of frames) usually positively affect the recognition rate for live video, because more facial appearances of a person can be captured per unit of time.

Experiment with face detection parameters, especially with `InternalResizeWidth`: higher values allow for faces to be detected at greater distance, but require additional time (and lower the frame rate). If you find a high number of false detections (i.e. when faces are detected where they are not present), try increasing the `FaceDetectionThreshold` parameter.

Setting `DetectGender`, `DetectAge` or `DetectExpression` to true will lower the frame rate. If you need only to detect gender, age or facial expressions, you may consider setting the `RecognizeFaces` parameter to false, in order to increase the frame rate.

Using the API

The API allows for creating several trackers within the program, each having a separate memory for the recognized subjects and their names.

The tracker is represented with the `HTracker` data type.

C++ Declaration:

```
typedef unsigned int HTracker;
```

Locking identifiers

There are cases when you need to work with (or tag) an identifier across several frames. For example, you may have the user interface running in a different thread than [FSDK_FeedFrame](#). Then, there is a chance that when a user selects an untagged identifier and starts to enter a name for it, the identifier may become purged by [FSDK_FeedFrame](#) running in parallel (see the [Tracker Memory](#) section). To prevent this, you need to use the [FSDK_LockID](#) functions as soon as the user selected an identifier. The function will prevent the untagged identifier from being purged completely.

Multiple camera support

Tracker API is designed to support multiple cameras, though in the current release only a single camera is supported. You should pass 0 as the CameraIdx parameter to every function that accepts it. You should not alternate frames from multiple cameras while sending them to [FSDK_FeedFrame](#), since it will disrupt the tracking process, and yield a lower recognition rate and a higher false acceptance rate. It is also not recommended to switch from one camera to another while sending the frames using [FSDK_FeedFrame](#). It is acceptable, however, to switch cameras before the memory of the tracker is loaded with [FSDK_LoadTrackerMemoryFromFile](#) or [FSDK_LoadTrackerMemoryFromBuffer](#).

Storing original facial images

As the internal format of facial appearances may change in future versions of FaceSDK, Tracker API has the KeepFaceImages parameter, which controls whether the original facial images are stored in the Tracker memory. If the format changes, you will be able to convert your Tracker memory to the new format automatically (if you've stored the original facial images). In such a case, you won't need to reenroll your subjects. It is recommended that you keep this parameter set to true, its default setting.

When the KeepFaceImages parameter is set to true, Tracker API stores an original facial image along with every facial appearance in the Tracker memory. The size of a facial appearance is about 1.5 Kbytes when KeepFaceImages is set to false, and about 11 Kbytes when KeepFaceImages is set to true. Note that if you've had this parameter set to false and accumulated some facial appearances, their original facial images will be lost, even if you set KeepFaceImages to true after that.

If you don't want the original facial images to be stored in the Tracker memory, set this parameter to false.

Usage Scenario

The following scenario is employed when using Tracker API.

1. Create a tracker ([FSDK_CreateTracker](#)) or load it from a file ([FSDK_LoadTrackerMemoryFromFile](#)) or from a memory buffer ([FSDK_LoadTrackerMemoryFromBuffer](#)).
2. Set tracker parameters ([FSDK_SetTrackerParameter](#), [FSDK_SetTrackerMultipleParameters](#)), such as face detection parameters, recognition precision, or the option to recognize gender/age/facial expression or to detect facial features.
3. Open a video camera ([FSDK_OpenVideoCamera](#), [FSDK_OpenIPVideoCamera](#)), or prepare to receive video frames from another source.
4. In a loop:
 - 1) Receive a frame from a camera ([FSDK_GrabFrame](#)) or another source.
 - 2) Send the image to the [FSDK_FeedFrame](#) function.
 - 3) Display the image on a screen.
 - 4) For each ID returned by [FSDK_FeedFrame](#):
 - i. Retrieve its facial coordinates ([FSDK_GetTrackerFacePosition](#)), eye center coordinates ([FSDK_GetTrackerEyes](#)), facial feature coordinates

- ([FSDK_GetTrackerFacialFeatures](#)), gender, age or facial expression ([FSDK_GetTrackerFacialAttribute](#)).
 - ii. Retrieve the list of possible names ([FSDK_GetAllNames](#)).
 - iii. If, relying on coordinates, you found that that user has clicked on a face, call [FSDK_LockID](#) on that identifier, display an input box and ask the user for a name of the subject. You may continue to run [FSDK_FeedFrame](#) in parallel.
 - iv. If the user entered a name, set it using the [FSDK_SetName](#) function. If the user chose to erase the subject, call [FSDK_SetName](#) with an empty name. In any case, call [FSDK_UnlockID](#) to unlock the identifier.
 - v. If manually handling identifiers (for example, storing the identifier of each subject to look up them later, or storing images of each subject), call [FSDK_GetSimilarIDCount](#) and [FSDK_GetSimilarIDList](#) to retrieve identifiers, similar to ID, and store (or compare against) them as well. In addition, call [FSDK_GetIDReassignment](#) for every previously stored identifier before comparing against them.
- 5) If necessary, save tracker memory to a file or a buffer ([FSDK_SaveTrackerMemoryToFile](#), [FSDK_SaveTrackerMemoryToBuffer](#)).
- 5. Free the tracker handle using [FSDK_FreeTracker](#).
- 6. Close the video camera ([FSDK_CloseVideoCamera](#)).

User Interaction with the System

In a typical scenario, a user observes the images from a camera, with faces outlined in rectangles and names displayed under the rectangles. There is an option to tag a subject with a name by clicking its face and entering the name, or to remove the subject from the memory. The software may notify the user when some previously defined subjects appear. The software may additionally store each image of a subject, and allow browsing such subject's images. The software may store images of untagged subjects as well (and store their ID along with the image), but keep in mind that if the memory limit is reached, earlier appearances of untagged subjects will be purged, and should these subjects appear again, they may be given with new ID numbers (unrelated to their old identifiers; see the [Tracker Memory](#) section).

The user normally should have control over the MemoryLimit and Theshold parameters to alter the recognition quality and the number of subjects that can be stored in the system.

Enrollment

To enroll a subject, the user is usually only required to click a subject's face and enter the name. If the subject has been already present in front of the camera for a certain time (for example, while approaching the user's desk), it is likely that the API has stored enough facial appearances of the subject to recognize it again. If this is not the case, the subject may be asked to tilt or rotate its head, to walk closer to or further away from the camera, and the lighting can be altered. If the frame rate is especially low, or if environmental conditions change unexpectedly, the API may not recognize the subject in some appearances. In such cases, the user may tag a subject with the same name on several occasions, until enough facial appearances are stored, and the subject is consistently recognized.

If you need to ensure that you track a live subject, consider detecting whether the facial expression changes with the [FSDK_GetTrackerFacialAttribute](#) function.

Dealing with false acceptances

The API is designed to return several names with [FSDK_GetAllNames](#) for a certain ID. In most cases, the system will return only a single name. If the system returns several names, it means that a false acceptance has occurred. That is, two (or more) subjects became confused.

Although the false acceptance rate is usually low, there is no way to eliminate it completely; instead, the user balances the false acceptance rate against the recognition rate. The software should account for the scenario when a false acceptance has been occurred.

In an access control setting, you may decide to grant access to the subject if any of the names recognized has the appropriate permissions. Alternatively, the software may signal about a false acceptance, and the user may decide to set the Threshold parameter to a higher value – to lower the probability of next false acceptance. In that case it is necessary, first, to erase the persons that were confused (by calling [FSDK_SetName](#) with an empty name to remove the name, and [FSDK_PurgeID](#) to remove all facial appearances of this ID), and then, when the threshold is set to a higher value, to set their names again.

Keep in mind that not every false acceptance will return several names of a person. It is possible that just a single incorrect name is returned, and the false acceptance may go unnoticed. However, with the appropriate setting of the Threshold parameter, such scenarios are rare.

Note that when there are one or more similar identifiers returned with [FSDK_GetSimilarIDList](#), and these identifiers do not have name tags, this does not always mean a false acceptance. As described in the [Understanding Identifiers](#) section, when the memory for an identifier is full, it will not become merged with other identifiers (even if they represent the same subject), so these identifiers will be returned in the list of similar identifiers.

Saving and Loading Tracker Memory

To save the memory of a tracker to file, use the [FSDK_SaveTrackerMemoryToFile](#) function. Alternatively, you may save it to a memory buffer (for example, to for later importing into a database). You need to call [FSDK_GetTrackerMemoryBufferSize](#) to determine the size of the buffer, and then call [FSDK_SaveTrackerMemoryToBuffer](#).

Conversely, to load the memory of a tracker from a file or a buffer, use the [FSDK_LoadTrackerMemoryFromFile](#) or [FSDK_LoadTrackerMemoryFromBuffer](#) functions. Note that you need to set the tracker parameters again after loading, because a new tracker handle has been created, with parameters set to default values.

Note that this operation saves only the memory contents of a tracker: stored facial appearances, identifiers, and names. The parameters of a tracker are *not* saved. Moreover, the internal state of face tracking is not saved as well. It means that if, during the main loop (where you call [FSDK_FeedFrame](#)), you save the tracker to a file, and then immediately reload it, such an operation will disrupt face tracking. Because of this, the later recognition results you receive will be different (compared to when such an operation was not done), and the parameters will be reset to defaults. Also, you will not be able to receive face position, eye coordinates, facial feature coordinates, or get the list of similar identifiers immediately after

loading. However, after the next [FSDK_FeedFrame](#) call, face tracking resumes, and the aforementioned functions operate normally.

Recognition Performance

The performance of face recognition (i.e. how often a subject is recognized, and how often different subjects are confused) is controlled with the Threshold and MemoryLimit parameters. The higher the Threshold parameter (and the lower the MemoryLimit parameter), the less often a subject will be recognized, and the less often confusions will occur.

Performance measures

Tracker API employs two performance measures: false acceptance rate (**FAR**) and recognition rate (**R**). **FAR** measures the rate of confusing different subjects (that is, assigning different subjects with equal identifier values) during a certain number of *storage events*, once the memory becomes full. **R** measures the rate of recognizing a person after tagging, once all available memory is full.

Understanding storage events

When calculating **FAR**, one could just count how often false acceptances occur during a certain time interval (for example, an hour). However, such a measure will vary greatly across different kinds of video footage.

For example, in an office setting, when subjects are sitting at their desks, and change their positions or facial expressions rather slowly, almost every frame will be very similar to the previous one. Therefore, the API will store novel facial appearances at a slow pace. If there were no false acceptances on a previous frame, they are very unlikely to occur on the next; therefore we expect false acceptances to occur rather rarely.

On the other hand, in an active setting (when many novel subjects appear in front of the camera, move around, and disappear from view), we expect the system to store novel facial appearances quite often, because many subjects appears at previously unseen views. Therefore, we expect false acceptances to occur more often, because of the faster pace of the video.

To employ a rate that is meaningful in both settings, we instead measure time not in seconds, but in *storage events*. For example, in the office setting, at 12 frames per second, we may get only 400 storage events during an hour, and in the active setting we may get 3600 *storage events* during an hour. We measure **FAR** at an interval of 2000 *storage events*, which could be roughly equal to 5 hours of a hypothetical less active setting, or 32 minutes of an active setting. It is important to note that as facial appearances of a subject accumulate, the rate of *storage events* will slow down, since there will be fewer novel facial appearances.

How to measure your rate of storage events

To measure the rate of storage events in your setting, call [FSDK_GetTrackerParameter](#) with the MemorySize parameter during the main loop. Each time a storage event occurs, the MemorySize parameter increases. As your video progresses, you may calculate how much time will be needed to reach 2000 storage events. Note that when the memory is full, storage events themselves still occur, but nothing is stored; this does not mean that **FAR** becomes zero. You should estimate the rate of storage events before the memory is full.

Understanding FAR

FAR is the rate of assigning different subjects with equal identifier values. The rate tells how often a certain subject (say, John) will be falsely accepted as any other subject. For example, if **FAR** is 0.001, John might expect a 0.001 probability of being falsely accepted as some other subject. However, if we have 10 subjects in the system, such a rate applies to every one of them. Therefore, it is practical to know the rate of falsely accepting at least two subjects among any of them. Such a rate can be calculated as

$$1 - (1 - \mathbf{FAR})^{N*(N-1)/2}$$

where N is the number of subjects. For example, at **FAR**=0.001, N=10, we have a 4.4% rate that at least one false acceptance will occur during the 2000 storage events considered. To have a 1% rate with 10 subjects, **FAR** should not exceed 0.0003.

Understanding R

R is the rate of recognizing a subject after it was tagged a single time, and all memory available for a subject becomes full. A subject is successfully recognized, if its name is present among the names returned by [FSDK_GetAllNames](#). R is measured from 0 to 1, which translates to recognition in 0% and 100%, respectively, of frames received by [FSDK_FeedFrame](#).

R depends mainly on the amount of memory available for each subject. For example, if there are 30 subjects in your system, and you allow 20 units of memory for each subject, your memory limit should be (30+1)*20=620.

Choosing Threshold value

To choose the Threshold value, refer to the tables below. You should consider the maximum number of subjects to be tagged within your system, and the maximum memory per subject.

Generally, the higher the MemoryLimit is set, the higher the FAR will be (once all available memory has been used).

Note that higher Threshold values together with a higher memory amount allow higher recognition rate *only* when enough facial appearances of an identifier have been accumulated. If there are sudden changes in facial appearance (due to low frame rate or environmental factors, for example), it may require more time to capture enough facial appearances with a higher Threshold value.

The tables below show the expected false acceptance rate and recognition rate.

False Acceptance Rate at Threshold and MemoryLimit

Threshold	MemoryLimit					
	350	700	1750	3500	5250	7500
0.992000	0.000081	0.000130	0.000231	0.000266	0.000277	0.000277
0.993141	0.000066	0.000107	0.000183	0.000209	0.000216	0.000216
0.994283	0.000062	0.000089	0.000144	0.000166	0.000170	0.000170
0.995424	0.000052	0.000068	0.000101	0.000114	0.000118	0.000118
0.996566	0.000042	0.000050	0.000072	0.000077	0.000081	0.000081
0.997707	0.000036	0.000040	0.000054	0.000055	0.000056	0.000056
0.998849	0.000030	0.000034	0.000045	0.000039	0.000039	0.000039

Threshold	MemoryLimit					
	350	700	1750	3500	5250	7500
0.999990	0.000002	0.000007	0.000009	0.000012	0.000014	0.000023

Recognition Rate at Threshold and Memory per subject

Threshold	Memory per subject			
	5	10	15	21
0.992000	0.995	0.999	0.999	0.999
0.993141	0.994	0.999	0.999	0.999
0.994283	0.993	0.998	0.999	0.999
0.995424	0.991	0.998	0.998	0.998
0.996566	0.986	0.997	0.997	0.997
0.997707	0.978	0.995	0.996	0.996
0.998849	0.956	0.986	0.988	0.988
0.999990	0.073	0.087	0.107	0.138

For example, let us assume that you have 30 subjects in an office setting, your frame rate is 12 per second, and you decide to allow 21 units of memory per subject. Therefore, your memory limit is $(30+1)*21 = 651$ (see the formula in the [Memory available for each subject](#) section). You decide to have a FAR of 0.000050 and calculate that with 30 subjects, there will be 2.2% rate that a subject will be given with an ID of any other subject (see the formula in the [Understanding FAR](#) section) during 2000 storage events (approximately 5 hours in an office setting). To have a FAR of 0.000050 with MemoryLimit=700 (the value closest to 651 in the table), you choose Threshold=0.996566. You note that at such a threshold and 21 units of memory per subject, you have a 0.997 recognition rate (meaning subjects will be recognized in 99.7% of frames in the video).

Note: it is not recommended to use Threshold higher than 0.999, since it will make Tracker API recognize faces less often.

Gender, Age and Facial Expression Recognition

The API allows for identifying gender and age of a face and its expressions by using the [FSDK_GetTrackerFacialAttribute](#) function.

To detect gender, you need to set the DetectGender tracking parameter to true. The function returns confidence levels for each gender (male and female) in the output string. You can parse this string using the [FSDK_GetValueConfidence](#) function.

To detect age, you need to set the DetectAge tracking parameter to true. The function then returns the age of the face in the output string.

To detect expression, you need to set the DetectExpression tracking parameter to true. The function returns confidence levels for each expression (if a smile is present and if the eyes are open or closed) in the output string. You can parse this string using the [FSDK_GetValueConfidence](#) function.

The confidence level for each attribute returned by the [FSDK_GetTrackerFacialAttribute](#) function, varies from 0 to 1 (except the “Age” attribute—for which the age itself—not a confidence level, is returned). When recognizing gender, you may assume that the recognized gender will be the one with the higher confidence level.

If your system should respond to the particular gender of a novel subject (for example, when advertising separate products for male and female visitors), consider waiting for about a second after the subject has first appear, for the gender to be recognized with higher accuracy. You may also consider responding when the confidence level is not just merely than 0.5, but exceeds a certain threshold (for example, 0.7 or 0.9, which translate to 70% or 90% accuracy).

If your system should respond to the particular expression of a subject (for example, taking a picture only when a person smiles and the eyes are open), consider waiting for about 0.5 seconds after the subject has appeared. To find out if the expression is present, it is usually optimal to compare the confidence in the attribute value with the 0.5 threshold (i.e., if the confidence in the “Smile” value is greater than 0.5, the person smiles, and if the confidence in the “EyesOpen” value is greater than 0.5, the eyes are open). You may use a higher threshold for greater certainty, but in this case some expressions may not be detected.

Note that gender, age and expression recognition requires the detection of facial features, so facial features will be detected regardless of the DetectFacialFeatures parameter value. This will decrease the frame rate. You might consider setting the RecognizeFaces parameter to false if you only need to detect gender, age or expressions and do not need recognition of the subjects’ identities, which will increase the frame rate.

Face, Eye and Facial Feature Tracking

Tracker API supports the tracking of face, eye centers, and facial features in addition to the recognition of a subject’s identity. You need to use the [FSDK_GetTrackerFacePosition](#), [FSDK_GetTrackerEyes](#) and [FSDK_GetTrackerFacialFeatures](#) to retrieve the corresponding coordinates. You also need to set the parameter DetectEyes or DetectFacialFeatures to true when tracking eyes or facial features, respectively. Tracker API perform smoothing of facial features (see the SmoothFacialFeatures parameter).

When you only need to track faces, and do not need to recognize subjects’ identities, you can disable face recognition to improve performance. To accomplish that, you need to set the RecognizeFaces parameter to false.

Counting the number of people

You should not estimate the amount of people the system observed based on the values of the identifiers, since some of they may have been merged with others. Instead, you may retain all the ID values returned by Tracker API, and at the point when the number of people should be estimated, you should replace each ID with the value returned by the [FSDK_GetIDReassignment](#) function. Then, you can count the amount of different identifiers in the list. Note that if memory limit is approached, some untagged identifiers may be purged, and the amount of people may be overestimated. See the [User Interaction with the System](#) section for details.

If each subject captured by the camera appears only once, you may consider not determining the subject’s identity (set RecognizeFaces to false). Then, the value of the ID returned by the API will be equal to the total number of continuous facial sequences, or approximately the number of people appeared in front of the camera.

Thread Safety

All tracker functions are thread safe. Note that you should avoid calling [FSDK_FeedFrame](#) simultaneously on the same tracker and camera (the `CameraIdx` parameter) from several threads, since it will disrupt the [FSDK_GetTrackerEyes](#), [FSDK_GetTrackerFacialFeatures](#), [FSDK_GetTrackerFacePosition](#), [FSDK_GetTrackerFacialAttribute](#), [FSDK_GetSimilarIDCount](#), [FSDK_GetSimilarIDList](#) and [FSDK_GetAllNames](#) functions. The reason is that the ID received from [FSDK_FeedFrame](#) must be passed to these functions before the next [FSDK_FeedFrame](#) is executed with the following frame; otherwise these functions may not perform correctly.

FSDK_CreateTracker Function

Creates a new tracker handle to be passed to other Tracker API functions.

C++ Syntax:

```
int FSDK_CreateTracker(HTracker * Tracker);
```

Delphi Syntax:

```
function FSDK_CreateTracker(Tracker: PHTracker): integer;
```

C# Syntax:

```
int FSDK.CreateTracker(ref int Tracker);
```

VB Syntax:

```
Function FSDKVB_CreateTracker(ByRef Tracker As Long) As Long
```

Java and Android Syntax:

```
int FSDK.CreateTracker(HTracker Tracker);
```

Parameters:

Tracker – pointer to the integer variable that will store the created tracker handle.

Return Value:

Returns `FSDKE_OK` if successful.

FSDK_FreeTracker Function

Frees a tracker handle. The handle becomes invalid, and all memory associated with it is released. You should not pass the tracker handle to any other Tracker API functions after the handle was freed.

C++ Syntax:

```
int FSDK_FreeTracker(HTracker Tracker);
```

Delphi Syntax:

```
function FSDK_FreeTracker(Tracker: HTracker): integer;
```

C# Syntax:

```
int FSDK.FreeTracker(int Tracker);
```

VB Syntax:

```
Function FSDKVB_FreeTracker (ByVal Tracker As Long) As Long
```

Java and Android Syntax:

```
int FSDK.FreeTracker(HTracker Tracker);
```

Parameters:

Tracker – handle of the tracker to be freed.

Return Value:

Returns FSDKE_OK if successful.

FSDK_ClearTracker Function

Clears the content of a tracker, releasing all its memory. The tracker handle stays valid. The parameters are reset to their default values, so if you just need to clear the tracker's memory, consider setting the parameters with the FSDK_SetTrackerParameter or the FSDK_SetTrackerMultipleParameters function again.

C++ Syntax:

```
int FSDK_ClearTracker(HTracker Tracker);
```

Delphi Syntax:

```
function FSDK_ClearTracker(Tracker: HTracker): integer;
```

C# Syntax:

```
int FSDK.ClearTracker(int Tracker);
```

VB Syntax:

```
Function FSDKVB_ClearTracker (ByVal Tracker As Long) As Long
```

Java and Android Syntax:

```
int FSDK.ClearTracker(HTracker Tracker);
```

Parameters:

Tracker – handle of the tracker to be cleared.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetTrackerParameter Function

Sets the parameter of a tracker. See the [Tracker Parameters](#) section for details.

C++ Syntax:

```
int FSDK_SetTrackerParameter(HTracker Tracker, const char *  
ParameterName, const char * ParameterValue);
```

Delphi Syntax:

```
function FSDK_SetTrackerParameter(Tracker: HTracker;  
ParameterName, ParameterValue: PAnsiChar): integer;
```

C# Syntax:

```
int FSDK.SetTrackerParameter(int Tracker, string  
ParameterName, string ParameterValue);
```

VB Syntax:

```
Function FSDKVB_SetTrackerParameter(ByVal Tracker As Long,  
ByVal ParameterName As String, ByVal ParameterValue As String)  
As Long
```

Java and Android Syntax:

```
int FSDK.SetTrackerParameter(HTracker Tracker, String  
ParameterName, String ParameterValue);
```

Parameters:

Tracker – handle of the tracker to have parameters set parameters.

ParameterName – name of the parameter to be set.

ParameterValue – value of the parameter.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetTrackerMultipleParameters Function

Sets multiple parameters of a tracker. The parameters and their values are specified in the following format:

```
"Parameter1=Value1[;Parameter2=Value2[;...]]"
```

See the [Tracker Parameters](#) section for details.

C++ Syntax:

```
int FSDK_SetTrackerMultipleParameters(HTracker Tracker, const  
char * Parameters, int * ErrorPosition);
```

Delphi Syntax:

```
function FSDK_SetTrackerMultipleParameters(Tracker: HTracker;  
Parameters: PAnsiChar; ErrorPosition: PInteger): integer;
```

C# Syntax:

```
int FSDK.SetTrackerMultipleParameters(int Tracker, string  
Parameters, ref int ErrorPosition);
```

VB Syntax:

```
Function FSDKVB_SetTrackerMultipleParameters(ByVal Tracker As  
Long, ByVal Parameters As String, ByRef ErrorPosition As Long)  
As Long
```

Java and Android Syntax:

```
int FSDK.SetTrackerMultipleParameters(HTracker Tracker, String Parameters, IntByReference ErrorPosition);
```

Parameters:

Tracker – handle of the tracker to have parameters set.

Parameters – string containing the parameters and corresponding values to be set.

ErrorPosition – pointer to the integer variable that will receive the position of the character that caused syntax error in the string.

Return Value:

Returns FSDK_OK if successful. In case of syntax error returns FSDK_SYNTAX_ERROR and sets the value of the ErrorPosition variable.

Example:

```
int err = 0;
FSDK_SetTrackerMultipleParameters(tracker,
"HandleArbitraryRotations=false;
DetermineFaceRotationAngle=false; InternalResizeWidth=100;
FaceDetectionThreshold=5;", &err);
```

FSDK_GetTrackerParameter Function

Retrieves the value of a tracker parameter. See the [Tracker Parameters](#) section for details.

C++ Syntax:

```
int FSDK_GetTrackerParameter(HTracker Tracker, const char *
ParameterName, char * ParameterValue, long long
MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_GetTrackerParameter(Tracker: HTracker;
ParameterName, ParameterValue: PAnsiChar; MaxSizeInBytes:
int64): integer;
```

C# Syntax:

```
int FSDK.GetTrackerParameter(int Tracker, string
ParameterName, out string ParameterValue, long MaxSizeInBytes)
```

VB Syntax:

```
Function FSDKVB_GetTrackerParameter(ByVal Tracker As Long,
ByVal ParameterName As String, ByRef ParameterValue As String,
ByVal MaxSizeInBytes As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetTrackerParameter(HTracker Tracker, String
ParameterName, String ParameterValue[], long MaxSizeInBytes);
```

Parameters:

Tracker – handle of the tracker whose parameter value is desired.

ParameterName – name of the parameter to be retrieved.

ParameterValue – pointer to the output null-terminated string that will store the value of the parameter.

MaxSizeInBytes – amount of memory allocated for the output string.

Return Value:

Returns FSDK_OK if successful. Returns FSDK_INSUFFICIENT_BUFFER_SIZE if there is not enough room to store the output string; however, the output string still fills up all the space available.

FSDK_FeedFrame Function

Processes a video frame according to tracker's parameters, and returns the identifiers of the tracked faces. See the [Understanding Identifiers](#), [Tracker Memory](#) and [Tracker Parameters](#) sections for details.

C++ Syntax:

```
int FSDK_FeedFrame(HTracker Tracker, long long CameraIdx,
HImage Image, long long * FaceCount, long long * IDs, long
long MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_FeedFrame(Tracker: HTracker; CameraIdx: int64;
Image: HImage; FaceCount: PInt64; IDs: PIDArray;
MaxSizeInBytes: int64): integer;
```

C# Syntax:

```
int FSDK.FeedFrame(int Tracker, long CameraIdx, int Image, ref
long FaceCount, out long[] IDs, long MaxSizeInBytes)
```

VB Syntax:

```
Function FSDKVB_FeedFrame(ByVal Tracker As Long, ByVal
CameraIdx As Currency, ByVal Image As Long, ByRef FaceCount As
Currency, ByRef IDs As Currency, ByVal MaxSizeInBytes As
Currency) As Long
```

Java and Android Syntax:

```
int FSDK.FeedFrame(HTracker Tracker, long CameraIdx, HImage
Image, long FaceCount[], long IDs[], long MaxSizeInBytes);
```

Parameters:

Tracker – handle of the tracker in which to process the frame.

CameraIdx – index of the camera; should be equal to 0 in the current release.

Image – the HImage handle of the video frame to process.

FaceCount – address of the 64-bit integer value that will receive the count of faces tracked in the current frame.

IDs – address of the array of 64-bit integer values that will receive the identifiers of the tracked faces.

MaxSizeInBytes – amount of memory allocated for the IDs array.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GetTrackerEyes Function

Retrieves the coordinates of the eye centers of a tracked face. The function accepts the identifier returned by [FSDK_FeedFrame](#). This identifier should be passed to [FSDK_GetTrackerEyes](#) before the next call of [FSDK_FeedFrame](#) using the same tracker.

For the function to return the eye center coordinates, at least one of the parameters DetectEyes, DetectFacialFeatures, RecognizeFaces, DetectGender, DetectAge or DetectExpression must be set to true.

C++ Syntax:

```
int FSDK_GetTrackerEyes(HTracker Tracker, long long CameraIdx,
long long ID, FSDK_Features * FacialFeatures);
```

Delphi Syntax:

```
function FSDK_GetTrackerEyes(Tracker: HTracker; CameraIdx, ID:
int64; FacialFeatures: PFSDK_Features): integer;
```

C# Syntax:

```
int FSDK.GetTrackerEyes(int Tracker, long CameraIdx, long ID,
out TPoint[] FacialFeatures)
```

VB Syntax:

```
Function FSDKVB_GetTrackerEyes(ByVal Tracker As Long, ByVal
CameraIdx As Currency, ByVal ID As Currency, ByRef
FacialFeatures As TPoint) As Long
```

Java Syntax:

```
int FSDK.GetTrackerEyes(HTracker Tracker, long CameraIdx, long
ID, FSDK_Features.ByReference FacialFeatures);
```

Android Syntax:

```
int FSDK.GetTrackerEyes(HTracker Tracker, long CameraIdx, long
ID, FSDK_Features FacialFeatures);
```

Parameters:

Tracker – handle of the tracker where the coordinates of the eye centers will be retrieved.

CameraIdx – index of the camera; should be equal to 0 in the current release.

ID – identifier of the subject returned by [FSDK_FeedFrame](#), whose eye center coordinates will be received.

FacialFeatures – pointer to the FSDK_Features variable that will receive the eye center coordinates.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID was not returned by the previous [FSDK_FeedFrame](#) call.

Returns FSDK_ATTRIBUTE_NOT_DETECTED if eye centers were not tracked on the previous [FSDK_FeedFrame](#) call.

FSDK_GetTrackerFacialFeatures Function

Retrieves the coordinates of a tracked face's features. The function accepts the identifier returned by [FSDK_FeedFrame](#). This identifier should be passed to [FSDK_GetTrackerFacialFeatures](#) before the next call of [FSDK_FeedFrame](#) with the same tracker.

For the function to return the facial feature coordinates, either of the parameters DetectFacialFeatures, DetectGender, DetectAge or DetectExpression should be set to true. See the [Tracker Parameters](#) section for details.

C++ Syntax:

```
int FSDK_GetTrackerFacialFeatures(HTracker Tracker, long long CameraIdx, long long ID, FSDK_Features * FacialFeatures);
```

Delphi Syntax:

```
function FSDK_GetTrackerFacialFeatures(Tracker: HTracker; CameraIdx, ID: int64; FacialFeatures: PFSDK_Features): integer;
```

C# Syntax:

```
int FSDK.GetTrackerFacialFeatures(int Tracker, long CameraIdx, long ID, out TPoint[] FacialFeatures)
```

VB Syntax:

```
Function FSDKVB_GetTrackerFacialFeatures(ByVal Tracker As Long, ByVal CameraIdx As Currency, ByVal ID As Currency, ByRef FacialFeatures As TPoint) As Long
```

Java Syntax:

```
int FSDK.GetTrackerFacialFeatures(HTracker Tracker, long CameraIdx, long ID, FSDK_Features.ByReference FacialFeatures);
```

Android Syntax:

```
int FSDK.GetTrackerFacialFeatures(HTracker Tracker, long CameraIdx, long ID, FSDK_Features FacialFeatures);
```

Parameters:

Tracker – handle of the tracker from which to retrieve the facial feature coordinates.

CameraIdx – index of the camera; should be equal to 0 in the current release.

ID – identifier of the subject returned by [FSDK_FeedFrame](#), whose facial feature coordinates will be received.

FacialFeatures – pointer to the [FSDK_Features](#) variable to receive facial feature coordinates.

Return Value:

Returns FSDK_OK if successful. Returns FSDK_ID_NOT_FOUND if the specified ID was not returned by the previous [FSDK_FeedFrame](#) call.

Returns `FSDK_ATTRIBUTE_NOT_DETECTED` if facial features were not tracked on the previous [FSDK_FeedFrame](#) call.

FSDK_GetTrackerFacePosition Function

Retrieves the position of a tracked face. The function accepts the identifier returned by [FSDK_FeedFrame](#). This identifier should be passed to [FSDK_GetTrackerFacePosition](#) before the next call of [FSDK_FeedFrame](#) with the same tracker.

C++ Syntax:

```
int FSDK_GetTrackerFacePosition(HTracker Tracker, long long CameraIdx, long long ID, TFacePosition * FacePosition);
```

Delphi Syntax:

```
function FSDK_GetTrackerFacePosition(Tracker: HTracker; CameraIdx, ID: int64; FacePosition: PFacePosition): integer;
```

C# Syntax:

```
int FSDK.GetTrackerFacePosition(int Tracker, long CameraIdx, long ID, ref TFacePosition FacePosition);
```

VB Syntax:

```
Function FSDKVB_GetTrackerFacePosition(ByVal Tracker As Long, ByVal CameraIdx As Currency, ByVal ID As Currency, ByRef facePosition As TFacePosition) As Long
```

Java Syntax:

```
int FSDK.GetTrackerFacePosition(HTracker Tracker, long CameraIdx, long ID, TFacePosition.ByReference FacePosition);
```

Android Syntax:

```
int FSDK.GetTrackerFacePosition(HTracker Tracker, long CameraIdx, long ID, TFacePosition FacePosition);
```

Parameters:

Tracker – handle of the tracker from which to retrieve the face position.

CameraIdx – index of the camera; should be equal to 0 in the current release.

ID – identifier of the subject returned by `FSDK_FeedFrame` whose face position will be received.

FacePosition – pointer to the `TFacePosition` variable that will receive the face position.

Return Value:

Returns `FSDK_OK` if successful. Returns `FSDK_ID_NOT_FOUND` if the specified ID was not returned by the previous [FSDK_FeedFrame](#) call.

FSDK_GetTrackerFacialAttribute Function

Given an attribute of a tracked face, retrieves its Values and their Confidences. The function accepts the identifier returned by [FSDK_FeedFrame](#). This identifier should be passed to

[FSDK_GetTrackerFacialAttribute](#) before the next call of [FSDK_FeedFrame](#) with the same tracker.

The function allows for detecting gender when provided with the “Gender” attribute name, for detecting age when provided with the “Age” attribute name and for detecting expression when provided with the “Expression” attribute name. Refer to the [FSDK_DetectFacialAttributeUsingFeatures](#) function description for details on attributes, their Values and Confidences.

C++ Syntax:

```
int FSDK_GetTrackerFacialAttribute(HTracker Tracker, long long CameraIdx, long long ID, const char * AttributeName, char * AttributeValues, long long MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_GetTrackerFacialAttribute(Tracker: HTracker; CameraIdx, ID: int64; AttributeName, AttributeValues: PAnsiChar; MaxSizeInBytes: int64): integer;
```

C# Syntax:

```
int FSDK.GetTrackerFacialAttribute(int Tracker, long CameraIdx, long ID, string AttributeName, out string AttributeValues, long MaxSizeInBytes);
```

VB Syntax:

```
Function FSDKVB_GetTrackerFacialAttribute(ByVal Tracker As Long, ByVal CameraIdx As Currency, ByVal ID As Currency, ByVal AttributeName As String, ByRef AttributeValues As String, ByVal MaxSizeInBytes As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetTrackerFacialAttribute(HTracker Tracker, long CameraIdx, long ID, String AttributeName, String AttributeValues[], long MaxSizeInBytes);
```

Parameters:

Tracker – handle of the tracker whose attribute will be retrieved.

CameraIdx – index of the camera; should be equal to 0 in the current release.

ID – identifier of a subject returned by [FSDK_FeedFrame](#) whose attribute will be retrieved.

AttributeName – name of the attribute.

AttributeValues – pointer to the null-terminated string that will receive the attribute Values and their Confidences.

MaxSizeInBytes – amount of memory allocated for the output string.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID was not returned by the previous [FSDK_FeedFrame](#) call. Returns FSDKE_ATTRIBUTE_NOT_DETECTED if the specified attribute was not detected on the previous [FSDK_FeedFrame](#) call. Returns FSDKE_UNKNOWN_ATTRIBUTE if the specified attribute name is not supported. Returns FSDKE_INSUFFICIENT_BUFFER_SIZE

if there is not enough room to store the output string; however, the output string still fills up all the space available.

FSDK_LockID Function

Locks an identifier. When an identifier is locked, at least one facial appearance of an identifier will not be deleted during any possible purge. You should call this function before the [FSDK_SetName](#) function. The function has no effect on identifiers which were already tagged with a name. The call should be usually paired with [FSDK_UnlockID](#) call. When the user does not set a name to a locked identifier, unlocking it allows it to become purged if necessary for memory efficient memory use.

See the [Locking Identifiers](#) section for details. You may call this function with any identifier regardless of when it was returned as long as it remains present in the tracker memory.

C++ Syntax:

```
int FSDK_LockID(HTracker Tracker, long long ID);
```

Delphi Syntax:

```
function FSDK_LockID(Tracker: HTracker; ID: int64): integer;
```

C# Syntax:

```
int FSDK.LockID(int Tracker, long ID);
```

VB Syntax:

```
Function FSDKVB_LockID(ByVal Tracker As Long, ByVal ID As  
Currency) As Long
```

Java and Android Syntax:

```
int FSDK.LockID(HTracker Tracker, long ID);
```

Parameters:

Tracker – handle of the tracker in which to lock an identifier.

ID – identifier of the subject to lock.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID is not present in the tracker memory.

FSDK_UnlockID Function

Unlocks the ID so it may be purged. You should call this function after the [FSDK_LockID](#) call. The function has no effect on identifiers which were already tagged with a name.

See the [Locking identifiers](#) section for details. You may call this function with any identifier regardless of when it was returned, as long as it is present in the tracker memory.

C++ Syntax:

```
int FSDK_UnlockID(HTracker Tracker, long long ID);
```

Delphi Syntax:

```
function FSDK_UnlockID(Tracker: HTracker; ID: int64): integer;
```

C# Syntax:

```
int FSDK.UnlockID(int Tracker, long ID);
```

VB Syntax:

```
Function FSDKVB_UnlockID(ByVal Tracker As Long, ByVal ID As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.UnlockID(HTracker Tracker, long ID);
```

Parameters:

Tracker – handle of the tracker in which to unlock an identifier.

ID – identifier of the subject to unlock.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID is not present in the tracker memory.

FSDK_PurgeID Function

Removes all facial appearances of the ID from the tracker memory. You must call this function if there was a false acceptance (see the [Dealing with false acceptances](#) section) or if you erroneously assigned equal names to different persons.

C++ Syntax:

```
int FSDK_PurgeID(HTracker Tracker, long long ID);
```

Delphi Syntax:

```
function FSDK_PurgeID(Tracker: HTracker; ID: int64): integer;
```

C# Syntax:

```
int FSDK.PurgeID(int Tracker, long ID);
```

VB Syntax:

```
Function FSDKVB_PurgeID(ByVal Tracker As Long, ByVal ID As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.PurgeID(HTracker Tracker, long ID);
```

Parameters:

Tracker – handle of the tracker in which to purge an identifier.

ID – identifier of the subject to purge.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID is not present in the tracker memory.

FSDK_GetName Function

Returns the name the identifier has been tagged with. You may call this function with any identifier regardless of when it was returned, as long as it is present in the tracker memory.

C++ Syntax:

```
int FSDK_GetName(HTracker Tracker, long long ID, char * Name, long long MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_GetName(Tracker: HTracker; ID: int64; Name: PAnsiChar; MaxSizeInBytes: int64): integer;
```

C# Syntax:

```
int FSDK.GetName(int Tracker, long ID, out string Name, long MaxSizeInBytes);
```

VB Syntax:

```
Function FSDKVB_GetName(ByVal Tracker As Long, ByVal ID As Currency, ByRef Name As String, ByVal MaxSizeInBytes As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetName(int Tracker, long ID, String Name[], long MaxSizeInBytes);
```

Parameters:

Tracker – handle of the tracker in which to retrieve the name.

ID – identifier of a subject to retrieve the name of.

Name – identifier of the subject whose name is to be retrieved.

MaxSizeInBytes – amount of memory allocated for the output string.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_INSUFFICIENT_BUFFER_SIZE if there is not enough room to store the output string; however, the output string still fills up all the space available. Returns FSDKE_ID_NOT_FOUND if the specified ID is not present in the tracker memory.

FSDK_SetName Function

Sets the name of an identifier. To erase the name tag, specify an empty name string. When erasing the name tag because of a false acceptance, or because you erroneously assigned equal names to different persons, you must also call the [FSDK_PurgeID](#) function (see the [Dealing with false acceptances](#) section). The function will unlock the identifier if the name is successfully set.

You may call this function with any identifier regardless of when it was returned, as long as it is present in the tracker memory.

C++ Syntax:

```
int FSDK_SetName(HTracker Tracker, long long ID, const char *
Name);
```

Delphi Syntax:

```
function FSDK_SetName(Tracker: HTracker; ID: int64; Name:
PAnsiChar): integer;
```

C# Syntax:

```
int FSDK.SetName(int Tracker, long ID, string Name);
```

VB Syntax:

```
Function FSDKVB_SetName(ByVal Tracker As Long, ByVal ID As
Currency, ByVal Name As String) As Long
```

Java and Android Syntax:

```
int FSDK.SetName(HTracker Tracker, long ID, String Name);
```

Parameters:

Tracker – handle of the tracker in which to set the name.

ID – identifier of the subject whose name is to be set.

Name – pointer to the null-terminated string containing the name of an identifier.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID is not present in the tracker memory.

Returns FSDKE_INSUFFICIENT_TRACKER_MEMORY_LIMIT if there is not enough room to store the identifier's facial appearances in memory. See the [Tracker Memory](#) section for details.

FSDK_GetIDReassignment Function

When provided with a subject's ID received on earlier frames, returns the new subject's ID if there was a merger. See the [Understanding Identifiers](#) section for details. If an identifier was not merged, the function returns the same ID value in the output variable. Note that the function does not return an error if an identifier is not present in the tracker memory; instead, the same ID value is returned in the output variable.

C++ Syntax:

```
int FSDK_GetIDReassignment(HTracker Tracker, long long ID,
long long * ReassignedID);
```

Delphi Syntax:

```
function FSDK_GetIDReassignment(Tracker: HTracker; ID: int64;
ReassignedID: PInt64): integer;
```

C# Syntax:

```
int FSDK.GetIDReassignment(int Tracker, long ID, ref long
ReassignedID);
```

VB Syntax:

```
Function FSDKVB_GetIDReassignment(ByVal Tracker As Long, ByVal ID As Currency, ByRef ReassignedID As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetIDReassignment(HTracker Tracker, long ID, long ReassignedID[]);
```

Parameters:

Tracker – handle of the tracker in which to get the reassigned ID value.

ID – identifier of the subject whose reassigned identifier is sought.

ReassignedID – pointer to the 64-bit integer value that will store the reassigned value of an identifier.

Return Value:

Returns FSDKE_OK if successful.

FSDK_GetSimilarIDCount Function

Returns the number of identifiers that are similar to a given identifier. The function accepts the identifier returned by [FSDK_FeedFrame](#). This identifier should be passed to [FSDK_GetSimilarIDCount](#) before the next call of [FSDK_FeedFrame](#) with the same tracker. See the [Understanding Identifiers](#) section for details.

C++ Syntax:

```
int FSDK_GetSimilarIDCount(HTracker Tracker, long long ID, long long * Count);
```

Delphi Syntax:

```
function FSDK_GetSimilarIDCount(Tracker: HTracker; ID: int64; Count: PInt64): integer;
```

C# Syntax:

```
int FSDK.GetSimilarIDCount(int Tracker, long ID, ref long Count);
```

VB Syntax:

```
FSDKVB_GetSimilarIDCount(ByVal Tracker As Long, ByVal ID As Currency, ByRef Count As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetSimilarIDCount(HTracker Tracker, long ID, long Count[]);
```

Parameters:

Tracker – handle of the tracker in which to retrieve the number of similar identifiers.

ID – identifier of the subject for which to return the number of similar identifiers.

Count – pointer to the 64-bit integer value that will store the number of similar identifiers.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID was not returned by the previous [FSDK_FeedFrame](#) call.

FSDK_GetSimilarIDList Function

Returns the list of identifiers that are similar to a given identifier. The function accepts the identifier returned by [FSDK_FeedFrame](#). This identifier should be passed to [FSDK_GetSimilarIDList](#) before the next call of [FSDK_FeedFrame](#) with the same tracker. See the [Understanding Identifiers](#) section for details.

C++ Syntax:

```
int FSDK_GetSimilarIDList(HTracker Tracker, long long ID, long long * SimilarIDList, long long MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_GetSimilarIDList(Tracker: HTracker; ID: int64; SimilarIDList: PIDArray; MaxSizeInBytes: int64): integer;
```

C# Syntax:

```
int FSDK.GetSimilarIDList(int Tracker, long ID, out long[] SimilarIDList, long MaxSizeInBytes)
```

VB Syntax:

```
Function FSDKVB_GetSimilarIDList(ByVal Tracker As Long, ByVal ID As Currency, ByRef SimilarIDList As Currency, ByVal MaxSizeInBytes As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetSimilarIDList(HTracker Tracker, long ID, long SimilarIDList[], long MaxSizeInBytes);
```

Parameters:

Tracker – handle of the tracker in which to get the list of similar identifiers.

ID – identifier of the subject for which to return the list of similar identifiers.

SimilarIDList – pointer to the array of 64-bit integer values that will store the list of similar identifiers.

MaxSizeInBytes – amount of memory allocated for the output array.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID was not returned by the previous [FSDK_FeedFrame](#) call. Returns FSDKE_INSUFFICIENT_BUFFER_SIZE if there is not enough room to store the output string; however, the output string still fills up all the space available.

FSDK_GetAllNames Function

Returns the list of names that an identifier can have. The function accepts the identifier returned by [FSDK_FeedFrame](#). This identifier should be passed to [FSDK_GetAllNames](#)

before the next call of [FSDK_FeedFrame](#) with the same tracker. See the [Understanding Identifiers](#) and [Dealing with false acceptances](#) sections for details.

The function returns all names that belong to a given identifier, and similar identifiers, separated by a semicolon. The output format is:

```
"Name1 [ ; Name2 [ ; ... ] ] "
```

You should call this function instead of [FSDK_GetName](#) whenever possible, and then parse the returned string for all returned names. Alternatively, you may implement the functionality of [FSDK_GetAllNames](#), calling [FSDK_GetName](#) on the given identifier, then [FSDK_GetSimilarIDCount](#) and [FSDK_GetSimilarIDList](#) to get the list of similar identifiers, then finally call [FSDK_GetName](#) on that list.

C++ Syntax:

```
int FSDK_GetAllNames(HTracker Tracker, long long ID, char *
Names, long long MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_GetAllNames(Tracker: HTracker; ID: int64; Names:
PAnsiChar; MaxSizeInBytes: int64): integer;
```

C# Syntax:

```
int FSDK.GetAllNames(int Tracker, long ID, out string Names,
long MaxSizeInBytes);
```

VB Syntax:

```
Function FSDKVB_GetAllNames(ByVal Tracker As Long, ByVal ID As
Currency, ByRef Names As String, ByVal MaxSizeInBytes As
Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetAllNames(HTracker Tracker, long ID, String
Names[], long MaxSizeInBytes);
```

Parameters:

Tracker – handle of the tracker in which to retrieve the names.

ID – identifier of the subject whose possible names are to be retrieved.

Names – pointer to the null-terminated string that will receive the possible names of an identifier.

MaxSizeInBytes – amount of memory allocated for the output string.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_ID_NOT_FOUND if the specified ID was not returned by the previous FSDK_FeedFrame call. Returns FSDKE_INSUFFICIENT_BUFFER_SIZE if there is not enough room to store the output string; however, the output string still fills up all the space available.

FSDK_SaveTrackerMemoryToFile Function

Saves the memory of a tracker to a file. Note that tracker parameters, along with its face tracking state, are not saved. See the [Saving and Loading Tracker Memory](#) section for details.

C++ Syntax:

```
int FSDK_SaveTrackerMemoryToFile(HTracker Tracker, const char * FileName);
```

Delphi Syntax:

```
function FSDK_SaveTrackerMemoryToFile(Tracker: HTracker; FileName: PAnsiChar): integer;
```

C# Syntax:

```
int FSDK.SaveTrackerMemoryToFile(int Tracker, string FileName);
```

VB Syntax:

```
Function FSDKVB_SaveTrackerMemoryToFile(ByVal Tracker As Long, ByVal FileName As String) As Long
```

Java and Android Syntax:

```
int FSDK.SaveTrackerMemoryToFile(HTracker Tracker, String FileName);
```

Parameters:

Tracker – handle of the tracker to save.

FileName – pointer to the null-terminated string containing the name of the file to which the tracker memory will be saved.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_IO_ERROR if an I/O error has occurred.

FSDK_LoadTrackerMemoryFromFile Function

Loads the memory of a tracker from a file. Note that tracker parameters, along with its face tracking state, are not loaded. See the [Saving and Loading Tracker Memory](#) section for details.

C++ Syntax:

```
int FSDK_LoadTrackerMemoryFromFile(HTracker * Tracker, const char * FileName);
```

Delphi Syntax:

```
function FSDK_LoadTrackerMemoryFromFile(Tracker: PHTracker; FileName: PAnsiChar): integer;
```

C# Syntax:

```
int FSDK.LoadTrackerMemoryFromFile(ref int Tracker, string FileName);
```

VB Syntax:

```
Function FSDKVB_LoadTrackerMemoryFromFile(ByRef Tracker As Long, ByVal FileName As String) As Long
```

Java and Android Syntax:

```
int FSDK.LoadTrackerMemoryFromFile(HTracker Tracker, String
FileName);
```

Parameters:

Tracker – pointer that will store the handle of the loaded tracker.

FileName – pointer to the null-terminated string containing the name of a file from which the tracker memory will be loaded.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_BAD_FILE_FORMAT if the file has unsupported format. Returns FSDKE_UNSUPPORTED_FILE_VERSION if the file was saved with Luxand FaceSDK of an unsupported version. Returns FSDKE_FILE_NOT_FOUND if there was an error opening the file. Returns FSDKE_IO_ERROR if an I/O error has occurred.

FSDK_GetTrackerMemoryBufferSize Function

Returns the size of a buffer (in bytes) needed to save the memory of a tracker.

C++ Syntax:

```
int FSDK_GetTrackerMemoryBufferSize(HTracker Tracker, long
long * BufSize);
```

Delphi Syntax:

```
function FSDK_GetTrackerMemoryBufferSize(Tracker: HTracker;
BufSize: PInt64): integer;
```

C# Syntax:

```
int FSDK.GetTrackerMemoryBufferSize(int Tracker, ref long
BufSize);
```

VB Syntax:

```
Function FSDKVB_GetTrackerMemoryBufferSize(ByVal Tracker As
Long, ByRef BufSize As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.GetTrackerMemoryBufferSize(HTracker Tracker, long
BufSize[]);
```

Parameters:

Tracker – handle of the tracker whose buffer size needs calculation.

BufSize – pointer to the 64-bit integer variable that will store the size of a buffer.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SaveTrackerMemoryToBuffer Function

Saves the memory of a tracker to a buffer. Note that tracker parameters, along with its face tracking state, are not saved. See the [Saving and Loading Tracker Memory](#) section for details.

C++ Syntax:

```
int FSDK_SaveTrackerMemoryToBuffer(HTracker Tracker, unsigned char * Buffer, long long MaxSizeInBytes);
```

Delphi Syntax:

```
function FSDK_SaveTrackerMemoryToBuffer(Tracker: HTracker; var Buffer; MaxSizeInBytes: int64): integer;
```

C# Syntax:

```
int FSDK.SaveTrackerMemoryToBuffer(int Tracker, out byte[] Buffer, long MaxSizeInBytes);
```

VB Syntax:

```
Function FSDKVB_SaveTrackerMemoryToBuffer(ByVal Tracker As Long, ByRef Buffer As Byte, ByVal MaxSizeInBytes As Currency) As Long
```

Java and Android Syntax:

```
int FSDK.SaveTrackerMemoryToBuffer(HTracker Tracker, byte Buffer[]);
```

Parameters:

Tracker – handle of the tracker to save.

Buffer – pointer to the buffer to which the tracker memory will be saved.

MaxSizeInBytes – amount of memory allocated for the output buffer, in bytes.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_INSUFFICIENT_BUFFER_SIZE if there is not enough room to store the output buffer.

FSDK_LoadTrackerMemoryFromBuffer Function

Loads the memory of a tracker from a buffer. Note that tracker parameters, along with its face tracking state, are not loaded. See the [Saving and Loading Tracker Memory](#) section for details.

C++ Syntax:

```
int FSDK_LoadTrackerMemoryFromBuffer(HTracker * Tracker, const unsigned char * Buffer);
```

Delphi Syntax:

```
function FSDK_LoadTrackerMemoryFromBuffer(Tracker: PHTracker; var Buffer): integer;
```

C# Syntax:

```
int FSDK.LoadTrackerMemoryFromBuffer(ref int Tracker, byte[]  
Buffer);
```

VB Syntax:

```
Function FSDKVB_LoadTrackerMemoryFromBuffer(ByRef Tracker As  
Long, ByRef Buffer As Byte) As Long
```

Java and Android Syntax:

```
int FSDK.LoadTrackerMemoryFromBuffer(HTracker Tracker, byte  
Buffer[]);
```

Parameters:

Tracker – pointer to store the handle of a loaded tracker.

Buffer – pointer to the buffer from which to load the tracker memory.

Return Value:

Returns FSDKE_OK if successful. Returns FSDKE_BAD_FILE_FORMAT if the file has unsupported format. Returns FSDKE_UNSUPPORTED_FILE_VERSION if the file was saved with Luxand FaceSDK of an unsupported version.

Multi-Core Support

The following FaceSDK functions use multiple CPU cores, thus speeding up the calculations:

```
FSDK_DetectEyes,  
FSDK_DetectEyesInRegion,  
FSDK_DetectFace,  
FSDK_DetectMultipleFaces,  
FSDK_DetectFacialFeatures,  
FSDK_DetectFacialFeaturesInRegion,  
FSDK_GetFaceTemplate,  
FSDK_GetFaceTemplateInRegion,  
FSDK_GetFaceTemplateUsingFeatures,  
FSDK_GetFaceTemplateUsingEyes,  
FSDK_FeedFrame.
```

By default, these functions use all available processor cores. To get the number of processor cores used, call the [FSDK_GetNumThreads](#) function. To limit the number of processor cores used, call the [FSDK_SetNumThreads](#) function. Calling `FSDK_SetNumThreads(1)` will disable multi-core support.

Note that each of these functions forks into a number of threads on each call. It is not recommended to use nested parallelism when calling these functions; if you need nested parallelism, you may limit the number of threads with the [FSDK_SetNumThreads](#) functions. For example, if your application runs in several threads, and each thread executes [FSDK_DetectFace](#) (which uses all available cores), this is acceptable; however, if each thread forks into several threads, each executing [FSDK_DetectFace](#), this could potentially reach the limit of resources available.

It is safe to use extensions for parallel computation (like OpenMP) with the above FaceSDK functions, if they are executed from a single thread. For example, the following C++ sample code is acceptable:

```
#pragma omp parallel for
    for (int i = 0; i < 100; i++)
        FSDK_DetectFace(...);
```

However, if your application forks into multiple threads, it is not recommended to execute the above FaceSDK functions within OpenMP statements in such threads. If you must, consider limiting the number of cores used by FaceSDK with the [FSDK_SetNumThreads](#) function.

FSDK_GetNumThreads Function

Retrieves the number of processor cores used by FaceSDK.

C++ Syntax:

```
int FSDK_GetNumThreads(int * Num);
```

Delphi Syntax:

```
function FSDK_GetNumThreads(Num: PInteger): integer;
```

C# Syntax:

```
int FSDK.GetNumThreads(ref int Num);
```

VB Syntax:

```
Function FSDKVB_GetNumThreads(ByRef Num As Long) As Long
```

Java and Android Syntax:

```
int FSDK.GetNumThreads(int Num[]);
```

Parameters:

Num – pointer to an integer value to receive the number of threads used by FaceSDK.

Return Value:

Returns FSDKE_OK if successful.

FSDK_SetNumThreads Function

Sets the number of processor cores to be used by FaceSDK. If you set the number of cores to 1, support for multiple cores will be disabled, and the SDK will use only a single processor core.

C++ Syntax:

```
int FSDK_SetNumThreads(int Num);
```

Delphi Syntax:

```
function FSDK_SetNumThreads(Num: integer): integer;
```

C# Syntax:

```
int FSDK.SetNumThreads(int Num);
```

VB Syntax:

```
Function FSDKVB_SetNumThreads (ByVal Num As Long) As Long
```

Java and Android Syntax:

```
int FSDK.SetNumThreads (int Num);
```

Parameters:

Num – the number of cores to be used by FaceSDK.

Return Value:

Returns FSDKE_OK if successful.

Thread Safety

This chapter describes using FaceSDK in applications that execute FaceSDK functions from multiple threads. If your program runs in a single thread (by default, this happens in almost all environments), you can skip this chapter.

Most FaceSDK functions are safe for multithreaded operations. The functions not guaranteed to be thread-safe are

```
FSDK_Initialize,  
FSDK_Finalize,  
FSDK_ActivateLibrary,  
FSDK_GetLicenseInfo,  
FSDK_GetHardware_ID,  
FSDK_SetNumThreads.
```

When working with cameras in multiple threads on Windows platforms, make sure that each thread calls the [FSDK_InitializeCapturing](#) function and the [FSDK_FinalizeCapturing](#) function when the thread is done. The following functions are thread safe (on all supported platforms) given that no different threads are simultaneously accessing the same camera handle:

```
FSDK_GetVideoFormatList,  
FSDK_FreeVideoFormatList,  
FSDK_SetVideoFormat,  
FSDK_OpenVideoCamera,  
FSDK_CloseVideoCamera,  
FSDK_GrabFrame.
```

The following functions set global parameters that have effect on each thread:

```
FSDK_SetFaceDetectionParameters,  
FSDK_SetFaceDetectionThreshold,  
FSDK_SetJpegCompressionQuality,  
FSDK_SetCameraNaming,  
FSDK_SetHTTPProxy,  
FSDK_SetNumThreads.
```

Note that HImage is safe only for multiple simultaneous reads or single write. Do not read from (e.g. with [FSDK_DetectFace](#)) and write to (e.g. with [FSDK_FreeImage](#)) the same HImage handle at one time.

For more information on thread safety of Tracker API, see the [Thread Safety](#) section in the Tracker API chapter.

Migration

Migration from FaceSDK 7.2, 7.2.1 to FaceSDK 8.0

For compatibility with the latest Xcode versions the libfsdk-static.a and libfsdk-static_64.a libraries for iOS have been joined into a single library libfsdk-static.a.

NET applications now use FaceSDK.NET.dll built from the samples\advanced\.NET wrapper source, so adding the appropriate binary file (facesdk.dll, libfsdk.dylib or libfsdk.so) is required.

The minimum supported iOS version is 9.0.

The minimum supported Android version is 5.0.

The minimum supported Windows version is Windows 7.

The minimum supported macOS version is 10.13.

Migration from FaceSDK 7.1 to FaceSDK 7.2, 7.2.1

InternalResizeWidth parameter values larger than 512 are now supported to allow for the detection of even small faces on high-resolution images.

JPEG images are now automatically rotated on load using the EXIF data.

The minimum supported Linux versions are CentOS/RHEL 7. Older Linux versions are no longer supported.

The minimum supported Windows version is Windows Vista.

Migration from FaceSDK 6.5.1 to FaceSDK 7.0, 7.1

Face Detection

Version 7.0 introduces a new face detection engine, which is more accurate when detecting faces that are rotated out of plane, blurred, backlit, or in low lighting conditions.

The new engine does not currently support *InternalResizeWidth* parameter values larger than 512 (see the [FSDK_SetFaceDetectionParameters](#) chapter). If you are detecting small faces on high-resolution images with large *InternalResizeWidth* values, consider splitting such images into parts.

When the *InternalResizeWidth* is set to 512, faces that are heavily cropped and occupy almost all of the image (such as when a face is too close to the camera) could be

detected less often than when smaller *InternalResizeWidth* values are used. However, large *InternalResizeWidth* values are not necessary when detecting such faces. In this case, consider setting the *InternalResizeWidth* to 256 or less.

As the new engine uses color information, the face detection rates on grayscale images could be lower (especially on small faces).

You may find that small faces are detected less often. In this case, try increasing the *InternalResizeWidth* value, as the internal meaning of *InternalResizeWidth* has changed in the engine.

Template format changes

Version 7.0 improves the accuracy of the [FSDK_GetFaceTemplate](#) and [FSDK_MatchFaces](#) functions. Templates are extracted in such a way that the false acceptance rates are decreased when matching blurred, low-lit, and out-of-plane faces.

The accuracy of matching templates extracted by previous FaceSDK versions is unchanged. To enjoy the increased accuracy, consider re-extracting your templates from the source images by using version 7.0.

If you are using Tracker API with the *KeepFaceImages* parameter set to true, the Tracker facial appearances will be automatically re-extracted when the Tracker memory, saved with versions 6.5 or 6.5.1, is loaded (using the [FSDK_LoadTrackerMemoryFromFile](#) or [FSDK_LoadTrackerMemoryFromBuffer](#) functions). This may take some time, depending on the size of your Tracker memory.

Removal of libstdc++ dependency on iOS

FaceSDK 7.0 for iOS has removed the dependency on the libstdc++ library to increase compatibility with Xcode 10. FaceSDK sample applications for iOS are no longer using libstdc++. To switch from using libstdc++ to using libc++ in your Xcode project, visit Build Settings – C++ Standard Library. iOS 5.x and 6.x are no longer supported.

Migration from FaceSDK 6.5 to FaceSDK 6.5.1

The 6.5.1 version improves the extraction of facial templates from images taken under very low lighting conditions. As the 6.5 version might produce higher false acceptance rates on such images, it is recommended that you re-extract your facial templates from the source pictures using the 6.5.1 version.

If you were using Tracker API with the *KeepFaceImages* parameter set to true, the Tracker facial appearances will be automatically re-extracted when the Tracker memory, saved with the 6.5 version, is loaded (using the [FSDK_LoadTrackerMemoryFromFile](#) or [FSDK_LoadTrackerMemoryFromBuffer](#) functions). This may take some time, depending on the size of your Tracker memory. If you were using the Tracker API in very low light, it is recommended that you start the Tracker memory from scratch and re-enroll your subjects.

Migration from FaceSDK 6.3, 6.3.1, 6.4 to FaceSDK 6.5

Template format changes

The 6.5 version improves face matching accuracy. It achieves a true acceptance rate of 99.83% and a false acceptance rate of 0.1% for the NIST FRGC protocol, ROC1 (compared to the 93.9% true acceptance rate achieved by the 6.4 version).

To achieve the accuracy increase, it was necessary to change the format of the face template (the `FSDK_FaceTemplate` structure). The size of the face template was decreased to 1040 bytes. To migrate to the 6.5 version and start using the new face templates, you need to update the Luxand FaceSDK interface header files in your project (see [Using FaceSDK with Programming Languages](#)) and rebuild your application. If you were relying on the size of the old face template structure (13324 bytes), for example, while saving it to a database, you need to change this value to 1040.

As the new face template is not compatible with the face template from previous versions, you need to re-extract the templates for every face in your database (with the `FSDK_GetFaceTemplate`, `FSDK_GetFaceTemplateInRegion`, or `FSDK_GetFaceTemplateUsingFeatures` functions) from the source pictures. As the template format may change again in future versions, it is recommended that you store both the original face images and their templates in the database.

The 6.5 version increased its matching accuracy, because of its more sophisticated models. However, such models require more calculations. The speed of template extraction is now lower than in the 6.4 version. This means that on slow devices, you may get lower frame rates when extracting face templates, with either the `FSDK_GetFaceTemplate`, `FSDK_GetFaceTemplateInRegion`, or `FSDK_GetFaceTemplateUsingFeatures` functions.

Template matching

The 6.5 version substantially increases the speed of face matching on most platforms. You are unlikely to require any code changes to adapt to this.

Tracker API changes

Since the template format has changed, the Tracker memory from previous Luxand FaceSDK versions is not compatible with the 6.5 version. This means you cannot load the Tracker memory saved with Luxand FaceSDK 6.4 (and earlier) using the `FSDK_LoadTrackerMemoryFromFile` or `FSDK_LoadTrackerMemoryFromBuffer` functions. To migrate to the 6.5 version, you need to start a new Tracker memory file and enroll all your subjects again.

If it is not possible for you to regenerate the templates or start a new Tracker memory, please contact our support at <https://www.luxand.com/support/>.

To make the transition to new template formats easier when using Tracker API, we added the `KeepFaceImages` parameter. When set to true (which is the default value), it will store the original facial images in the Tracker memory. If the template format changes in the new version of Tracker API, you will be able to convert your previous Tracker memory to the new template format automatically, so you won't need to reenroll your subjects. If you don't like the original facial images to be stored in the Tracker memory, you need to explicitly set this parameter to false. See the [Storing original facial images](#) section for more details.

As the speed of template extraction has decreased compared to the 6.4 version, you may get lower frame rates when the `RecognizeFaces` parameter is set to true. In older versions, it was

recommended that higher frame rates were preferable, in order that Tracker API could collect more facial appearances of a person per unit of time, which positively affected the accuracy (see the [Tuning for Optimal Performance](#) chapter). However, the Tracker API in the 6.5 version is less affected by the frame rate, because it has more robust face matching. Even if you're processing about 1 frame per second on a slow device, it is usually enough for Tracker API to efficiently recognize persons; even with that frame rate, you're likely to get much higher recognition rates than in the 6.4 version with higher frame rates.

Note that if your `RecognizeFaces` is set to false when using Tracker API (for example, when you only detect faces or facial features), your frame rate will not decrease.

The *RecognitionPrecision* parameter now has no effect and is not recommended for use.

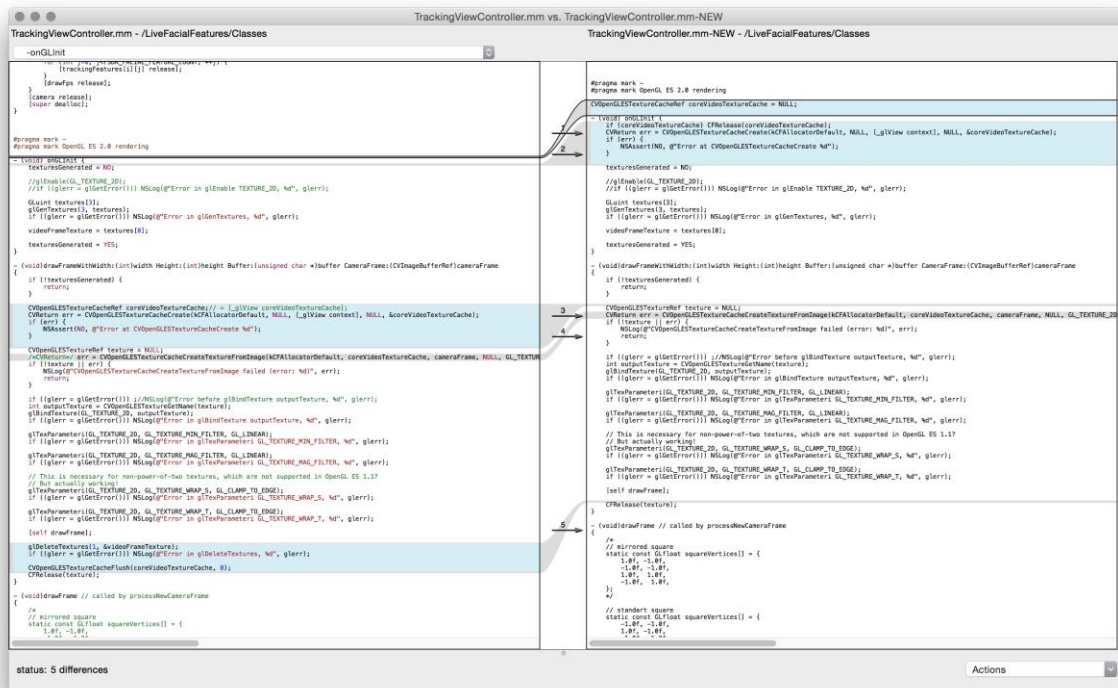
Migration from FaceSDK 6.2 to FaceSDK 6.3, 6.3.1, 6.4

The 6.3 version increases minimal OS version requirements. Applications developed with Luxand FaceSDK 6.3 will not support any Windows versions earlier than Windows XP SP3 or Windows 2003 SP2 and will not support any macOS versions earlier than 10.7.

If you are using Luxand FaceSDK with Microsoft .NET, note that FaceSDK.NET.dll now requires .NET 4.0 or higher. You need to redistribute Microsoft Visual C++ Redistributable for Visual Studio 2017 with your application.

If you are using an older version of .NET (for example, 2.0, 3.0 or 3.5), you must switch to the component available in the source code form in the `samples\advanced\.NET wrapper` directory. Note that this component is actually a wrapper for `facesdk.dll` that is linked dynamically, so `facesdk.dll` must be redistributed with the application that uses this wrapper. The LiveRecognition sample includes projects for Microsoft C# 2005/2008 and Visual Basic .NET 2005/2008 that are using this wrapper.

The LiveFacialFeatures, GenderRecognition and ExpressionRecognition samples were updated to be compatible with iOS 11. If your code was based on these samples, update your code as indicated below. You need to implement the changes as per the illustration (in short, moving the code that creates the cache from `drawFrameWithWidth` to `onGLInit`). These changes were implemented in the `TrackingViewController.mm` file of the samples.



Migration from FaceSDK 6.0, 6.0.1, 6.1 to FaceSDK 6.2

The 6.2 version adds 4 new facial feature points (numbered 66 to 69), detecting 70 facial features instead of 66 in the previous release. The following constants are added:

FSDKP_FACE_CONTOUR14,
FSDKP_FACE_CONTOUR15,
FSDKP_FACE_CONTOUR16,
FSDKP_FACE_CONTOUR17.

The numbering of the 66 facial features detected previously was not changed. It means that the new numbering is backwards compatible with the previous numbering. When migrating to the 6.2 or 6.3 version, make sure that you are using the new header files (or wrappers), since the FSDK_Features type now contains 4 more points.

The [FSDK_GetFaceTemplateUsingFeatures](#) function still employs only the first 66 facial features.

The performance of retrieving live video in the LiveFacialFeatures sample for iOS and Android is increased. If you were using this sample code in your project, consider replacing the corresponding parts of your code to increase the performance of your app.

Migration from FaceSDK 5.0, 5.0.1 to FaceSDK 6.0, 6.0.1, 6.1

Important changes for users migrating to Luxand FaceSDK 6.0, 6.01 or 6.1 from the 5.0 or 5.0.1 versions:

The [FSDK_GetFaceTemplate](#) and [FSDK_GetFaceTemplateInRegion](#) functions now detect facial features with the same accuracy as the [FSDK_DetectFacialFeatures](#) function. The accuracy of face recognition is now the same regardless of whether you call

[FSDK_GetFaceTemplate](#)/[FSDK_GetFaceTemplateInRegion](#) or first detect facial features and then pass them to the [FSDK_GetFaceTemplateUsingFeatures](#) function.

The recognition accuracy of Tracker API when RecognitionPrecision=1 is now independent of the values of the DetectFacialFeatures or DetectGender parameters.

The FacialFeatureJitterSuppression parameter of Tracker API now allows for better facial feature smoothing; however, its default setting may consume more processor resources. You may set its value to 0 if you need to consume fewer resources.

Migration from FaceSDK 4.0 to FaceSDK 5.0, 5.0.1

Important changes for users migrating to Luxand FaceSDK 5.0 or 5.0.1 from the 4.0 version:

The function [FSDK_GetFaceTemplateUsingFeatures](#) is no longer deprecated. The function expects that the coordinates of all facial features are detected. If you are passing just the coordinates of eye centers (detected with [FSDK_DetectEyes](#), [FSDK_DetectEyesInRegion](#)) to the function, call [FSDK_GetFaceTemplateUsingEyes](#) instead.

The format of the face template has changed. The size of the template is now 13324 bytes. If you have stored face templates in a database, you must recreate them from the original photos by calling the [FSDK_GetFaceTemplate](#) or [FSDK_GetFaceTemplateInRegion](#) functions.

The [FSDK_MatchFaces](#) function now returns an error when the template has an invalid format or when the formats of the templates are not supported (that is, when face templates was created with an unsupported version of Luxand FaceSDK).

If you were calling [FSDK_GetFaceTemplate](#) or [FSDK_GetFaceTemplateInRegion](#), you may find that they consume more time. This is because these functions extract face templates with higher accuracy. If you need higher performance, replace these calls with [FSDK_DetectEyes](#) or [FSDK_DetectEyesInRegion](#), and call [FSDK_GetFaceTemplateUsingEyes](#). See the [Face Matching](#) section for details.

On the other hand, if you were detecting eyes and then passing their coordinates to [FSDK_GetFaceTemplateUsingEyes](#), you need to replace this call together with the detection of eye centers to the [FSDK_GetFaceTemplate](#) or [FSDK_GetFaceTemplateInRegion](#) function to achieve the higher accuracy available in the 5.0 version.

Migration from FaceSDK 3.0 to FaceSDK 4.0

Recommendations on how to migrate from version 3.0 to version 4.0 are included.

1. The 4.0 version introduces a new high-quality algorithm for facial feature detection. The new version detects 66 facial features (see the Detected Facial Features chapter).
2. The 4.0 version enhances the accuracy of inner facial feature (nose, eyes, and mouth) detection. To make this enhancement possible, facial feature points of the upper part of the head were removed. The following facial feature points (and their corresponding constants) were removed:

FSDKP_FACE_CONTOUR3, FSDKP_FACE_CONTOUR4,
FSDKP_FACE_CONTOUR5, FSDKP_FACE_CONTOUR6,
FSDKP_FACE_CONTOUR7, FSDKP_FACE_CONTOUR8,
FSDKP_FACE_CONTOUR9, FSDKP_FACE_CONTOUR10,
FSDKP_FACE_CONTOUR11

If you were using some of these features, you may calculate their approximate positions by relying on the coordinates of other facial features.

3. You may find that FaceSDK consumes more CPU resources than the previous version. The SDK uses all available CPU cores for face detection and recognition functions, achieving higher speed. If you need the SDK to use just a single core (as in the previous version), use the [FSDK_SetNumThreads](#) function.
4. The following functions are deprecated and will not be supported in the future Luxand FaceSDK versions: `FSDK_GetFaceTemplateUsingFeatures`

Migration from FaceSDK 2.0 to FaceSDK 3.0

This section tells about changes in FaceSDK 3.0 as compared to FaceSDK 2.0. There are also recommendations on how to migrate from version 2.0 to version 3.0.

1. As version 3.0 has introduced a new enhanced face recognition algorithm, the format of a template changed as well. Now it is enough to detect only eye centers, rather than all features to build a template. If your application used to detect facial features and then created a template using detected features, now the feature detection stage can be skipped or replaced by detection of eye centers (i.e. [FSDK_DetectFacialFeatures](#) can be replaced by [FSDK_DetectEyes](#)). The size of a template was reduced to 16384 bytes. If your application used a database of saved templates, it is necessary to recreate these templates using source images.
2. New version introduces new functions for quick detection of eye centers – [FSDK_DetectEyes](#) and [FSDK_DetectEyesInRegion](#). These functions are recommended for use if it is necessary to detect eye centers in real time.
3. The meaning of similarity returned by [FSDK_MatchFaces](#) function has changed. Now similarity is approximately equal to the probability that templates belong to one and the same person. More information on this topic can be found in [Face Matching](#) chapter.
4. The new [FSDK_SetCameraNaming](#) function is added. It determines what the [FSDK_GetCameraList](#) function returns – either the list of camera names available in the system, or the list of unique device paths of these cameras. (It may be required if two similar webcams of the same manufacturer are plugged in to the computer.)
5. Camera management functions are included into `facesdk.dll`. If you used camera management in your applications, you may remove `facesdkcam.dll` and header files related to `facesdkcam`.
6. .NET wrapper does not require `facesdk.dll`. The camera management functions are also included into `FaceSDK.NET.dll` (but they are still located in class `FSDKcam`). Now the wrapper is located in the directories `\bin\win32\` for 32-bit applications and `\bin\win64\` for 64-bit applications.
7. The following functions have been removed from the library and will not be supported: `FSDK_LocateFace`, `FSDK_LocateFacialFeatures`, `FSDK_ExtractFaceImage`.
8. The following functions are deprecated and will not be supported in the future Luxand FaceSDK versions: `FSDK_GetFaceTemplateUsingFeatures`

Error Codes

The FaceSDK library defines the following error codes:

Error Name	Value
FSDKE_OK	0
FSDKE_FAILED	-1
FSDKE_NOT_ACTIVATED	-2
FSDKE_OUT_OF_MEMORY	-3
FSDKE_INVALID_ARGUMENT	-4
FSDKE_IO_ERROR	-5
FSDKE_IMAGE_TOO_SMALL	-6
FSDKE_FACE_NOT_FOUND	-7
FSDKE_INSUFFICIENT_BUFFER_SIZE	-8
FSDKE_UNSUPPORTED_IMAGE_EXTENSION	-9
FSDKE_CANNOT_OPEN_FILE	-10
FSDKE_CANNOT_CREATE_FILE	-11
FSDKE_BAD_FILE_FORMAT	-12
FSDKE_FILE_NOT_FOUND	-13
FSDKE_CONNECTION_CLOSED	-14
FSDKE_CONNECTION_FAILED	-15
FSDKE_IP_INIT_FAILED	-16
FSDKE_NEED_SERVER_ACTIVATION	-17
FSDKE_ID_NOT_FOUND	-18
FSDKE_ATTRIBUTE_NOT_DETECTED	-19
FSDKE_INSUFFICIENT_TRACKER_MEMORY_LIMIT	-20
FSDKE_UNKNOWN_ATTRIBUTE	-21
FSDKE_UNSUPPORTED_FILE_VERSION	-22
FSDKE_SYNTAX_ERROR	-23
FSDKE_PARAMETER_NOT_FOUND	-24
FSDKE_INVALID_TEMPLATE	-25
FSDKE_UNSUPPORTED_TEMPLATE_VERSION	-26
FSDKE_CAMERA_INDEX_DOES_NOT_EXIST	-27
FSDKE_PLATFORM_NOT_LICENSED	-28

Library Information

The FaceSDK library uses:

- TensorFlow-Lite © Google, LLC;
- libjpeg-turbo (copyright by Miyasaka Masaru, TigerVNC and VirtualGL projects);
- libcurl © Daniel Stenberg;
- libpng © Glenn Randers-Pehrson;
- easybmp © The EasyBMP Project (<http://easybmp.sourceforge.net>);
- RSA Data Security, Inc. MD5 Message-Digest Algorithm.