EyeFace SDK

Developer's Guide

Version 4.4.0

Contact:

Address:
Eyedea Recognition, s.r.o.
Vyšehradská 320/49
128 00, Prague 2
Czech Republic

web: http://www.eyedea.cz

email: info@eyedea.cz

Table of Contents

1 Product Description 4
1.1 Technical Details 4
2 Distribution Contents 5
3 Installation Guide 6
3.1 Trial Version 6
3.2 Before Installation 6
3.3 Installation on Windows 6
3.4 Installation on Linux 6
3.5 Verification of Installation 7
3.6 Installation Failures 7
3.7 Managing licenses 8
3.8 Error codes on licensing failures 8
4 EyeFace SDK Standard API 20
4.1 Constants 20
4.2 Enumerators 20
4.3 Structures 21
4.4 Functions 27
5 EyeFace SDK Expert API 34
5.1 Constants 34
5.2 Enumerators 34
5.3 Structures 34
5.4 Functions 36
6 EyeFace SDK Examples 44
6.1 Overview 44
6.2 Configuring OpenCV 45
6.3 Building Examples 45
6.4 Example-OpenCV Window 46
7 Hello EyeFace! 47
7.1 Explicit Linking Explained 47
7.2 Global Initialization 48
7.3 Loading EyeFace SDK Shared Library 49
7.4 Initializing EyeFace SDK Engine 49
7.5 Processing Loop 49
7.6 Getting the Results 50
7.7 Freeing EyeFace SDK Resources 50

7.8 Compiling and Linking 50
7.9 Example Remarks 50
8 EyeFace SDK Wrappers 51
8.1 C# Wrapper 51
8.2 Java Wrapper 51
8.3 MEX Wrapper 51
9 EyeFace SDK Licensing 52
9.1 EyeFace SDK API's 52
9.2 License Management 52
9.3 Generating a Licensing Request 53
9.4 Attaching a New License 53
9.5 Transferring License to Another Machine 54
9.6 License Logout Failures 55
10 Configuration Parameters 56
10.1 Configuration File Syntax 56
10.2 EyeFace Parameters 56
10.3 Face Detector Parameters 57
10.4 Camera Parameters 58
10.5 Face Attributes Recognition Parameters 58
10.6 Tracking Parameters 59
10.7 Log to File Parameters 60
10.8 Log to Server Parameters 60
10.9 Initializing EyeFace SDK with Custom Configuration 61
10.10 Example OpenCV with Rotations Enabled 61
11 Logging Modules 62
11.1 Setting up the File Logging 62
11.2 Setting up the Server Logging 62
11.3 Syntax of Log Messages 62
12 Known Issues 65
12.1 Maintaining Relative Paths to the EyeFace SDK 65
12.2 Microsoft Windows - Windows Live 65
12.3 Distributing EyeFace SDK to Customers 65
13 Problem Solving 66
14 Third Party Software 67

1 Product Description

EyeFace SDK is a cross-platform software library designed to provide a convenient way for face detection and face attribute recognition in the input images and videos. It defines an interface between the client's software and our state-of-the-art recognition modules. EyeFace SDK allows the client to detect faces, recognize gender, age, emotion and ancestry, estimate position of facial landmarks such as eyes, nose and mouth. EyeFace SDK also allows the client to track faces in videos, collect statistics over facial tracks and log results into a file or to a remote server. A decade of research and engineering has enabled Eyedea Recognition to bundle all the functionality into a single easy-to-use software library, EyeFace SDK.

1.1 Technical Details

EyeFace SDK is a standalone cross-platform x86/x64 C++ library with internal multi-threading. It currently provides the following to be integrated into the client's software solutions (the non-native APIs are bundled in a form of wrappers with source code included):

- C native API
- C# API
- Java JNI API
- Matlab/Octave MEX API

Eyedea Recognition, Ltd. focuses on cross-platform software development. The following operating systems and platforms are officially supported:

- Windows 7 & Windows 10 32b/64b (Microsoft Visual Studio 2015)
- Ubuntu 16.04 64b
- CentOS 7 (1611) 64b

Newer versions of the above systems and platforms should be compatible with EyeFace SDK. Other platforms like Debian 8 or Red Hat Linux 8 should be compatible, though are not tested. For information regarding additional interfaces, operating systems and/or platforms (e.g. ARM), please contact Eyedea Recognition.

2 Distribution Contents

The following list is an excerpt from the EyeFace SDK directory structure, highlighting the most important directories and files contained in the distribution. A brief description of the items is given. More information about each of the items will be provided in subsequent chapters.

- [EyeFaceSDK]/ ..... *distribution main folder*
  - data/ ..... *support data for examples*
    - test-images-id/ ..... *test images*
  - documentation/ ..... *documentation folder*
    - doxygen/ ..... *doxygen documentation*
    - developers-guide.pdf ..... *this document*
    - release-notes.txt ..... *new version info*
  - examples/ ..... *examples source files*
    - example-API/ ..... *basic face recognition API examples*
    - example-getinfo/ ..... *software protection API example*
    - example-hello-eyeface ..... *hello world example in EyeFace SDK*
    - example-logserver/ ..... *remote server logging example*
    - example-opencv/ ..... *OpenCV webcam example*
  - eyefacesdk/ ..... *EyeFace SDK apps mandatory data*
    - include/ ..... *header files*
    - lib/ ..... *shared libraries*
    - models/ ..... *recognition models*
    - config.ini ..... *main configuration file*
  - hasp/ ..... *software protection binaries*
  - wrappers/ ..... *API wrappers (C#, Java, MEX)*

3 Installation Guide

Installation of software licensing daemon is the first step to start using EyeFace SDK. The library comes equipped with a standard third party software licensing solution, Sentinel LDK by *gemalto*. This chapter will guide the client through installation on Windows and Linux. In the process, the client will install a daemon service, Sentinel License Manager, that will automatically start upon system startup. The application enables encrypted binaries of EyeFace SDK to run, and to manage licenses using a web browser.

3.1 Trial Version

EyeFace SDK is bundled with a 10-day trial license with an unlimited number of concurrently running instances. The trial license comes with the Expert interface enabled by default. The library can be also configured to use the Standard interface only (please see Chapter 8 for details). The trial license does not support virtual machines.

3.2 Before Installation

Prior to the installation of the licensing software, all Sentinel Hardware Keys should be removed from the target computer based on the recommendation by *gemalto*. Leaving it connected during the installation process might cause the Sentinel Hardware Key to not be properly recognized by the new installation of Sentinel License Manager.

The Sentinel License Manager does not support read-only filesystems (on Windows, the functionality is called Enhanced Write Filter).

3.3 Installation on Windows

Follow these steps to install Sentinel License Manager on a Windows machine:

- start the command line cmd with **Administrator** privileges
- navigate to **[EyeFaceSDK]/hasp/** directory
- execute **"dunst.bat"** to uninstall any previous versions of the Sentinel License Manager
- execute **"dinst.bat"** to install the Sentinel License Manager and the trial license keys

3.4 Installation on Linux

Follow these steps to install Sentinel License Manager on a Linux machine:

- start the command line and navigate to **[EyeFaceSDK]/hasp/** directory
- on 64-bit Linux distributions, install the **32-bit** compatibility binaries
- on Ubuntu 16.04:
  - execute **"sudo apt-get install libc6:i386"**
- on CentOS 7:
  - execute **"sudo yum install glibc.i686"**
- execute **"sudo ./dunst"** to uninstall any previous versions of the Sentinel License Manager
- execute **"sudo ./dinst"** to install the Sentinel License Manager and the trial license keys

3.5 Verification of Installation

The software licensing daemon contains a web based interface, which also allows the client to check the available licenses. To verify that the installation of Sentinel License Manager was successfully completed, the client should open a web browser at **http://localhost:1947/_int_/devices.html**. The web page will be displayed, as seen in Illustration 1. The client must check that the trial licenses were installed properly and EyeFace SDK works on the machine before ordering a full license. If not, a problem may arise in the future when connecting the full license, resulting in licensing failure and additional costs to re-license the software to another machine.

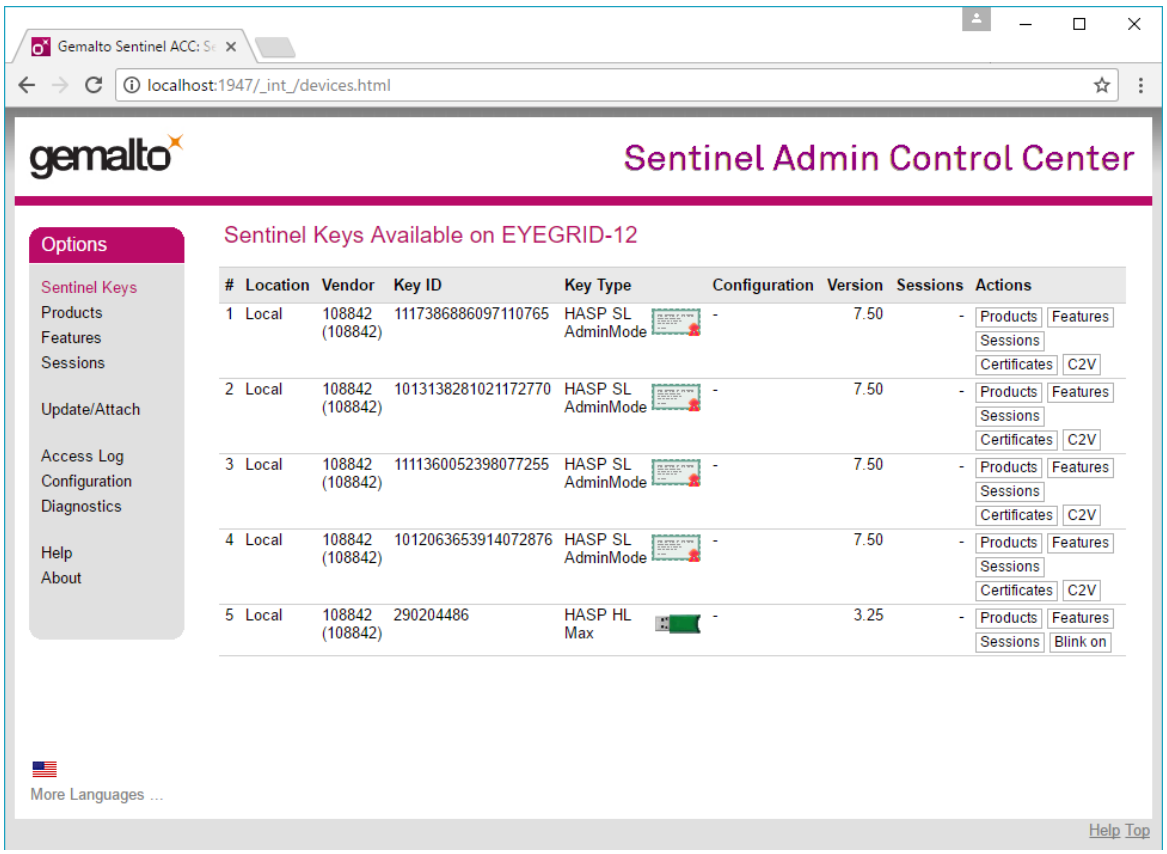


Illustration 1: Sentinel License Manager screenshot.

The web page lists all the available license keys. Under the "Products" link in the left pane all available products are listed. After the installation, the "Products" page should contain Product 2 (EyeFace-Standard-SDK) and Product 12 (EyeFace-Expert-SDK) licenses with Vendor ID 108842.

3.6 Installation Failures

On Windows, the installation of Sentinel License Manager might be broken by Anti-Virus application. If the installation failed, the client should disable the Anti-Virus application and rerun the installation of Sentinel License Manager. Even after successful installation, the Sentinel License Manager might fail to show up in the web browser. This can be solved by adding **C:\Windows\system32\hasplms.exe** to the exception list of the Anti-Virus. Port number 1947 must be also added to the exception list in the Windows firewall and also in the Anti-Virus in case it uses its own firewall.

3.7 Managing licenses

It is of the most importance that the client understands the licensing schemes used in *gemalto*'s Sentinel LDK software protection framework. Otherwise, unrepairable damage might be caused leading to additional costs to recover the already purchased licensing keys. The topic of license management is fully covered in Chapter 7.2.

3.8 Error codes on licensing failures

The error codes are outputted to error stream of the application (typically stderr) using EyeFace SDK. The user needs to check the error stream for the error codes and fix the issues before deployment. The following error codes and messages are the most common ones

- H0007 – Sentinel HASP key not found. (There is no license of the EyeFace SDK on the PC.)
- H0033 – Unable to access Sentinel HASP Runtime Environment. (No License Manager found.)
- H0041 – Feature has expired. (The license on the PC has expired, consider renewal.)

The shared library of EyeFace SDK is encrypted for enhanced software protection. On the other hand, in case of failure the application does not terminate but crashes after a few calls to the library, which is a security measure against reverse engineering but also causes confusion of the users. The client need to make sure he monitors the error codes described in the standard error to distinguish between programming errors and licensing problems.

4 ERImage Application Interface

This part contains the information about the ways of digital image data storage and processing. It describes the image data storing in the memory from the theoretical point of view in the document part *Image format*, remaining parts cover the application interface used for image manipulation with the data structure *ERImage*. Description of all available *Enumerators* and *Functions* is included.

4.1 Image format

Digital image data can be persisted in many different forms. Since it is the main input of the processing, it is very important to understand the form used for image storage and manipulation. Currently three color models are supported in the *ERImage* image structure. First is *BGR* color model, second is *Gray* color model and the third is *YCbCr 4:2:0* color model.

Each color model is described for the cases where the image data are stored in the 1D data array, same as the *ERImage* structure supports. The data are stored in the 1D array per image rows.

4.1.1 BGR

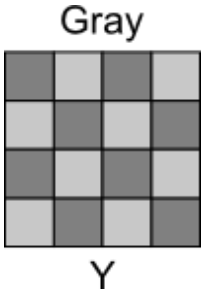
Three channel model, which is derived from *RGB*, and is supported by the *ERImage* is *BGR* (B – blue, G – green, R – red). The model stores image using three values per pixel, where the first value is blue component, second value is green component and the third is red component. Image is saved row by row in the 1D array. Following formulas show how to access the pixel color components B, G and R in the 1D array *data* of the image with resolution *width* × *height* on coordinates *(x, y)*. Coordinates *x*, *y* and *data* array indices are 0-based.

$$B(x,y) = data(3 * (width * y + x) + 0)$$ *B component on (x, y) coordinates*

$$G(x,y) = data(3 * (width * y + x) + 1)$$ *G component on (x, y) coordinates*

$$R(x,y) = data(3 * (width * y + x) + 2)$$ *R component on (x, y) coordinates*

4.1.2 Gray



One channel model *Gray* is used for storing grayscale image, which is composed from luminance values (Y - luminance). The model stores images using one value per pixel, where the value is the luminance component. Image is saved row by row in the 1D array. Following formula shows how to access the pixel luminance component Y in the 1D array *data* of the image with resolution *width* × *height on coordinates (x, y)*. Coordinates *x*, *y* and *data* array indices are 0-based.

$$Y(x,y) = data(width * y + x)$$ *Y component on (x, y) coordinates*

4.1.3 YCbCr 4:2:0

Three plane model *YCbCr 4:2:0* is used for storing color image, where the first plane contains luminance (Y component, image brightness), the second plane contains blue-difference chroma component (Cb) and the third plane contains red-difference chroma component (Cr). Cb and Cr planes have half resolution than Y image plane. Four neighboring Y values belongs to one Cb and one Cr value.

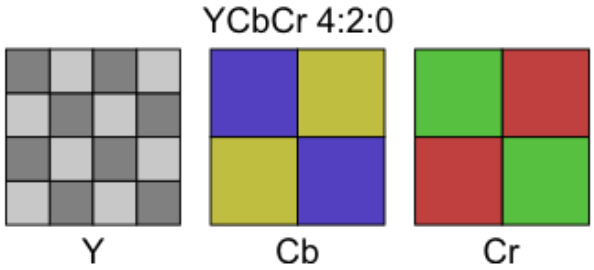


Image is saved per planes in the 1D array, where each plane is saved row by row. Following formulas show how to access the pixel color components Y, Cb and Cr in the 1D array *data* of the image with resolution *width × height* on coordinates *(x, y)*. Coordinates *x*, *y* and *data* array indices are 0-based. **All divisions in the formulas are integer divisions.**

$$Y(x,y) = data(width * y + x)$$ *Y component on (x, y) coordinates*

$$|Y| = width * height$$ *Size of the Y image plane*

$$Cb(x,y) = data\left(|Y| + \frac{y}{2} * \frac{width}{2} + \frac{x}{2}\right)$$ *Cb component on (x, y) coordinates*

$$|Cb| = |Cr| = \frac{width * height}{4}$$ *Size of the Cb and Cr image plane*

$$Cr(x,y) = data\left(|Y| + |Cb| + \frac{y}{2} * \frac{width}{2} + \frac{x}{2}\right)$$ *Cr component on (x, y) coordinate*

4.2 Application Interface

4.2.1 Enumerators

This part defines the API enumerators which are related to the *ERImage* structure:

ERImageColorModel

ERImageColorModel is used to specify the way how color channel values are saved in the image. More information about the supported color models is in the section *Image format*.

- **ER_IMAGE_COLORMODEL_UNK = 0**
  - Default value - Unknown color model.
- **ER_IMAGE_COLORMODEL_GRAY = 1**
  - One channel grayscale color model. Image luminance values are saved row by row.
- **ER_IMAGE_COLORMODEL_BGR = 2**
  - Three channel BGR color model. Three values per pixel stored row by row.
- **ER_IMAGE_COLORMODEL_YCBCR420 = 3**
  - Three plane YCbCr 4:2:0 color model. Luminance plane and two chroma planes are stored separately each row by row.

ERImageDataType

ERImageDataType specifies the data type used for storing values of the image.

- **ER_IMAGE_DATATYPE_UNK = 0**
  - Default value – Unknown data type.
- **ER_IMAGE_DATATYPE_UCHAR = 1**
  - All image values are saved as *unsigned char*.
- **ER_IMAGE_DATATYPE_FLOAT = 2**
  - All image values are saved as *float*.

4.2.2 Structures

This part defines the API structure *ERImage* used for digital image data manipulation:

ERImage

```
typedef struct {
    ERImageColorModel  color_model;
    ERImageDataType    data_type;
    unsigned int       width;
    unsigned int       height;
    unsigned int       num_channels;
    unsigned int       depth;
    unsigned int       step;
    unsigned int       size;
    unsigned int       data_size;
    unsigned char*     data;
    unsigned char**    row_data;
    unsigned char      data_allocated;
} ERImage;
```

ERImage represents digital image data in the special structure designed to work with the SDK. The structure contains the color model and the data type in the *ERImageColorModel* and the *ERImageDataType* enumerators together with the parameters defining the size of the image and the underlying data. Image data is saved in the *data* field row by row. For more information see the part *Image format*.

The structure contains following fields:

- **color_model**
  Image data color model represented by the enumerator *ERImageColorModel*.
- **data_type**
  Image date type represented by the enumerator *ERImageDataType*.
- **width**
  Width of the image in pixels.
- **height**
  Height of the image in pixels.
- **num_channels**
  Number of image channels.
- **depth**
  Size of one image pixel in bytes.
- **step**
  Number of bytes between each two beginnings of the row in the *data* array.
- **size**
  Size of the image in bytes.
- **data_size**
  Size of the allocated data in the structure.
- **data**
  Array containing the image data.
- **row_data**
  Array containing pointers to the *data* array. Each points to the beginning of the specific image row in the *data* array.
- **data_allocated**
  Value containing the flag whether the *data* field was allocated within the structure allocation or on the user side. (0 – allocated by user, 1 – allocated with the structure)

4.2.3 Functions

This part defines the API functions which are designed to work with the *ERImage* structure:

- **Allocation**
  erImageAllocate, *erImageAllocateBlank,*
  erImageAllocateAndWrap and *erImageCopy*
- **Properties**
  erImageGetDataTypeSize, *erImageGetColorModelNumChannels*,
  erImageGetPixelDepth and *erVersion*
- **IO Operations**
  erImageRead and *erImageWrite*
- **Freeing**
  erImageFree

These functions are defined in the *er_image.h* file.

erImageAllocate

Allocates the *ERImage* structure.

Specification:

```
int erImageAllocate(ERImage* image, unsigned int width, unsigned int height,
                    ERImageColorModel color_model, ERImageDataType data_type);
```

Inputs:

- **image**
  Pointer to the *ERImage* structure instance to allocate.
- **width**
  Width of the image to allocate.
- **height**
  Height of the image to allocate.
- **color_model**
  Color model of the image to allocate (see *ERImageColorModel*).
- **data_type**
  Data type of the image to allocate (see *ERImageDataType*).

Returns:

- 0 – Image successfully allocated.
- *other* – Error during image allocation.

Description:
The function *erImageAllocate()* is used for *ERImage* structure data allocation. The input of the function is the pointer to the *ERImage* structure instance, width and height of the image to allocate and the color model and the data type specification.

Example:

```
ERImage* image = new ERImage();
// Allocate grayscale (1 channel) image with resolution 800x600 and 1 byte per channel
int res = erImageAllocate(image, 800, 600, ER_IMAGE_COLORMODEL_GRAY, ER_IMAGE_DATATYPE_UCHAR);
```

erImageAllocateBlank

Allocates the *ERImage* structure without the internal data arrays.

Specification:

```
int erImageAllocateBlank(ERImage* image, unsigned int width, unsigned int height,
                         ERImageColorModel color_model, ERImageDataType data_type);
```

Inputs:

- **image**
  Pointer to the *ERImage* structure instance to allocate.
- **width**
  Width of the image to allocate.
- **height**
  Height of the image to allocate.
- **color_model**
  Color model of the image to allocate (see *ERImageColorModel*).
- **data_type**
  Data type of the image to allocate (see *ERImageDataType*).

Returns:

- 0 – Image successfully allocated.
- *other* – Error during image allocation.

Description:
The function er*ImageAllocateBlank()* is used for *ERImage* structure properties allocation, but without the internal data array allocation. The input of the function is the pointer to the *ERImage* structure instance, width and height of the image to allocate and the color model and the data type specification.

IMPORTANT: Only the fields with image properties are allocated. Image *data* field is NULL, *row_data* is NULL and field *data_size* is 0 after the successful function call.

Example:

```
ERImage* image = new ERImage();
// Allocate blank BGR (3 channel) image with resolution 640x480 and 1 float per channel
int res = erImageAllocateBlank(image, 640, 480, ER_IMAGE_COLORMODEL_BGR, ER_IMAGE_DATATYPE_FLOAT);
// image->data == NULL, image->row_data == NULL and image->data_size == 0
```

erImageAllocateAndWrap

Allocates the *ERImage* structure and wrap it over the supplied image data.

Specification:

```
int erImageAllocateAndWrap(ERImage* image, unsigned int width, unsigned int height,
                           ERImageColorModel color_model, ERImageDataType data_type,
                           unsigned char* data, unsigned int step);
```

Inputs:

- **image**
  Pointer to the *ERImage* structure instance to allocate.
- **width**
  Width of the image to allocate.
- **height**
  Height of the image to allocate.
- **color_model**
  Color model of the image to allocate (see *ERImageColorModel*).
- **data_type**
  Data type of the image to allocate (see *ERImageDataType*).
- **data**
  Image data to wrap.
- **step**
  Definition of the input data image row step.
  (length of one image row in bytes in the input data)

Returns:

- 0 – Image successfully allocated.
- *other* – Error during image allocation.

Description:

The function *erImageAllocateAndWrap()* is used for *ERImage* structure data allocation and supplied image data wrapping. The input of the function is the pointer to the *ERImage* structure instance, width and height of the image to allocate, the color model and the data type specification, the pointer to the image data to wrap and step value which defines the size of the row in bytes.

Example:

```
unsigned char* data; // Image data to wrap
ERImage* image = new ERImage();
// Allocate grayscale (1 channel) image with resolution 800x600 and 1 byte per channel
// and wrap it over the image data supplied in the unsigned char* data array.
int res = erImageAllocateAndWrap(image, 800, 600, ER_IMAGE_COLORMODEL_GRAY,
                                 ER_IMAGE_DATATYPE_UCHAR, data, 800);
```

erImageCopy

Performs deep copy of the *ERImage* structure instance.

Specification:

```
int erImageCopy(const ERImage* image, ERImage* image_copy);
```

Inputs:

- **image**
  Pointer to the *ERImage* structure instance to copy.
- **image_copy**
  Pointer to the *ERImage* structure to copy the data into.

Returns:

- 0 – Image successfully copied.
- *other* – Error during image copying.

Description:
The function *erImageCopy()* is used for *ERImage* data copying to another instance of *ERImage* structure. The input is the pointer to the *ERImage* structure instance to copy and the output is the pointer to the *ERImage* structure instance to copy the data into.

IMPORTANT:
The allocation of the *image_copy* is done within the function before the data copying.

Example:

```
ERImage* image;                      // Image with source data
ERImage* image_copy = new ERImage(); // Destination image to copy the data into
// Deep copy of the image
int res = erImageCopy(image, image_copy);
```

erImageGetDataTypeSize

Returns size in bytes of the specific *ERImageDataType*.

Specification:

```
unsigned int erImageGetDataTypeSize(ERImageDataType data_type);
```

Inputs:

- **data_type**
  ERImageDataType to get the size of.

Returns:

- *data type size* – Size of the one channel image element value in bytes.
- 0 – Unknown *ERImageDataType* used.

Description:
The function er*ImageGetDataTypeSize()* is used to get the size in bytes of the specific *ERImageDataType* when used for image allocation. The input is the *ERImageDataType* value. The output is the value, which represents the number of bytes needed for storing one channel value of one pixel when specific *ERImageDataType* is used.

Example:

```
unsigned int sizeUC = erImageGetDataTypeSize(ER_IMAGE_DATATYPE_UCHAR);
// sizeUC == sizeof(unsigned char)

unsigned int sizeF  = erImageGetDataTypeSize(ER_IMAGE_DATATYPE_FLOAT);
// sizeF  == sizeof(float)
```

erImageGetColorModelNumChannels

Returns number of channels of the *ERImageColorModel*.

Specification:

```
unsigned int erImageGetColorModelNumChannels(ERImageColorModel color_model);
```

Inputs:

- **color_model**
  ERImageColorModel to get the number of channels.

Returns:

- *number of channels* – Number of channels of the supplied color model.
- 0 – Unknown *ERImageColorModel* used.

Description:

The function *erImageGetColorModelNumChannels()* is used to get the number of channels of the specific *ERImageColorModel*. The input is the *ERImageColorModel* value. The output is the value, which represents the number color model channels used when storing the image with specific *ERImageColorModel*.

IMPORTANT: In case of the *ER_IMAGE_COLORMODEL_YCBCR420* color model the number of image planes is returned instead of number of channels.

Example:

```
unsigned int numChannelsGRAY   = erImageGetColorModelNumChannels(ER_IMAGE_COLORMODEL_GRAY);
// numChannelsGRAY    == 1

unsigned int numChannelsBGR    = erImageGetColorModelNumChannels(ER_IMAGE_COLORMODEL_BGR);
// numChannelsBGR     == 3

unsigned int numPlanesYCBCR420 = erImageGetColorModelNumChannels(ER_IMAGE_COLORMODEL_YCBCR420);
// numPlanesYCBCR420  == 3
```

erImageGetPixelDepth

Returns size of the pixel in bytes for supplied *ERImageColorModel* and *ERImageDataType*.

Specification:

```
unsigned int erImageGetPixelDepth(ERImageColorModel color_model,
                                  ERImageDataType data_type);
```

Inputs:

- **color_model**
  Input *ERImageColorModel* for pixel depth computation.
- **data_type**
  Input *ERImageDataType* for pixel depth computation.

Returns:

- *depth of the pixel* – Number of bytes needed to store one pixel using the specified color model and data type.
- 0 – Unknown *ERImageColorModel* and/or *ERImageDataType* used.

Description:

The function *erImageGetPixelDepth()* is used to get the size of one pixel in bytes of the combination of *ERImageColorModel* and *ERImageDataType*. The input is the *ERImageColorModel and ERImageDataType* values. The output is the value, which represents the size of one pixel in bytes used when storing the image with specific *ERImageColorModel* and *ERImageDataType*.

IMPORTANT: In case of the *ER_IMAGE_COLORMODEL_YCBCR420* color model the pixel depth value is not valid due to different way of image storing when image plane color model used.

Example:

```
unsigned int dUCGray = erImageGetPixelDepth(ER_IMAGE_COLORMODEL_GRAY, ER_IMAGE_DATATYPE_UCHAR);
// dUCGray == 1

unsigned int dFBGR   = erImageGetPixelDepth(ER_IMAGE_COLORMODEL_BGR, ER_IMAGE_DATATYPE_FLOAT);
// dFBGR   == 3*sizeof(float)
```

erVersion

Returns the version of the *ERImage* structure and all related image utilities.

Specification:

```
const char* erVersion(void);
```

Returns:

- *version of the ERImage* – String containing the version of the *ERImage*.

Description:

The function *erVersion()* is used to get the version of the *ERImage* structure and all related image utilities. The function returns the string which contains the version number.

Example:

```
const char* version = erVersion();
std::cout << "ERImage version: " << version << std::endl;
```

erImageRead

Reads the image from file, decodes it and loads it into the *ERImage* structure instance.

Specification:

```
int erImageRead(ERImage* image, const char* filename);
```

Inputs:

- **image**
  Pointer to the *ERImage* structure instance to load the image into.
- **filename**
  String containing the path to the image file to read.

Returns:

- 0 – Image successfully read.
- *other* – Error during image reading.

Description:

The function *erImageRead()* is used to read and decode the image from given file and load it into the *ERImage* structure instance. The input is the pointer to the *ERImage* instance and the string containing the path to the image file to open.

Supported image formats:

- JPEG files - *.jpeg, *.jpg, *.jpe
- JPEG 2000 files - *.jp2
- Portable Network Graphics - *.png
- Windows bitmaps - *.bmp, *.dib
- TIFF files - *.tiff, *.tif
- Portable image format - *.pbm, *.pgm, *.ppm *.pxm, *.pnm

Example:

```
char* filename = "./image.jpg";          // Image file path to read
ERImage* image = new ERImage();          // Initialize the ERImage
int res = erImageRead(image, filename); // Read the image
```

erImageWrite

Encodes and writes the image from the *ERImage* structure to file.

Specification:

```
int erImageWrite(const ERImage* image, const char* filename);
```

Inputs:

- **image**
  Pointer to the *ERImage* structure instance containing the image to write.
- **filename**
  String containing the path to the image file to write.

Returns:

- 0 – Image successfully written.
- *other* – Error during image writing.

Description:
The function *erImageWrite()* is used to encode and write the image to given file from the *ERImage* structure instance. The input is the pointer to the *ERImage* instance and the string containing the path to the image file to write. Output image format is automatically selected from the filename extension with respect to the table of supported formats in the *erImageRead* chapter.

Example:

```
char* filename = "./image.jpg";          // Image file path to write
ERImage* image;                          // ERImage containing the image to write
int res = erImageWrite(image, filename); // Write the image
```

erImageFree

Frees the whole structure instance *ERImage*.

Specification:

```
void erImageFree(ERImage* image);
```

Inputs:

- **image**
  Pointer to the *ERImage* structure instance to delete.

Description:
The function *erImageFree()* is used to free the image data arrays contained in the *ERImage* structure instance and also all the property fields are set to 0. The input is the pointer to the *ERImage* instance.

IMPORTANT: The function DOES NOT delete the *ERImage* instance pointer because the user creates the pointer.

Example:

```
erImageAllocate(image, 800, 600, ER_IMAGE_COLORMODEL_GRAY, ER_IMAGE_DATATYPE_UCHAR);
// ...
erImageFree(image); // every field in the image structure is freed and set to NULL or 0
```

5 EyeFace SDK Standard API

The Standard API can be used for video processing. It is possible to detect and track faces in video, recognize facial attributes like age, gender, emotions and ancestry and recover the results either programmatically, to a file or even to a remote server.

The API is described in its C interface. The other interfaces are very similar, sometimes using advanced features not available in C. None of the functions can be considered thread safe.

5.1 Constants

Constants are used to define version and static sizes of input arguments.

5.1.1 EYEFACE_VERSION_NUMBER

Version of EyeFace SDK header files. Compare with the return value of *efGetLibraryVersion()* to verify you use the same version of EyeFace SDK both in build time and runtime.

5.1.2 EYEDEA_EYEFACE_EF2DPOINTS_MAX_SIZE

Maximum number of points in Ef2dPoints structure. Required to make the whole C API independent of runtime allocation.

5.2 Enumerators

This part contains all the information about enumerators used in the EyeFace SDK's Standard API.

5.2.1 EfBool

EfBool is a parallel to C++ bool type. It is used to store logical values true and false.

- **EF_FALSE = 0**
  - Logical "false" value.
- **EF_TRUE = 1**
  - Logical "true" value.

5.2.2 EfGenderClass

Gender classification return value, can be unknown, male or female.

- **EF_GENDER_MALE = -1**
  - Gender "male" value.
- **EF_GENDER_FEMALE = 0**
  - Gender "unknown" value.
- **EF_GENDER_FEMALE = 1**
  - Gender "female" value.

5.2.3 EfEmotionClass

Emotion classification return value, can be unknown, not smiling or smiling.

- **EF_EMOTION_NOTSMILING = -1**
  - Emotion “not smiling” value.
- **EF_EMOTION_UNKNOWN = 0**
  - Emotion “unknown” value.
- **EF_EMOTION_SMILING = 1**
  - Emotion “smiling” value.

5.2.4 EfAncestryClass

Ancestry classification return value, can be unknown, Caucasian, Asian or African.

- **EF_ANCESTRY_UNKNOWN = 0**
  - Ancestry “unknown” value.
- **EF_ANCESTRY_CAUCASIAN = 1**
  - Ancestry “Caucasian” value.
- **EF_ANCESTRY_ASIAN = 1**
  - Ancestry “Asian” value.
- **EF_ANCESTRY_AFRICAN = 3**
  - Ancestry “African” value.

5.2.5 EfTrackStatus

Track status value, can be live or finished. This enumerator identifies whether the track is active or has finished in the current frame.

- **EF_TRACKSTATUS_LIVE = 0**
  - Track status “live” value. Track is alive.
- **EF_TRACKSTATUS_FINISHED = 0**
  - Track status “finished” value. Track has finished in the current frame.

5.3 Structures

This part contains all information about structures used in EyeFace SDK’s Standard API.

5.3.1 Ef2dPoints

```
typedef struct
{
    unsigned int        length;
    double              rows[EYEDEA_EYEFACE_EF2DPOINTS_MAX_SIZE];
    double              cols[EYEDEA_EYEFACE_EF2DPOINTS_MAX_SIZE];
}Ef2dPoints;
```

Ef2dPoints represents an array of two dimensional points, defined by row and column coordinates. The coordinates are 0-based in C, C# and Java and 1-based in Matlab as is the convenience.

- **length**
  Number of points used in the Ef2dPoints structure.

- **rows**
  Row coordinates of points. Can be referred to as y-axis coordinates.
- **cols**
  Column coordinates of points. Can be referred to as x-axis coordinates.

5.3.2 EfBoundingBox

```
typedef struct
{
    int       top_left_col;
    int       top_left_row;
    int       top_right_col;
    int       top_right_row;
    int       bot_left_col;
    int       bot_left_row;
    int       bot_right_col;
    int       bot_right_row;
}EfBoundingBox;
```

EfBoundingBox represents a quadrilateral (typically a rectangle or square) by its four corners. It is used to represent position of faces in the image or to select active area for processing. The coordinates are 0-based in C, C# and Java and 1-based in Matlab as is the convenience.

The coordinates are relative to the detected object in case of detection. That means that the *top_left* corner is the corner most close to the face's right eye (physiognomic directions). In case of rotated faces, the *top_left* corner might not be the uppermost nor the leftmost corner.

- **top_left_col**
  Column index of the top left corner of the bounding box.
- **top_left_row**
  Row index of the top left corner of the bounding box.
- **top_right_col**
  Column index of the top right corner of the bounding box.
- **top_right_row**
  Row index of the top right corner of the bounding box.
- **bot_left_col**
  Column index of the bottom left corner of the bounding box.
- **bot_left_row**
  Row index of the bottom left corner of the bounding box.
- **bot_right_col**
  Column index of the bottom right corner of the bounding box.
- **bot_right_row**
  Row index of the bottom right corner of the bounding box.

5.3.3 EfLandmarks

```
typedef struct
{
    EfBool        recognized;
    Ef2dPoints    points;
    double        angles[3];
}EfLandmarks;
```

Structure representing the facial landmarks of a single face. The landmarks are points on the face like eyes, nose, etc.

The order of storage of landmarks is as follows (C-like zero-based indexing):

```
points[0]  = “face center”                    points[11] = “right eyebrow, left corner”
points[1]  = “right eye, left canthus”        points[12] = “right eyebrow, center”
points[2]  = “left eye, right canthus”        points[13] = “right eyebrow, right corner”
points[3]  = “mouth, right corner”            points[14] = “nose root”
points[4]  = “mouth, left corner”             points[15] = “nose left”
points[5]  = “right eye, right canthus”       points[16] = “nose right”
points[6]  = “left eye, left canthus”         points[17] = “mouth, center top”
points[7]  = “nose tip”                       points[18] = “mouth, center bottom”
points[8]  = “left eyebrow, left corner”      points[19] = “chin”
points[9]  = “left eyebrow, center”
points[10] = “left eyebrow, right corner”
```

By default, the landmarks are not computed by EyeFace SDK. The facial attribute recognition does not use them, so the only reason to compute landmarks is for visualization purposes. You can turn the computation on in the configuration file.

- **recognized**
  If set to “EF_TRUE”, the *points* and *angles* values are valid. Otherwise, they are zero initialized.
- **points**
  Positions of landmark points.
- **angles**
  Face orientation computed from landmarks in degrees. The order is “roll”, “pitch”, “yaw”. The “pitch” is not computed, always set to zero. The roll is only computed if rotations are enabled in the configuration file.

5.3.4 EfAge

```
typedef struct
{
    EfBool    recognized;
    double    value;
    double    response;
}EfAge;
```

Age recognition result type.

- **recognized**
  If set to “EF_TRUE”, the *value* and *response* values are valid. Otherwise, they are zero initialized.

- **value**
  Recognized age in years, in range 0-99 years.
- **response**
  Age classifier score function response (for data analysts / statisticians).

5.3.5 EfGender

```
typedef struct
{
    EfBool          recognized;
    EfGenderClass   value;
    double          response;
}EfGender;
```

Gender recognition result type.

- **recognized**
  If set to "EF_TRUE", the *value* and *response* values are valid. Otherwise, they are zero initialized.
- **value**
  Recognized gender, saved as *EfGenderClass* enumeration value.
- **response**
  Gender classifier score function response (for data analysts / statisticians).

5.3.6 EfEmotion

```
typedef struct
{
    EfBool          recognized;
    EfEmotionClass  value;
    double          response;
}EfEmotion;
```

Emotion recognition result type.

- **recognized**
  If set to "EF_TRUE", the *value* and *response* values are valid. Otherwise, they are zero initialized.
- **value**
  Recognized emotion, saved as *EfEmotionClass* enumeration value.
- **response**
  Emotion classifier score function response (for data analysts / statisticians).

5.3.7 EfAncestry

```
typedef struct
{
    EfBool            recognized;
    EfAncestryClass   value;
    double            response;
}EfAncestry;
```

Ancestry recognition result type.

- **recognized**
  If set to “EF_TRUE”, the *value* and *response* values are valid. Otherwise, they are zero initialized.
- **value**
  Recognized ancestry, saved as *EfAncestryClass* enumeration value.
- **response**
  Ancestry classifier score function response (for data analysts / statisticians).

5.3.8 EfFaceAttributes

```
typedef struct
{
    EfAge          age;
    EfGender       gender;
    EfEmotion      emotion;
    EfAncestry     ancestry;
}EfFaceAttributes;
```

Face attribute recognition result type. This type aggregates all face attributes in single structure.

- **age**
  Face attribute age result as *EfAge* type.
- **gender**
  Face attribute gender result as *EfGender* type.
- **emotion**
  Face attribute emotion result as *EfEmotion* type.
- **ancestry**
  Face attribute ancestry result as *EfAncestry* type.

5.3.9 EfTrackInfo

```
typedef struct
{
    EfTrackStatus       status;
    unsigned int        track_id;
    unsigned int        person_id;
    EfBoundingBox       image_position;
    double              world_position[2];
    double              angles[3];
    EfLandmarks         landmarks;
    EfFaceAttributes    face_attributes;
    double              energy;
    double              start_time;
    double              current_time;
    double              total_time;
    double              attention_time;
    EfBool              attention_now;
    int                 detection_index;
}EfTrackInfo;
```

Result structure containing visualization / statistics data for a single track. This structure is the main result in the EyeFace SDK Standard API.

- **status**
  Status of the track, whether it is alive or has just finished.
- **track_id**
  Unique identifier of a track. A track is a consecutive set of positions of detected face in a video sequence which belong to single person. A person can spawn multiple tracks with different id if the tracking is interrupted, for example if the person walks out of view for a while or turn his/her face in the direction opposite to the camera.
- **person_id**
  Unique identifier of a person. Our smart tracking feature can assign a unique person id to multiple tracks if the tracks are verified to be spawned by a single person. This is achieved both by spatial and temporal similarity of face position and a fast version of our face recognition engine. **It is not possible to store person templates. This technology is not intended for face recognition, but unique person counting. See "Eyedentify" for face recognition.**
- **image_position**
  Position of face in the image in pixels. The position is internally aggregated over past video frames.
- **world_position**
  Ground plane real world position of face relative to camera. The first dimension refers to left-right axis of camera, the second dimension to the depth (forward-backward). The value is stored in meters, internally aggregated over time. It is useful to measure distance of persons from camera, for example to select the closest person.
- **angles**
  Face orientation computed from face detector in degrees. The order is "roll", "pitch", "yaw". The "pitch" is not computed, always set to zero. The roll is only computed if rotations are enabled in the configuration file.
- **landmarks**
  Facial landmark points. Not computed by default, must be turned on in configuration file.
- **face_attributes**
  Face attributes (age, gender, emotion, ancestry) result.
- **energy**
  Fade out energy of the track. Set to 1 if the track was verified by detection in the last frame, otherwise a value between 0 and 1, the longer without detection the lower value.
- **start_time**
  Track start time in seconds. Based on *frame_time*.
- **current_time**
  Current tracker time in seconds. Based on *frame_time*.
- **total_time**
  Track duration in seconds.
- **attention_time**
  Duration for which person's attention has been caught. Based on *angles*.
- **attention_now**
  Set to EF_TRUE if the person's attention is caught in current frame.
- **detection_index**
  For Expert API users only. Enables users to link results in *EfDetectionArray* to results in this structure. Set to -1 if this track has no corresponding face detection in the current frame,

otherwise set to index to variable *detections* in *EfDetectionArray*. This is useful to get non-aggregated face position in the current frame, e.g. for Eyedentify Face SDK.

5.3.10 EfTrackInfoArray

```
typedef struct
{
    unsigned int    num_tracks;
    EfTrackInfo*    track_info;
}EfTrackInfoArray;
```

Result type wrapping array of *EfTrackInfo* values.

- **num_tracks**
  Number of elements in *track_info* array.
- **track_info**
  Array of *EfTrackInfo* values.

5.3.11 EfLogToServerStatus

```
typedef struct
{
    EfBool          server_is_reachable;
    unsigned int    num_messages_ok;
    unsigned int    num_messages_failed;
}EfLogToServerStatus;
```

Information regarding “Log To Server” feature of EyeFace SDK. Only valid if the log is enabled in configuration file.

- **server_is_reachable**
  Set to EF_TRUE if the server set in configuration file is reachable.
- **num_messages_ok**
  Number of messages successfully sent to the server.
- **num_messages_failed**
  Number of messages failed to send to the server.

5.4 Functions

This part of the Developer’s Guide contains all information about functions available in EyeFace SDK Standard API. The functions must be linked via explicit (runtime) linking. To define pointer to functions, a well-defined set of function pointer types are defined. For each function, a prefix “fcn_” is used to define such type.

On error, the functions write the error information into the standard error stream. The error stream might not be directly accessible when using wrappers.

5.4.1 efInitEyeFace

Initializes the EyeFace SDK using the configuration filed supplied as argument, outputting EyeFace SDK state on success.

Specification

```
EfBool efInitEyeFace(const char* eyefacesdk_dir, const char* config_ini_dir,
                     const char* config_ini_filename, void** eyeface_state)
```

Inputs

- **eyefacesdk_dir**
  Path to EyeFace SDK's "eyefacesdk" directory. The referenced directory contains detection and recognition models required to initialize EyeFace SDK.
- **config_ini_dir**
  Path to a directory containing the configuration file.
- **config_ini_filename**
  Filename of the configuration file.

Outputs

- **eyeface_state**
  Pointer to the initialized EyeFace SDK state. This state hold all the information about current processing state. It is used as input to almost all functions.

Returns

- **EF_TRUE**
  EyeFace SDK was initialized successfully.
- **EF_FALSE**
  EyeFace SDK initialization failed. See the standard error stream for more information.

Description

Initializes EyeFace SDK engine and loads detection and recognition models from *eyefacesdk_dir*/models. Initializes the license sessions.

IMPORTANT: In case of multithreading, EyeFace SDK states must be initialized in a code part where it is guaranteed that only a single thread is running. That does not mean only threads running EyeFace SDK, but all application threads. Do not initialize EyeFace SDK states concurrently, but one by one in a single thread and then assign initialized module instances to the other threads.

Example

```
void* eyeface_state             = NULL;
std::string eyefacesdk_dir      = "../../eyefacesdk";
std::string config_ini_dir      = "../../eyefacesdk";
std::string config_ini_filename = "config.ini";

EfBool init_ok = efInitEyeFace(eyefacesdk_dir.c_str(), config_ini_dir.c_str(),
                               config_ini_filename.c_str(), &eyeface_state);
if (init_ok == EF_FALSE)
{
    // Handle errors
}
```

5.4.2 efShutdownEyeFace

Flushes internal buffers of EyeFace SDK.

Specification

```
void efShutdownEyeFace(void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Description

This function is called after the main loop of video processing is finished. It causes all internal buffers to flush, so that logs and track info can be aggregated for tracks that are still alive during the shutdown.

Example

```
efShutdownEyeFace(eyeface_state);
```

5.4.3 efResetEyeFace

Reset the EyeFace SDK state.

Specification

```
void efResetEyeFace(void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Description

This function causes the EyeFace SDK state to reset, allowing to process the next video sequence without a need for time consuming initialization.

Example

```
efResetEyeFace(eyeface_state);
```

5.4.4 efFreeEyeFace

Release memory allocated during EyeFace SDK initialization.

Specification

```
void efFreeEyeFace(void** eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to the pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Description

Releases resources allocated by *efInitEyeFace*. Releases the license sessions. This function must be called from a single thread, when all threads using the corresponding *eyeface_state* have been suspended or terminated.

Example

```
efFreeEyeFace(&eyeface_state);
```

5.4.5 efMain

Main processing function of EyeFace SDK Standard API.

Specification

```
EfBool efMain(ERImage image, EfBoundingBox* bounding_box, double frame_time,
              void* eyeface_state)
```

Inputs

- **image**
  Input image. The image is expected to be part of a video sequence. Implementation is guaranteed not to write into *image*'s buffers.
- **bounding_box**
  Pointer to an orthogonal bounding box for selection of active area. Only active area is processed, resulting in computation speed improvements. In case the bounding box is not orthogonal, the function overwrites it with the smallest orthogonal bounding box which contains the original one. Set to NULL to process the whole image.
- **frame_time**
  Timestamp of the current frame in seconds, with millisecond precision. The first frame time is traditionally set to "0.000". The frame time must be unique for each frame and increasing.
- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Returns

- **EF_TRUE**
  Function processed the image successfully.
- **EF_FALSE**
  Image processing failed. See the standard error stream for more information.

Description

Run this function to process a single image of a video sequence. This function is always run in the main processing loop, where you acquire an image and process it by *efMain*. To retrieve the processing results, use *efGetTrackInfo* function.

Example

```
EfBool run_ok = efMain(image, NULL, 0.000, eyeface_state);
```

5.4.6 efGetTrackInfo

Get results of processing the current frame.

Specification

```
EfBool efGetTrackInfo(EfTrackInfoArray* track_info_array, void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

- **track_info_array**
  Pointer to user allocated structure which is filled with results.

Returns

- **EF_TRUE**
  Results filled successfully.
- **EF_FALSE**
  Results acquisition failed. See the standard error stream for more information.

Description

After running *efMain*, run this function to obtain current results. The results are aggregated over time. Use the results for visualizations in the current frame.

Two types of results are available, based on the "status" field of *EfTrackInfo*. The first type is information about alive tracks, which can be visualized, the second type is information about track finished in the current frame. The finished information is traditionally used to log into file or server for statistical purposes of measuring the attendance. This way you will get single information of face attribute estimation per every confirmed track in the video sequence.

The result must be freed after use by *efFreeTrackInfo*.

Example

```
EfTrackInfoArray track_info_array;
EfBool get_ok = efGetTrackInfo(&track_info_array, eyeface_state);
```

5.4.7 efFreeTrackInfo

Release memory allocated by *efGetTrackInfo*.

Specification

```
void efFreeTrackInfo(EfTrackInfoArray* track_info_array, void* eyeface_state)
```

Inputs

- **track_info_array**
  Pointer to user allocated structure filled by *efGetTrackInfo* to be cleared.
- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Description

Run this function to clear the data allocated by *efGetTrackInfo*. The memory pointed to by "track_info_array" itself is not freed, because it is user allocated.

Example

```
EfTrackInfoArray track_info_array;
EfBool get_ok = efGetTrackInfo(&track_info_array, eyeface_state);
// ...
efFreeTrackInfo(&track_info_array, eyeface_state);
```

5.4.8 efLogToServerGetConnectionStatus

Get information regarding "Log To Server" feature.

Specification

```
EfBool efLogToServerGetConnectionStatus(EfLogToServerStatus* connection_status,
                                        void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

- **connection_status**
  Pointer to user allocated structure which is filled with "Log To Server" statistics.

Returns

- **EF_TRUE**
  Statistics filled successfully.
- **EF_FALSE**
  Statistics acquisition failed. See the standard error stream for more information.

Description

Run this function to get statistics of the server logging. The server logging must be turned on in configuration file.

Example

```
EfLogToServerStatus connection_status;
EfBool get_ok = efLogToServerGetConnectionStatus(&connection_status, eyeface_state);
```

5.4.9 efGetKeyID

Get current license key ID after call to *efInitEyeFace*.

Specification

```
long long efGetKeyID(void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Returns

- **License key ID.**
  Key ID retrieved successfully.
- **-1**
  Key ID retrieval failed.

Description

Run this function to get the license key ID currently used by the eyeface_state.

6 EyeFace SDK Expert API

This chapter describes the Expert API of EyeFace SDK. The Expert API enables users to process image databases as well as video sequences. Opposite to the way information is gathered in Standard API, where it is aggregated over time, in Expert API you can process single images with non-aggregated results for precise localization and face attribute estimation. This way, you are given much more control over the data.

None of the functions can be considered thread safe.

6.1 Constants

6.1.1 EF_FACEATTRIBUTES_*

The Expert API contains the face attributes constant flags, which control the face attributes computation in *efRecognizeFaceAttributes*. The constants can be aggregated using bitwise *or* function. To compute all attributes, use "EF_FACEATTRIBUTES_ALL". In current version, there is no time saving in computing only selected attributes, because all attributes are computed using a single deep network model.

The face attributes options are the following:

```
EF_FACEATTRIBUTES_AGE           = 0x01
EF_FACEATTRIBUTES_GENDER        = 0x02
EF_FACEATTRIBUTES_EMOTION       = 0x04
EF_FACEATTRIBUTES_ANCESTRY      = 0x08
EF_FACEATTRIBUTES_SMARTTRACKING = 0x10
EF_FACEATTRIBUTES_ALL           = 0xFFFFFFFF
```

6.2 Enumerators

EyeFace SDK Expert API does not contain any enumerators.

6.3 Structures

6.3.1 EfPosition

```
typedef struct
{
    EfBoundingBox    bounding_box;
    double           center_col;
    double           center_row;
    double           size;
}EfPosition;
```

Face position in image as returned by face detector.

- **bounding_box**
  Face location as bounding box.
- **center_col**
  Face location center in column coordinate.

- **center_row**
  Face location center in row coordinate.
- **size**
  Face detection size as length of bounding box edge.

6.3.2 EfDetection

```
typedef struct
{
    double          confidence;
    EfPosition      position;
    double          angles[3];
}EfDetection;
```

Face detection return type. Contains information regarding a single face.

- **confidence**
  Score of the face detector, the higher the score the higher confidence that the detected object is a face. The minimal confidence for face to be detected can be set via threshold in configuration file.
- **position**
  Detected face location in image coordinates.
- **angles**
  Face orientation computed from detection in degrees. The order is “roll”, “pitch”, “yaw”. The “pitch” is not computed, always set to zero. The roll is only computed if rotations are enabled in the configuration file.

6.3.3 EfDetectionArray

```
typedef struct
{
    unsigned int    num_detections;
    EfDetection*    detections;
}EfDetectionArray;
```

Result type wrapping array of *EfDetection* values. One element per detected face.

- **num_detections**
  Number of elements in *detections* array.
- **detections**
  Array of *EfDetection* values.

6.3.4 EfLandmarksArray

```
typedef struct
{
    unsigned int    num_detections;
    EfLandmarks*    landmarks;
}EfLandmarksArray;
```

Result type wrapping array of *EfLandmarks* values. One element per detected face.

- **num_detections**
  Number of elements in *landmarks* array. The array is order the same as *EfDetectionArray*.

- **landmarks**
  Array of *EfLandmarks* values.

6.3.5 EfFaceAttributesArray

```
typedef struct
{
    unsigned int         num_detections;
    EfFaceAttributes*    face_attributes;
}EfFaceAttributesArray;
```

Result type wrapping array of *EfFaceAttributes* values. One element per detected face.

- **num_detections**
  Number of elements in *face_attributes* array. The array is order the same as *EfDetectionArray*.
- **face_attributes**
  Array of *EfFaceAttributes* values.

6.4 Functions

6.4.1 efRunFaceDetector

Run face detector on a single image.

Specification

```
EfBool efRunFaceDetector(ERImage image, EfDetectionArray* detection_array,
                         void* eyeface_state)
```

Inputs

- **image**
  Input image. The image can be a database image or part of an image sequence. Implementation is guaranteed not to write into *image*'s buffers.
- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

- **detection_array**
  Pointer to a user allocated structure which is filled with face detection results.

Returns

- **EF_TRUE**
  Face were detected successfully.
- **EF_FALSE**
  Face detection failed. See the standard error stream for more information.

Description

This function is used to detect faces on an image, a first step to recognize statistics of the faces either in image database or in a video sequence. The function is internally multi-threaded, the number of

threads can be set via configuration file. The face detection results are inputs to functions responsible for tracking, computing landmarks and face attributes. The output structure is filled with dynamically allocated data which must be freed using *efFreeDetections*.

Example

```
EfDetectionArray detection_array;
EfBool detection_ok = efRunFaceDetector(image, detection_array, eyeface_state);
```

6.4.2 efFreeDetections

Release memory allocated by *efRunFaceDetector*.

Specification

```
void efFreeDetections(EfDetectionArray* detection_array, void* eyeface_state)
```

Inputs

- **detection_array**
  Pointer to user allocated structure filled by *efRunFaceDetector* to be cleared.
- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Description

Run this function to clear the data allocated by *efRunFaceDetector*. The memory pointed to by "detection_array" itself is not freed, because it is user allocated.

Example

```
EfDetectionArray detection_array;
EfBool detection_ok = efRunFaceDetector(image, &detection_array, eyeface_state);
// ...
efFreeDetections(&detection_array, eyeface_state);
```

6.4.3 efUpdateTracker

Merges face detection into face tracks in image sequences.

Specification

```
EfBool efUpdateTracker(ERImage image, EfDetectionArray detection_array,
                       double frame_time, void* eyeface_state)
```

Inputs

- **image**
  Input image. The image must be a part of image sequence. Implementation is guaranteed not to write into *image*'s buffers.
- **detection_array**
  Array of face detections, acquired via a call to *efRunFaceDetector*. The implementation is guaranteed not to write into *detection_array*'s detections.
- **frame_time**
  Timestamp of the current frame in seconds, with millisecond precision. The first frame

time is traditionally set to “0.000“. The frame time must be unique for each frame and increasing.

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

No outputs, run for side effects.

Returns

- **EF_TRUE**
  Faces were tracked successfully.
- **EF_FALSE**
  Face tracking failed. See the standard error stream for more information.

Description

Face tracks are chains of face detection in time. Face detections are merged based on time and space localization of face in the image and face appearance (based on tracking). Every time faces are detected in an image coming from a video sequence, this function must be called to assign the detections to tracks immediately after *efRunFaceDetector*. Do not use this function if you want to process image database!

This function is run only for side effect. After running this function, you can call *efRunFaceLandmarks* and *efRecognizeFaceAttributes* or use the logging functions. The aggregated tracking results can then be recovered via *efGetTrackInfo*.

Example

```
EfDetectionArray detection_array;
EfBool detection_ok = efRunFaceDetector(image, &detection_array, eyeface_state);
EfBool tracker_ok   = efUpdateTracker(image, detection_array, 0.000, eyeface_state);
// ...
efFreeDetections(&detection_array, eyeface_state);
```

6.4.4 efRunFaceLandmark

Compute face landmarks on the detected faces.

Specification

```
EfBool efRunFaceLandmark(ERImage image, EfDetectionArray detection_array,
                         EfBool* detections_to_process,
                         EfLandmarksArray* facial_landmarks_array,
                         void* eyeface_state)
```

Inputs

- **image**
  Input image. The image must be a part of image sequence. Implementation is guaranteed not to write into *image*’s buffers.
- **detection_array**
  Array of face detections, acquired via a call to *efRunFaceDetector*. The implementation is guaranteed not to write into *detection_array*’s detections.

- **detections_to_process**
  Array of Boolean values. If the nth element is set to EF_TRUE, the landmarks on the nth face in *detection_array* is going to be computed. Set to NULL to process all faces.
- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

- **facial_landmarks_array**
  User allocated structure, which will be filled with face landmarks results. The face landmarks belong to the face with the same index in the *detection_array*. Must be cleared by *efFreeLandmarks*.

Returns

- **EF_TRUE**
  Face landmarks were computed successfully.
- **EF_FALSE**
  Face landmarks computation failed. See the standard error stream for more information.

Description

Face landmarks are discriminative points on face, like eye corners, mouth corners and so on. In EyeFace SDK, the landmarks are used for visualization purposes only.

This function must be called after efUpdateTracker in case you process a video sequence. As a side effect, the result is stored to tracks as well as to output array.

By default, landmark computation is turned off, switch this function on in the configuration file.

Example

```
EfDetectionArray detection_array;
EfBool detection_ok = efRunFaceDetector(image, &detection_array, eyeface_state);
EfBool tracker_ok   = efUpdateTracker(image, detection_array, 0.000, eyeface_state);
EfLandmarksArray landmarks_array;
EfBool landmark_ok  = efRunFaceLandmark(image, detection_array, NULL, &landmarks_array,
                                        eyeface_state);
// ...
efFreeDetections(&detection_array, eyeface_state);
```

6.4.5 efFreeLandmarks

Clear EfLandmarksArray filled by efRunFaceLandmark.

Specification

```
void efFreeLandmarks(EfLandmarksArray* facial_landmarks_array, void* eyeface_state)
```

Inputs

- **facial_landmarks_array**
  Pointer to user allocated structure filled by *efRunFaceLandmark* to be cleared.
- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Description

Run this function to clear the data allocated by *efRunFaceLandmark*. The memory pointed to by "facial_landmarks_array" itself is not freed, because it is user allocated.

Example

```
EfDetectionArray detection_array;
EfBool detection_ok = efRunFaceDetector(image, &detection_array, eyeface_state);
EfBool tracker_ok   = efUpdateTracker(image, detection_array, 0.000, eyeface_state);
EfLandmarksArray landmarks_array;
EfBool landmark_ok  = efRunFaceLandmark(image, detection_array, NULL, &landmarks_array,
                                        eyeface_state);
// ...
efFreeLandmarks(&landmarks_array, eyeface_state);
efFreeDetections(&detection_array, eyeface_state);
```

6.4.6 efRecognizeFaceAttributes

Recognize face attributes (age, gender, emotions, ancestry) of the detected faces.

Specification

```
EfBool efRecognizeFaceAttributes(ERImage image, EfDetectionArray detection_array,
                                 const EfLandmarksArray* facial_landmarks_array,
                                 EfBool* detections_to_process,
                                 unsigned int request_flag, double frame_time,
                                 EfBool process_sequentially,
                                 EfFaceAttributesArray* face_attributes_array,
                                 void* eyeface_state)
```

Inputs

- **image**
  Input image. The image must be a part of image sequence. Implementation is guaranteed not to write into *image*'s buffers.
- **detection_array**
  Array of face detections, acquired via a call to *efRunFaceDetector*. The implementation is guaranteed not to write into *detection_array*'s detections.
- **facial_landmarks_array**
  Array of face landmarks, acquired via a call to *efRunFaceLandmark*. Set to NULL in the current version.
- **detections_to_process**
  Array of Boolean values. If the nth element is set to EF_TRUE, the face attributes on the nth face in *detection_array* are going to be computed. Set to NULL to process all faces.
- **request_flag**
  Select which face attributes to compute. Set to *EF_FACEATTRIBUTES_ALL*.
- **frame_time**
  Timestamp of the current frame in seconds, with millisecond precision. The first frame time is traditionally set to "0.000". The frame time must be unique for each frame and increasing.
- **process_sequentially**
  Set to EF_FALSE to process video sequences. This way, the face attributes, which are very expensive to compute using deep networks, are computed in the background and the

control is returned to the user immediately. The results are written to tracks in one of the next calls of this functions. The results are not written to face_attributes_array.
Set to EF_TRUE to process image databases. The face attributes are computed in place and the result is stored into face_attributes_array. The function might take a long time to execute, approximately 100 milliseconds per face divided by number of threads assigned. Number of internal threads this function should use can be set in the configuration file.

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

- **face_attributes_array**
  User allocated structure, which will be filled with face attributes results. The face attributes belong to the face with the same index in the *detection_array*. Must be cleared by *efFreeAttributes*.

Returns

- **EF_TRUE**
  Face attributes were computed successfully / Face attributes were inserted into processing queue successfully – based on sequential flag.
- **EF_FALSE**
  Face attributes computation failed. See the standard error stream for more information.

Description

Compute face attributes on the selected faces. There are two modes of operation. For video sequence processing, the sequential flag must be set to EF_FALSE. The face attributes are then computed in the background and the result is later aggregated into tracks. In case of sequential processing, intended to process image databases, the function waits for all the face attributes to be computed and the results is filled into *EfFaceAttributesArray* structure.

6.4.7 efFreeAttributes

Clear EfFaceAttributesArray filled by efRecognizeFaceAttributes.

Specification

```
void efFreeAttributes(EfFaceAttributesArray* face_attributes_array,
                      void* eyeface_state)
```

Inputs

- **face_attributes_array**
  Pointer to user allocated structure filled by *efRecognizeFaceAttributes* to be cleared.
- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Description

Run this function to clear the data allocated by *efRecognizeFaceAttributes*. The memory pointed to by “face_attributes_array” itself is not freed, because it is user allocated.

Example

```
EfDetectionArray detection_array;
EfBool detection_ok  = efRunFaceDetector(image, &detection_array, eyeface_state);
EfBool tracker_ok    = efUpdateTracker(image, detection_array, 0.000, eyeface_state);
EfLandmarksArray landmarks_array;
EfBool landmark_ok   = efRunFaceLandmark(image, detection_array, NULL, &landmarks_array,
                                         eyeface_state);
EfFaceAttributesArray face_attributes_array;
EfBool attributes_ok = efRecognizeFaceAttributes(image, detection_array, NULL, NULL,
                                                 EF_FACEATTRIBUTES_ALL, 0.000, EF_TRUE,
                                                 &face_attributes_array, eyeface_state);
// ...
efFreeAttributes(&face_attributes_array, eyeface_state);
efFreeLandmarks(&landmarks_array, eyeface_state);
efFreeDetections(&detection_array, eyeface_state);
```

6.4.8 efLogToFileWriteTrackInfo

Log track information to a file.

Specification

```
EfBool efLogToFileWriteTrackInfo(void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

No output, run for a side effect.

Returns

- **EF_TRUE**
  Track information written to log file successfully.
- **EF_FALSE**
  Writing track information to log file failed. See the standard error stream for more information.

Description

This function writes track information to a log file. The feature must be turned on in the configuration file. The log functions must be the last of EyeFace calls in the processing loop. In Standard API, this function is called in the efMain. The information is written in JSON format as a stringified *EfTrackInfoArray*.

6.4.9 efLogToServerSendPing

Send a ping to a remote server. Used to verify remote connection.

Specification

```
EfBool efLogToServerSendPing(void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

No output, run for a side effect.

Returns

- **EF_TRUE**
  Ping send to server successfully.
- **EF_FALSE**
  Sending ping to server failed. See the standard error stream for more information.

Description

This function sends ping to a remote server to verify the connection is up and running. The feature must be turned on in the configuration file. The log functions must be the last of EyeFace calls in the processing loop. In Standard API, this function is called in the efMain every 15 seconds.

6.4.10 efLogToServerSendTrackInfo

Log track information to a remote server.

Specification

```
EfBool efLogToServerSendTrackInfo(void* eyeface_state)
```

Inputs

- **eyeface_state**
  Pointer to EyeFace SDK state, initialized by *efInitEyeFace()*.

Outputs

No output, run for a side effect.

Returns

- **EF_TRUE**
  Track information send to server successfully.
- **EF_FALSE**
  Sending track information to server failed. See the standard error stream for more information.

Description

This function sends track information to a remote server. The feature must be turned on in the configuration file. The log functions must be the last of EyeFace calls in the processing loop. In Standard API, this function is called in the efMain. The information is written in JSON format as a stringified *EfTrackInfoArray*.

7 EyeFace SDK Examples

EyeFace SDK contains several examples to help the client grasp all its functionality as fast as possible. The examples come with prepared Microsoft Visual Studio Solution for Windows and Makefiles for Linux distributions. This chapter explains step by step how to build EyeFace SDK examples on different platforms and how to install OpenCV library to run the example which uses a camera. The OpenCV is not mandatory to use EyeFace SDK, the client can skip the OpenCV examples.

7.1 Overview

There are several examples in EyeFace SDK distribution. They are in **[EyeFaceSDK]/examples** directory, and are divided into several categories. There are examples processing a video sequence, image database, stream from camera and auxiliary examples showing the licensing API and a simple remote logging server.

7.1.1 Example-Hello-EyeFace

This is a "Hello World!" type example, showing the basic functionality of EyeFace SDK. The example is described in detail in Chapter 5.

7.1.2 Example-API

This tutorial contains two examples, both showing the EyeFace SDK API. The first one shows how to use the Standard API to detect and recognize faces easily. The goal is to process images taken from a continuous image sequence (video). The main processing function detects faces in each image, runs face attribute recognition (age, gender, emotion, ancestry) and tries to connect the current face detections with detections in previous frames, i.e. a simple tracker is applied. Smart tracking is applied to join tracks of same individual over time. Results corresponding to each track are on the output. This API is used for live demo applications.

The other example uses the Expert API, showing how the client can get a higher level of control over the processing. Single images as well as image sequences can be processed. The example shows how to process an image database.

7.1.3 Example-OpenCV

In this example, the client will build a simple GUI application that processes a video stream from camera using OpenCV library. In Chapter 4.2, a guide is provided on how to build OpenCV in the client's platform of choice. OpenCV installation is not mandatory to use EyeFace SDK.

7.1.4 Example-LogServer

This is an auxiliary example, that can be used with one of the previous examples. When remote logging is turned on (see Chapter 8.8), this example shows the messages sent in communication between EyeFace SDK running application and this logging server.

7.1.5 Example-GetInfo

The last example shows the software licensing API usage. It enables the client to list his license keys, generate licensing requests and attach licenses to a target machine. This is important to check license availability before linking the EyeFace SDK.

7.2 Configuring OpenCV

To build the EyeFace SDK's Example OpenCV, the OpenCV library needs to be installed on client's computer. The installation differs for Linux and Windows. OpenCV installation is not mandatory to use EyeFace SDK. We use the OpenCV version 3 **world** library, which unifies all OpenCV stuff in one file.

7.2.1 Configuring OpenCV on Windows

It is expected that the client uses Microsoft Visual Studio 2015. The client should follow these steps to configure OpenCV on Windows:

- Download OpenCV 3.2 or newer for Windows from GitHub
- Run the downloaded executable, extract the library into any directory (**OPENCV_INST_DIR**)
- On x64 version, the binaries are prebuilt, on win32, the client must build OpenCV using CMake
- Add environment variable, referencing the directory of OpenCV, based on the target machine architecture, set OPENCV_DIR as **OPENCV_INST_DIR\build**
- Append path to OpenCV's DLL libraries to "PATH" environment variable, based on the target machine architecture
- for 32 bit, append **OPENCV_INST_DIR\build\x86\vc14\bin**
- for 64 bit, append **OPENCV_INST_DIR\build\x64\vc14\bin**

To add or edit an environment variable on Windows, open "Control Panels > System and Security > System > Advanced System Settings", and click "Environment Variables..." button to open a dialog window. There you can modify the existing environment variables or create new ones.

7.2.2 Building and Configuring OpenCV on Linux

The client should follow these steps to configure OpenCV on Linux:

- Download OpenCV 3.2 or newer Source Code from GitHub
- Extract the tar archive
- On Linux, download the libgtk2.0-dev and v4l2-dev packages
- Download libav packages; libavcodec-dev, libavutil-dev, libavformat-dev, libswscale-dev
- Download cmake-gui package or installer at http://www.cmake.org
- Using cmake-gui, click Configure (choose different directory) and then click Generate
- In the generated directory, run **"make"** from the command line
- In the generated directory, run **"sudo make install"** from the command line
- Check that the OpenCV files were installed in **/usr/local/include** and **/usr/local/lib**
- Set the LD_LIBRARY_PATH environment variable to point to **/usr/local/lib**
- run **"export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib"**
- check if the path was exported, run **"echo $LD_LIBRARY_PATH"**

7.3 Building Examples

Building examples with EyeFace SDK is easy. On Linux, every example has a Makefile in its directory. The example is built by executing **"make"** command from the command line. The built binary is outputted to the example's directory.

On Windows, there is a Visual Studio project available for each example. The newly created executable is copied together with all dependencies to the example's directory. There is a dependency of the

linker setting in example-opencv on the version of OpenCV installed. If the client does not use OpenCV 3.2, the linker input settings must be updated with a newer version of OpenCV import libraries.

The binaries created are dependent on relative path to **[EyeFaceSDK]/eyefacesdk** directory. The binaries will therefore not work on their own. Additional data and models are always loaded from the filesystem on EyeFace SDK initialization. The client should always copy **[EyeFaceSDK]/eyefacesdk** together with the prebuilt examples or apps, preserving their relative paths.

7.4 Example-OpenCV Window

After building the "Example-OpenCV", the example can be executed. It displays a window with visualizations of face detection and face attribute recognition.

The visualizations are described in Illustration 2.

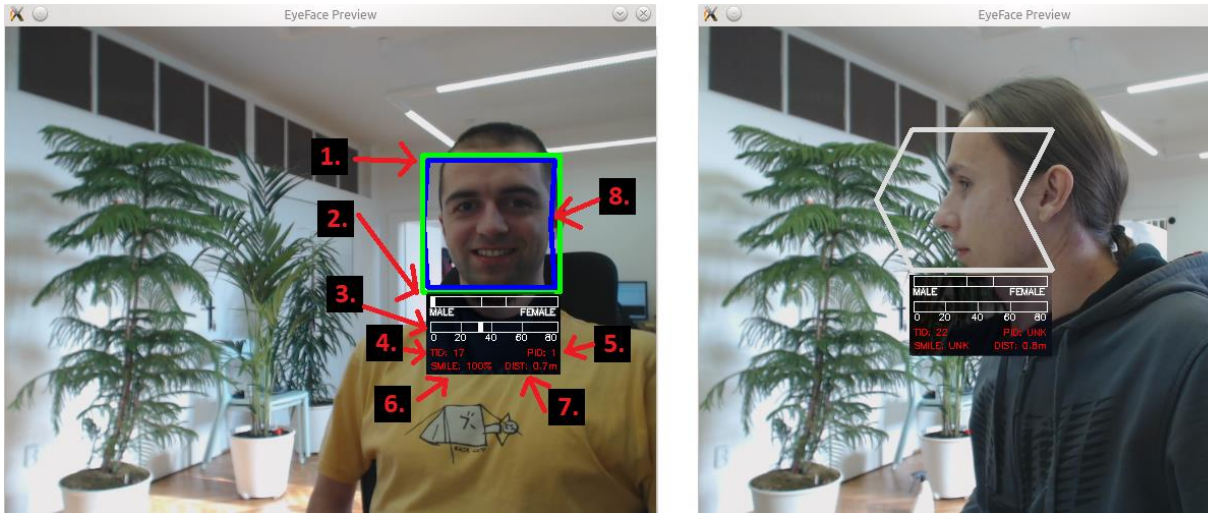


Illustration 2: Example-OpenCV window. (Left) Frontal face detection with face attributes estimated. [1.] Attention detection. [2.] Gender. [3.] Age. [4]. Track ID. [5.] Unique ID. [6.] Smile detection. [7.] Distance from camera. [8.] Yaw angle arrow shape. (Right) Profile face detection, yaw angle shown by the arrow shape of detection bounding box. Statistics are not detected on profile faces.

8 Hello EyeFace!

In this chapter, the simplest example will be described, resembling the common "Hello World!" example. The "Hello EyeFace!" example is in **[EyeFaceSDK]/examples/example-hello-eyeface/**. It will be shown how to read image from a file, how to initialize EyeFace SDK engine, and how to run the face attribute recognition on a video sequence. This chapter will also show how to create the explicit linking framework, necessary to use EyeFace SDK with software licensing protection. Information about all the functions and data structures used can be found in *Doxygen* documentation.

8.1 Explicit Linking Explained

The explicit (runtime) linking is a method of using shared libraries in C-like programming languages. The distribution of EyeFace SDK contains encrypted shared library and standard implicit linking is not possible. In contrast to implicit linking, which is the common way when the actual linking is done during build time, the explicit linking takes places during the runtime. The explicit linking itself is platform dependent, but EyeFace SDK solves this issue by using its own cross-platform explicit linking macros.

First thing when it comes to explicit linking is to open the shared library at runtime. Please note that on Windows, you must manually load the “libopenblas.dll” via ER_OPEN_SHLIB prior to loading EyeFace SDK.

```
#include “EyeFace.h”                          // Header file with EyeFace SDK Standard API
shlib_hnd lib_handle;                        // Handle to the shared library
std::string lib_path = “path/to/xyz.dll”;    // Path to the shared library

ER_OPEN_SHLIB(lib_handle, lib_path.c_str());    // Load the shared library from file
```

The library is opened if the lib_handle is not NULL after execution of these lines. Next step is to assign the pointers to functions that will be used in our code. The following example also shows naming conventions for pointer to function type, pointer to function and function name used in EyeFace SDK.

```
fcn_efFunctionName fcnEfFunctionName = NULL;     // Pointer to a function variable
ER_LOAD_SHFCN(fcnEfFunctionName, fcn_efFunctionName,
              lib_handle, “efFunctionName”);     // Assign function pointer
```

The library function pointer is successfully recovered if it is not NULL. Now the function pointer can be used to call a function. The syntax overloading allows to use the pointer as a standard function.

```
int ret_val = fcnEfFunctionName(parameters);     // Function call.
```

It is not recommended to unload the EyeFace SDK library at runtime.

8.2 Global Initialization

The first lines of code of "Hello EyeFace!" source contains includes and paths to EyeFace SDK, its configuration file and path to the image to be processed.

```
#include <iostream>                             // Header file for I/O
#include <string>                               // Header file for string
#include “EyeFace.h”                            // Header file with EyeFace SDK Standard API

const std::string EYEFACE_DIR = “../../eyefacesdk”;   // Path to EyeFace SDK (also for config)
const std::string CONFIG_INI  = “config.ini”;         // Configuration file filename

ER_OPEN_SHLIB(lib_handle, lib_path.c_str());     // Load the shared library from file
```

Next, the path to explicitly linked shared library is defined. The path depends on the actual operating system and architecture of client's machine.

```
#if defined(WIN32) || defined(_WIN32) || defined(_DLL) || defined(_WINDOWS_)
// Windows
std::string shlib_path = "..\\..\\eyefacesdk\\lib\\EyeFace.dll";
#else
// Linux
#if defined(x86_32) || defined(__i386__)
// 32 bit Linux
std::string shlib_path = "../../eyefacesdk/lib/libeyefacesdk-x86_32.so";
#else
// 64 bit Linux
std::string shlib_path = "../../eyefacesdk/lib/libeyefacesdk-x86_64.so";
#endif
#endif
```

Next, a handle to the shared library and pointers to functions are declared.

```
shlib_hnd shlib_handle;                         // Handle to the shared library

fcn_erImageRead     fcnErImageRead = NULL;      // Handles to functions
fcn_erImageFree     fcnErImageFree = NULL;
fcn_efInitEyeFace   fcnEfInitEyeFace = NULL;
fcn_efFreeEyeFace   fcnEfFreeEyeFace = NULL;
fcn_efMain          fcnEfMain = NULL;
fcn_efGetTrackInfo  fcnEfGetTrackInfo = NULL;
fcn_efFreeTrackInfo fcnEfFreeTrackInfo = NULL;
```

Finally, an image path which will simulate a video sequence with static image is defined.

```
const std::string IMAGE_FILENAME = "../../data/test-images-id/0000.png";
```

8.3 Loading EyeFace SDK Shared Library

After the global initialization, the main function is implemented. It starts with the explicit linking of EyeFace SDK shared library.

```
int main()
{
    // Load shared library - explicit linking.
    ER_OPEN_SHLIB(shlib_handle, shlib_path.c_str());

    // Get pointers to functions from loaded library.
    ER_LOAD_SHFCN(fcnErImageRead, fcn_erImageRead, shlib_handle, "erImageRead");
    ER_LOAD_SHFCN(fcnErImageFree, fcn_erImageFree, shlib_handle, "erImageFree");
    ER_LOAD_SHFCN(fcnEfInitEyeFace, fcn_efInitEyeFace, shlib_handle, "efInitEyeFace");
    ER_LOAD_SHFCN(fcnEfFreeEyeFace, fcn_efFreeEyeFace, shlib_handle, "efFreeEyeFace");
    ER_LOAD_SHFCN(fcnEfMain, fcn_efMain, shlib_handle, "efMain");
    ER_LOAD_SHFCN(fcnEfGetTrackInfo, fcn_efGetTrackInfo, shlib_handle, "efGetTrackInfo");
    ER_LOAD_SHFCN(fcnEfFreeTrackInfo, fcn_efFreeTrackInfo, shlib_handle, "efFreeTrackInfo");
```

8.4 Initializing EyeFace SDK Engine

Before running the face detection and recognition, EyeFace SDK engine must be initialized. The initialization proceeds as follows. The paths to /eyefacesdk directory are important.

```
void * eyeface_state = NULL;
EfBool init_success = fcnEfInitEyeFace(EYEFACE_DIR.c_str(), EYEFACE_DIR.c_str(),
                                       CONFIG_INI.c_str(), &eyeface_state);
```

8.5 Processing Loop

The processing of a video sequence is performed in a loop. The loop consists of preparing the image, running EyeFace SDK analytics and acquiring results. The end of the processing loop is used to release memory after use. In this example, we acquire the results only once after the processing loop is finished to get a single result for our virtual image sequence.

```
    // Run face detection and recognition on imaginary video sequence of 1 second with 25 fps.
    for (unsigned int i = 0; i < 25; i++)
    {
        // Load image. Imagine the sequence consists of this one image repeated several times.
        ERImage input_image;
        fcnErImageRead(&input_image, IMAGE_FILENAME.c_str());

        fcnEfMain(input_image, NULL, i / 25.0, eyeface_state);

        fcnErImageFree(&input_image);
    }
```

8.6 Getting the Results

After the detection and recognition, the results are requested using the following line of code. In the example, we only acquire the result for the last image, but in real life you will be getting the result in every frame inside the processing loop.

```
EfTrackInfoArray track_info_array;
fcnEfGetTrackInfo(&track_info_array, eyeface_state);
```

The results are stored in *EfTrackInfoArray* data structure. For example, number of detected faces can be extracted as follows.

```
std::cout << "Number of detected faces: " << track_info_array.num_tracks << "." << std::endl;
```

8.7 Freeing EyeFace SDK Resources

The resources used by EyeFace SDK must be freed after use. The following lines shows how to free the track info results and the EyeFace SDK engine itself. Note that typically the track info is release in the processing loop when using the *efGetTrackInfo* in the processing loop.

```
fcnEfFreeTrackInfo(&track_info_array, eyeface_state);
fcnEfFreeEyeFace(&eyeface_state);
```

8.8 Compiling and Linking

On Windows, use the Microsoft Visual Studio to build the example. On Linux use the Makefile to build the example.

8.9 Example Remarks

The "Hello EyeFace!" example is the simplest piece of code showing how to detect faces and recognize face attributes using EyeFace SDK. For the sake of clarity, most of the error checks were omitted. However, in a real application, one should always check the return value of functions for errors. To see the return values correctly handled, the client is advised to check any of the other example of EyeFace SDK.

9 EyeFace SDK Wrappers

EyeFace SDK contains a range of various wrappers to provide developers with easy access to the actual functionality and lift the burden of being unable to use the environment of their choice. In this chapter, a description of the interfaces will be given together with instructions on how to prepare them for use. The package currently contains these wrappers:

- C#
- Java
- Matlab/Octave MEX

9.1 C# Wrapper

The C# wrapper requires no actions on the client's side to be performed to use it. The wrapper together with an example Visual Studio solution is in **[EyeFaceSDK]/wrappers/csharp/** directory. The wrapper is only available in Windows packages.

The wrapper itself is in the *EfCsSDK.cs* and *ErCsSDK.cs* (ERImage API) source files which are in the folder *eyeface-cs-example* for user to be able to run the example without any modifications or copying.

The interface is composed from the C# class EfCsSDK, which covers all needed functions of the SDK, and several C# structures, which are copies of the structures contained in the C/C++ API.

9.2 Java Wrapper

The Java wrapper provides a JNI interface to the EyeFace SDK on all platforms, where the SDK is available. Part of the Java wrapper is the example project *eyeface-java-example* for Eclipse IDE, which demonstrates the basic usage of the SDK.

The wrapper package is in the **[EyeFaceSDK]/wrappers/java/** directory. The wrapper itself is contained in the JAR archive *eyeface-java-sdk.jar* and is in the folder *eyeface-java-example* for user to be able to run the example without any modifications or copying.

The wrapper contains the JNI native wrapper library which manages the communication between the Java API and the EyeFace SDK C/C++ library. The JNI library is automatically extracted to the system's temp folder during the runtime to be linked with the Java Virtual Machine. Extracted JNI library is automatically deleted from system's temp folder, when the JVM shuts down.

The Java wrapper sources and Windows x86/x86_64 JNI wrapper library were built on Windows using Java SE JDK version 1.8.0_131 available from Oracle Corporation and using Visual Studio 2015. Linux i686/x86_64 JNI wrapper library was built on Ubuntu 16.04 using Java SE JDK version 1.8.0_131 available from Oracle Corporation and using GCC version 5.4.

9.3 MEX Wrapper

The Matlab/Octave MEX wrapper is available for all platforms. It is in **[EyeFaceSDK]/wrappers/mex/** directory together with examples. Before use, the MEX files must be built using **make_eyeface_sdk_mex.m** script. It enables EyeFace SDK to be used in rapid prototyping environments to speed up development process.

10 EyeFace SDK Licensing

In this chapter, the steps needed to update the trial license to a full EyeFace SDK license will be explained. The individual licensing options, pricing and license terms are available at the websites.

10.1EyeFace SDK API's

EyeFace SDK features two application programming interfaces, based on the license used. The Standard API, used to process video streams, and the Expert API, primarily used to process image databases. The Expert API contains the whole interface of Standard API, and adds the advanced functionality for a higher level of control over the processing.

The functions and data types available in Standard API are listed in the "EyeFace.h" and "EyeFaceType.h" header files, located in **[EyeFaceSDK]/eyefacesdk/include**. The client using the EyeFace SDK Expert API can also use the functions and data types available in "EyeFaceExpert.h" and "EyeFaceExpertType.h". The main documentation sources for the API's available are the examples and the *Doxygen* documentation, located in **[EyeFaceSDK]/documentation/doxygen**. As a starting point for the *Doxygen* documentation, the client should use the **index.html** file.

The client may test the APIs available by switching from an Expert license to a Standard license in the configuration file, see Chapter 8.9. By default, the EyeFace SDK uses the Expert license, if available. If the client wants to test with the Standard license, the corresponding license key number must be set in the configuration file. To list the licenses and select the license key number with the Standard License, the client should open http://localhost:1947/_int_/devices.html, see Illustration 1, and list the products of the licenses available. The Standard license is the one containing product **Product 2**, as opposed to Expert license, which is the one containing product **Product 12**.

10.2License Management

This chapter is a must read before purchasing a license of EyeFace SDK. If not fully understood, unrepairable damage might be caused to the license keys by client's actions. Eyedea Recognition will charge additional costs to recover the license key if the client did not follow instructions in this document.

Eyedea Recognition provides two types of license keys together with EyeFace SDK. The first possibility is a **software key** (Sentinel SL) delivered to the client by email. The client must not think about the software key as a licensing number, but rather as a secret file. The software key can only be connected to a single fingerprinted machine (see next chapter). If the hardware is changed after the client sends the machine fingerprint to us and before he attaches the license, the licensing process will fail. The software key is stored as a file in a secret storage on the machines hard drive. The secret key is generated at attachment time, Eyedea Recognition does not have access to the secret key neither can create a new one. In case the client plans to change the hard drive, change partitions on the drive or reinstall operating system, the software license key must first be rehosted (moved) to another machine for safekeeping. If the file token is kept on the original machine during the mentioned procedures, it will be destroyed and it will not be possible to recover it. Thus, the software license scheme is recommended for permanent installations, not for development purposes and changing environment.

The other licensing option is a **USB dongle** (Sentinel HL) license key. This type of key is specifically targeted for developers, as it is not locked to a single machine, but can be used on any machine of the client, easily transferred from one machine to another and is not influenced by hardware or operating system changes. The downside is that it must be physically sent to the client, which takes time and additional charge for the dongle itself.

Please contact us to negotiate other possibilities of licensing.

10.3Generating a Licensing Request

EyeFace SDK features a machine locked license. Before Eyedea Recognition, Ltd. can create a license the client orders, a unique key of the target machine must be created. On Linux/Mac OS X, the client generates the unique key by executing

```
sudo [EyeFaceSDK]/hasp/gen-request
```

where **[EyeFaceSDK]** is the directory, where EyeFace SDK package was extracted. On Windows, the client generates the unique key by executing **[EyeFaceSDK]/hasp/gen-request** as **Administrator**. In the directory, from which the "gen-request" binary was executed, a unique key will be stored in "c2v-info.mc2v" file. The client should send this file to Eyedea Recognition's mailbox,

```
store@eyedea.cz
```

The email address used to send the file by the client must be the same as the one which was used for the purchase of EyeFace SDK license. The client should not send any licensing request unless a working trial of EyeFace SDK is run on the target machine without any conflict. The license key that will be shipped by Eyedea Recognition can only be attached to the same machine that generated the fingerprint.

10.4Attaching a New License

After a full license is received by the client, it must be attached to the Sentinel License Manager. The client should open the web browser at http://localhost:1947/_int_/checkin.html, see Illustration 3, click the "Choose file" button, select the received full license file (with .V2C suffix) and click "Apply File" to apply the license. The new license will be listed in the Sentinel License Manager's "Sentinel Keys" pane, see Illustration 1.

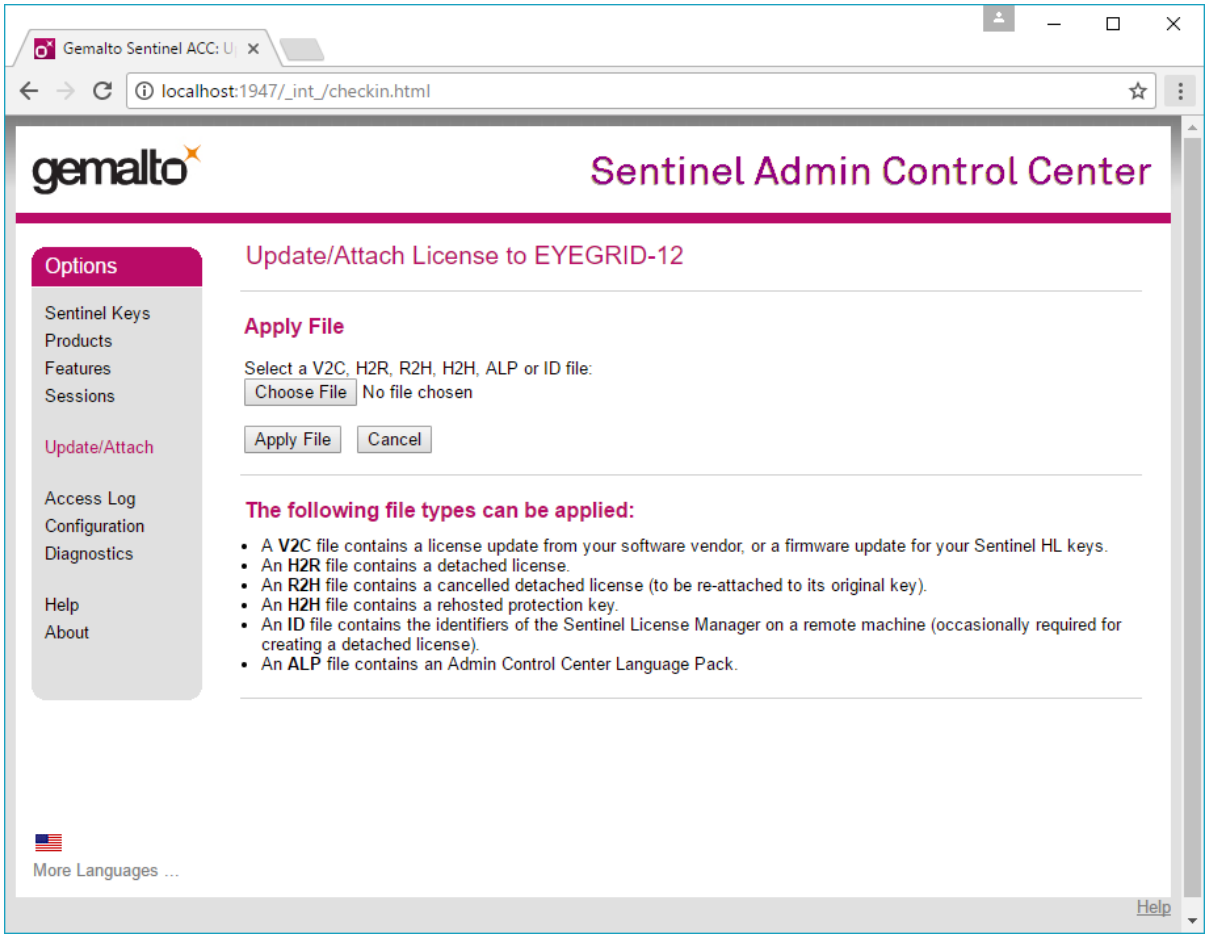


Illustration 3: Attaching a new EyeFace SDK license via Sentinel License Manager.

10.5Transferring License to Another Machine

The software license of EyeFace SDK can be transferred from one machine to another, the process is system independent, e.g. a license can be transferred from a Linux PC to a Windows PC. During the process, the license is detached from the machine, where it was previously installed (source machine), transferred as a text file (transfer license) to another machine (recipient machine) and attached to it.

- The client should complete the following steps to transfer a license:
- Generate a recipient ID file on the recipient machine
- Copy the recipient ID file to the source machine
- Create a transfer license on the source machine, using the recipient ID file
- Attach the transfer license on the recipient machine, see Chapter 7.4

During the process, the client has full responsibility for the license key. The license key will not be refunded in case that the client will corrupt or destroy the transfer license file, and so the transferring is not recommended as a standard action. To transfer the license on Linux, the client should use the **"rehost-tool"** binary located in **[EyeFaceSDK]/hasp**. On Windows, the client should use the **"RUS.exe"** executable located in the same directory. Additional information is available in the binaries.

10.6License Logout Failures

When using the Matlab MEX interface, a problem may arise if the EyeFace SDK is improperly shut down. During initialization, EyeFace SDK connects to the Sentinel License Manager. After EyeFace SDK engine is properly closed, the application disconnects from the Sentinel License Manager allowing other applications to use the license. The license is also returned to the pool when the application terminates. But when using a GUI, where EyeFace SDK code runs in the process of the GUI (e.g. Matlab), if the EyeFace SDK engine is ended without calling *efFreeEyeFace*, the license is not returned because the process is not closed. The license then becomes stuck and the client cannot run another instance of EyeFace SDK. The solution is to manually disconnect all the sessions from the Sentinel License Manager using the web interface at http://localhost:1947/_int_/sessions.html, see Illustration 4. The client should never disconnect a running EyeFace SDK instance. The EyeFace SDK periodically checks the connection, and if the connection was disconnected by the client, it will terminate.

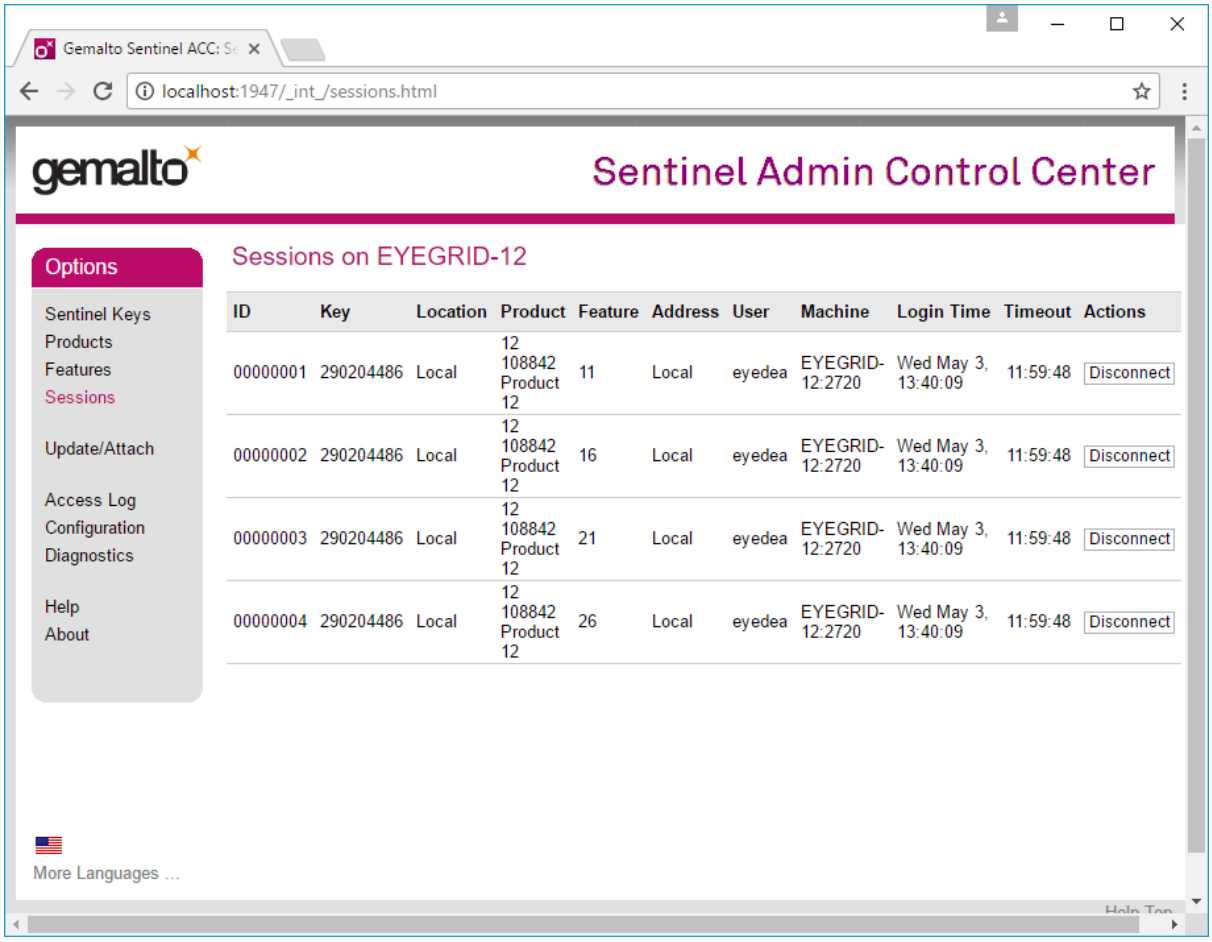


Illustration 4: Disconnecting license from Sentinel License Manager. Click the "Disconnect" button at all session items to release the license.

11 Configuration Parameters

This chapter focuses on the EyeFace SDK configuration file, used to setup the detection, attribute recognition, logging and auxiliary parameters, which is by default located in **[EyeFaceSDK]/eyefacesdk/config.ini**. Using the configuration file, the client can easily choose between precision and speed of processing, vary the CPU usage by managing the number of threads EyeFace SDK engine uses and among other possibilities also setup the logging to either a file or a server.

11.1Configuration File Syntax

The configuration file is managed using iniparser library. It is divided into paragraphs, created by paragraph name enclosed in square brackets as follows:

```
[DETECTOR]
```

The parameters themselves are positioned after the corresponding paragraph title. This simple config.ini example shows a commentary followed by two paragraphs, first containing two parameters and the second on containing a single parameter. The actual meaning of the parameters will be explained later in this chapter. Note the **mandatory** empty line at the end of the configuration file done by a commentary.

```
# This is a configuration file commentary.
[DETECTOR]
num_threads = 32
min_win_size = 24

[FACE ATTRIBUTES]
num_threads = 64
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

11.2EyeFace Parameters

These parameters describe global behavior of EyeFace SDK. You can setup the license key information as well as unique identifier of the running EyeFace SDK and turn on or off EyeFace SDK modules to save time.

A default EyeFace parameters setup is the following:

```
[EYEFACE]
license_id                = 0
device_id                 = 0
landmark_switch_on        = 0
face_attributes_switch_on = 1
log_to_file_switch_on     = 0
log_to_server_switch_on   = 0
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

The parameters are explained in Table 1.

license_id	Sentinel LDK software license ID. If set to 0, uses the default license key order provided by Sentinel License Manager.
device_id	Unique ID set by the client to possibly distinguish running instances.
landmark_switch_on	Switch the landmarks computation on or off.
face_attributes_switch_on	Switch the face attributes on or off.
log_to_file_switch_on	Switch log to file on or off.
log_to_server_switch_on	Switch log to server on or off.

Table 1: EyeFace SDK configuration file, EyeFace parameters.

11.3Face Detector Parameters

EyeFace SDK contains a machine learned face detector. The detector is highly configurable, so that the client can tune between performance and processing speed.

A default detector parameters setup is the following:

```
[DETECTOR]
num_threads  = 4
min_win_size = 48
shift_factor = 0.125
scale_factor = 1.2
num_scales   = 100
threshold    = 28.0
rotations    = []
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

The parameters are explained in Table 1.

num_threads	Number of threads used for face detection. The more the threads used (even more than number of cores multiplied by two in case of hyperthreading), the faster the execution and more CPU resources used. Minimal number is 1, maximal is limited by client's system.
min_win_size	Sizes of the smallest face detection window used. The face is not detected if it is smaller than the detection window. Minimal value is 24 pixels, maximal value is limited by the resolution of the input image.
shift_factor	Distance between two detection windows which will be evaluated. Relative to the size of the detection window. A value

	between 0 and infinity, the lower the value the more windows will be evaluated.
scale_factor	The face detection is performed in multiple scales. The scale factor is a multiplication constant, defining a size difference between subsequent scales. Minimal value must be larger than 1.
num_scales	Number of scales to detect faces in. Minimal value is 1. The size of maximal detection window (bounded by image size) is computed as max_win_size = min_win_size*scale_factor^(num_scales-1)
threshold	Face detection ROC curve threshold. Minimal value is 0. The higher the value, the less false positives but more false negatives.
rotations	Detection window rotations. A single rotation detects faces with roll angle of -15 to +15 degrees. You can add additional rotations as [-30, 0, 30] to detect higher spread of roll angles. The detection is linearly slower in the number of rotations.

Table 1: EyeFace SDK configuration file, detector parameters.

11.4Camera Parameters

EyeFace SDK can reconstruct a 2D world position of people in front of the camera from face detection. To convert the relative positions to actual meters, horizontal field of view of the camera must be set in the configuration file. By default, the field of view is set to 75 degrees.

```
[CAMERA]
fov = 75
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

The parameter is explained in Table 2.

fov	Horizontal field of view of the camera used, in degrees. Used for 2D world position reconstruction.

Table 2: EyeFace SDK configuration file, camera parameters.

11.5Face Attributes Recognition Parameters

EyeFace SDK uses deep networks to recognize face attributes like age, gender, emotion etc. You can set the number of threads the EyeFace SDK will use to process the attributes. The parallelization is on per face basis.

A default face attributes parameters setup is the following:

```
[FACE ATTRIBUTES]
num_threads  = 1
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

The parameters are explained in Table 1.

num_threads	Number of threads used for face attribute recognition. The more the threads used(even more than number of cores multiplied by two in case of hyperthreading), the faster the execution and more CPU resources used. Minimal number is 1, maximal is limited by client's system.

Table 1: EyeFace SDK configuration file, face attributes parameters.

11.6Tracking Parameters

EyeFace SDK contains a tracker to merge detections in subsequent images into tracks. The tracks aggregate information about detection and recognition in all previous frames. After a face disappears from the image, the tracks are stored. In case that a new face appears, the smart tracking tries to link it with previous tracks. The client may tune several parameters of the tracker.

The client can also detect attention and dwell time. The dwell time is simply the time the person (face) appeared in front of the camera and the attention time is the time the person looked in the desired direction. The client can set an angular threshold on yaw angle, so that if the yaw angle is less than the threshold, the person is said to be in attention.

A default face attributes parameters setup is the following:

```
[TRACKING]
max_time_in_buffer  = 300
buffer_size         = 200
attention_max_yaw   = 25
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

The parameters are explained in Table 1.

max_time_in_buffer	Number of seconds the track is logged after the face disappears. After the time expires, the tracker cannot link the person in the track to the same person in a new track.
buffer_size	Maximum number of tracks in the buffer. After the capacity is reached, the oldest tracks are removed to keep the maximum size.
attention_max_yaw	Maximal absolute yaw angle in degrees, when the person's attention is assumed to be caught.

Table 1: EyeFace SDK configuration file, tracking parameters.

11.7Log to File Parameters

Log to file enables logging of track info into a file. Information about finished tracks or possibly live tracks is logged in JSON format. The feature is off by default and can be enabled via configuration.

A default log to file parameters setup is the following:

```
[LOG TO FILE]
log_directory = "."
log_duration  = 0
use_live      = 0
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

The parameters are explained in Table 1.

log_directory	Absolute or relative (to the executable) path to the log file directory.
log_duration	Amount of time in hours, after which a new log file is created. The log file name is augmented with the log start time.
use_live	Log live tracks to the file as well.

Table 1: EyeFace SDK configuration file, log to file parameters.

11.8Log to Server Parameters

Log to server enables logging of track info into a remote server via HTTP POST requests. Information about finished tracks or possibly live tracks is logged in JSON format. The feature is off by default and can be enabled via configuration.

A default log to server parameters setup is the following:

```
[LOG TO SERVER]
address   = "https://94.230.156.166 "
port      = 8443
use_live  = 0
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

The parameters are explained in Table 1.

address	A string, network address identifying the target remote server.
port	Integer, a port of the target log server, where the server is listening.
use_live	Log live tracks to the server as well. Results in high network traffic.

Table 1: EyeFace SDK configuration file, log to server parameters.

11.9Initializing EyeFace SDK with Custom Configuration

To initialize EyeFace SDK with a custom configuration, create a new text file with the described format and input the path to custom configuration file to *efInitEyeFace*.

11.10 Example OpenCV with Rotations Enabled

Enabling rotations allows the face detection in images, where faces are subject to in plane rotation (roll). The following configuration file can be used to enable detection in such scenarios:

```
# This is a configuration file commentary.
[DETECTOR]
rotations = [-30, 0, 30]
# MANDATORY LAST LINE (DO NOT REMOVE OR WRITE AFTER THIS LINE.)
```

See Illustration 5 for results of running example-opencv with the above configuration file. The face recognition of face attributes likes age and gender is also available in case of in plane rotated faces.

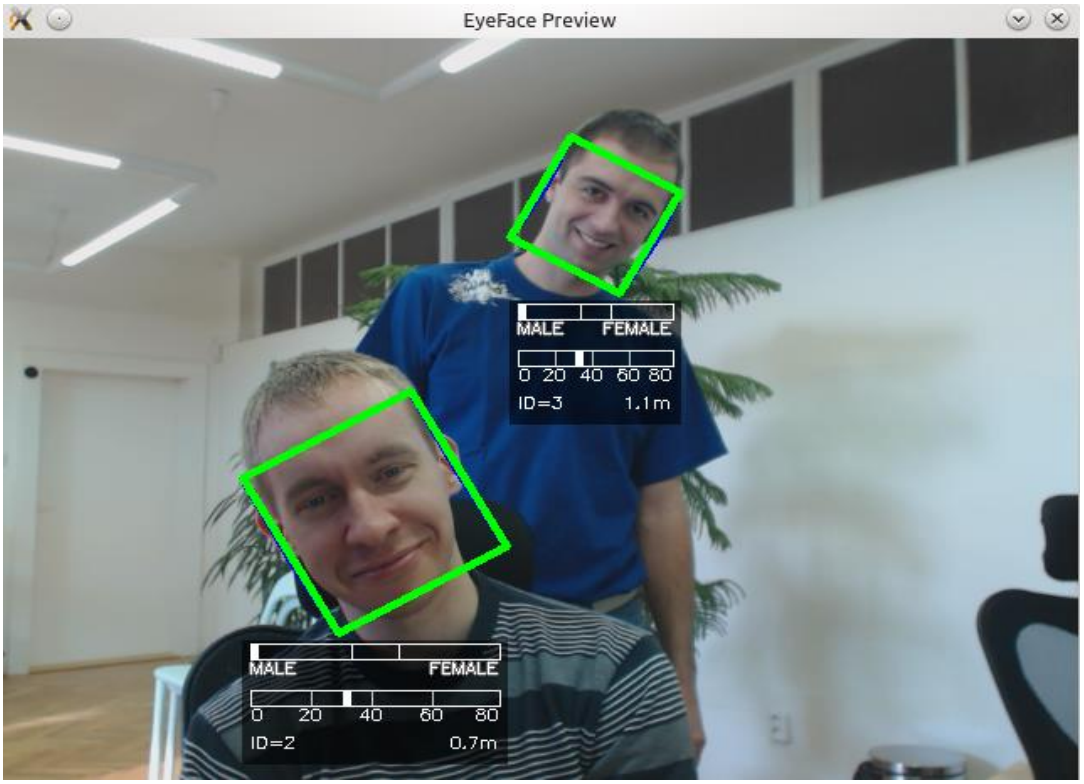


Illustration 5: EyeFace SDK example OpenCV with rotations enabled.

12 Logging Modules

EyeFace SDK contains file logging module and server logging module, which enables the client to get the tracking information not only through the C API, but also as output to file or to HTTP POST. The data that are send or written are the start/stop messages and *EfTrackInfoArray* serialized to JSON. In case of server logging module, there is also a ping signal message, that allows the client to check whether the server is listening to requests.

12.1 Setting up the File Logging

By default, the file logging is disabled in EyeFace SDK. The file logging can be enabled in the configuration file as described in Chapter 8.8.

12.2 Setting up the Server Logging

By default, the file logging is disabled in EyeFace SDK. The server logging can be enabled in the configuration file as described in Chapter 8.8.

12.3 Syntax of Log Messages

The following messages are sent to the logging server by EyeFace SDK. The syntax is a single line without any line break, line breaks were only inserted here for formatting purposes. The order of the arguments after the first ampersand may change without notice. The client therefore must process both the variable name and variable value of each item.

12.3.1 Camera Start Time

The camera start time message is send/written when *efInitEyeFace* is called. The time is in UNIX/POSIX format, i.e. the number of seconds elapsed since January 1, 1970, with millisecond precision.

Syntax

```
{ "action_id":0, "license_id":[LICENSE_ID], "device_id":[DEVICE_ID], "time": [START_TIME] }
```

- LICENSE_ID – license key id (feature 5 or 6).
- DEVICE_ID – unique identifier that can be set in configuration file
- START_TIME – camera start time in UNIX/POSIX format

Example

```
{ "action_id":0, "license_id":0, "device_id":0, "time":1491909193.379 }
```

12.3.2 Camera Stop Time

The camera stop time message is send/written when *efFreeEyeFace* is called. The time is in UNIX/POSIX format.

Syntax

```
{ "action_id":2, "license_id":[LICENSE_ID], "device_id":[DEVICE_ID], "time": [STOP_TIME] }
```

- LICENSE_ID – license key id (feature 5 or 6).
- DEVICE_ID – unique identifier that can be set in configuration file
- STOP_TIME – camera stop time in UNIX/POSIX format

Example

```
{ "action_id":2, "license_id":0, "device_id":0, "time": 1491909708.498 }
```

12.3.3 Track Info Array

In case of file logging, only tracks with status EF_TRACKSTATUS_FINISHED are logged into file. In case of logging to server, by default only tracks with status EF_TRACKSTATUS_FINISHED are logged, this can be changed in the configuration file to log all tracks. Keep in mind that the network traffic when sending each track might reach several gigabytes per day. The C structure is directly converted into JSON string for output.

Syntax

```
{ "action_id":1, "license_id":[LICENSE_ID], "device_id":[DEVICE_ID], "time": [STOP_TIME],
"track_info_array": [TRACK_INFO_ARRAY]}
```

- LICENSE_ID – license key id (feature 5 or 6).
- DEVICE_ID – unique identifier that can be set in configuration file
- STOP_TIME – camera stop time in UNIX/POSIX format
- TRACK_INFO_ARRAY – *EfTrackInfoArray* in JSON format

Example

```
{ "action_id":1, "license_id":0, "device_id":0, "time":1491909513.264, "track_info_array":{
"num_tracks":1, "track_info":[ { "status":1, "track_id":42, "person_id":1, "image_position":{
"top_left_col":394, "top_left_row":187, "top_right_col":507, "top_right_row":187,
"bot_left_col":394, "bot_left_row":299, "bot_right_col":507, "bot_right_row":299 },
"world_position":[ -0.164, 0.802 ], "angles":[ 0.000, 0.000, -15.191 ], "landmarks":{
"recognized":false, "points":{ "length":0, "rows":null, "cols":null }, "angles":[ 0.000, 0.000,
0.000 ] }, "face_attributes":{ "age":{ "recognized":true, "value":29.200, "response":0.000 },
"gender":{ "recognized":true, "value":-1, "response":-1.000 }, "emotion":{ "recognized":true,
"value":-1, "response":0.001 }, "ancestry":{ "recognized":true, "value":1, "response":0.995 } },
"energy":-0.030, "start_time":318.610, "current_time":319.875, "total_time":1.265,
"attention_time":0.750, "attention_now":false, "detection_index":-1 } ] } }
```

12.3.4 Ping Signal

This message is used to inform both the client and the server that the connection is online.

Syntax

```
{ "action_id":3, "license_id":[LICENSE_ID], "device_id":[DEVICE_ID], "time": [CURRENT_TIME] }
```

- LICENSE_ID – license key id (feature 5 or 6).
- DEVICE_ID – unique identifier that can be set in configuration file
- CURRENT_TIME – camera current time in UNIX/POSIX format

12.3.5 Requested Server Response

Clients logging server can be connected to EyeFace SDK using log to server feature. The EyeFace SDK logging interface expects the following response string to all its POST requests:

```
"HTTP/1.1 200 OK\r\nContent-Length: 2\r\nContent-Type=text/plain\r\n\r\nOK"
```

Without this response from the server, the logging cannot proceed to send the next message. The response sending is shown in the logserver example.

13 Known Issues

13.1 Maintaining Relative Paths to the EyeFace SDK

Relative paths of custom executables to EyeFace SDK must be maintained by the client, often causing confusion. On Linux/Max OS X, the relative paths from the executed binary to the **[EyeFaceSDK]/eyefacesdk** directory must be set up when calling efInitEyeFace() function. On Windows, the same applies and all the .dll files from **[EyeFaceSDK]/eyefacesdk/lib** must be either copied to the executed binary location or added to PATH environment variable. The client is strongly advised to experiment with the examples available at the EyeFace SDK package. The files mentioned must be distributed with every client's software.

13.2 Microsoft Windows - Windows Live

There is a known issue with shared library WLIDNSP.DLL located in Windows Live installation. If this library is in the target machine's PATH environment variable, it will slow down or even crash the explicit linking to the EyeFace.dll. The WLIDNSP.DLL is located under

```
C:\Program Files (x86)\Common Files\microsoft shared\Windows Live
```

To solve the issue, the client should either delete the .dll (it will not break Windows Live installation), or uninstall all the Windows Live software, e.g. all the services in Windows Live Essentials and finally check that the above directory was deleted during the process of uninstallation.

13.3 Distributing EyeFace SDK to Customers

The client must copy all the files contained in **[EyeFaceSDK]/eyefacesdk** together with the distribution of the client's software, preserving the original relative paths. The **[EyeFaceSDK]/eyefacesdk** directory contains all the binaries and recognition model files needed to start the EyeFace SDK engine. The customer will also need to install EyeFace SDK licensing daemon and have a software license using the approach described in Chapter 3.

14 Problem Solving

In case that the client encounters any problem while using EyeFace SDK, Eyedea Recognition, Ltd. will provide support at

info@eyedea.cz

To help debug the issue, the client should first run the executable causing the problem with stderr stream redirected to a file. EyeFace SDK outputs its inner status errors to the stderr, which often explain what went wrong. The error stream file should be attached to the issue report email.

To redirect stderr stream to a file, the client should run the executable from the command line with option **"2>err.txt"**.

15 Third Party Software

The EyeFace SDK uses third party software libraries, in accordance with their licenses. The licenses can be found under **[EyeFaceSDK]/documentation/3rdparty-licenses**. Here is a complete list of all libraries used, in alphabetical order.

- Boost
- Caffe
- Curl
- IniParser
- Ippicv
- LibJasper
- LibJPEG
- LibPNG
- LibTIFF
- OpenBLAS
- OpenCV
- OpenSSL
- Protobuf
- pthread
- ZLib

The following statements are published to fulfill the license terms of the respective libraries.

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org/)."

"This software is based in part on the work of the Independent JPEG Group".