Copyright ©2019, Eyedea Recognition s.r.o.

All rights reserved

Eyedea Recognition s.r.o. is not responsible for any damages or losses caused by incorrect or inaccurate results or unauthorized use of the software Eyedentify.

Gemalto, the Gemalto logo, are trademarks and service marks of Gemalto and are registered in certain countries. Safenet, Sentinel, Sentinel Local License Manager and Sentinel Hardware Key are registered trademarks of Safenet, Inc.


Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Intel is a trademark of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

NVIDIA, the NVIDIA logo, GeForce®, GeForce® GTX, CUDA®, the CUDA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and/or other countries.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

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. What's New .................................................................................................................... 3  
2. Product Description ....................................................................................................... 4  
   2.1. Technical Details .................................................................................................... 4  
   2.2. System Workflow ................................................................................................... 5  
3. Distribution Contents .................................................................................................... 6  
4. Hardware Requirements ............................................................................................... 7  
   4.1. Minimal Requirements ............................................................................................ 7  
   4.2. Recommended Requirements .................................................................................. 7  
   4.3. Recommended Requirements (GPU version) .......................................................... 7  
   4.4. Supported Operating Systems ............................................................................... 7  
5. Installation Guide .......................................................................................................... 8  
   5.1. Pre-installation ....................................................................................................... 8  
   5.2. Installation ............................................................................................................ 8  
   5.3. Verification of Installation ..................................................................................... 9  
   5.4. Installation Failures ............................................................................................... 9  
   5.5. Managing Licenses ............................................................................................... 9  
   5.6. License Error Codes ............................................................................................. 10  
6. ERIImage Application Interface .................................................................................... 11  
   6.1. Image format ......................................................................................................... 11  
   6.2. Application Interface ........................................................................................... 13  
7. SDK Application Interface ........................................................................................... 22  
   7.1. Enumerators ......................................................................................................... 22  
   7.2. Structures ........................................................................................................... 22  
   7.3. Functions ............................................................................................................ 27  
8. Examples ....................................................................................................................... 38  
   8.1. Eyedentify MMR SDK Example ......................................................................... 38  
9. Eyedentify SDK Licensing ............................................................................................ 42  
   9.1. License Key Types ............................................................................................... 42  
   9.2. Licenses Overview .............................................................................................. 42  
   9.3. License Management ......................................................................................... 43  
   9.4. License Update .................................................................................................. 45  
10. Third Party Software .................................................................................................... 47
1 What’s New

Eyedentify MMR 2.6.0  Release date: 2016/06/27
• Completely new binary models for vehicle classification with actual data for 2019/Q2.
• Newly added rear vehicle model classification.
• The GPU version build with CUDA® Toolkit 10.0.

Eyedentify MMR 2.5.0-BETA  Release date: 2018/12/12
• Completely new binary models for vehicle classification with actual data for 2018/Q4.
• Added support for batch descriptor computation.
• Added support for GPU descriptor computation.
• The GPU version build with CUDA® Toolkit 9.1.
• Added gpu-devices binary which can be used for listing installed CUDA® GPUs.
• Java Wrapper built with the Java version 1.8.0_191.

Eyedentify MMR 2.4.1  Release date: 2018/06/27
• Completely new binary models for vehicle classification with actual data for 2018/Q1.
• Added BGRA image color model support.
• Sentinel Run-time updated to the newest version 7.80.
• Java Wrapper built with the Java version 1.8.0_171.
• Bugfixes in SDK Wrappers.

Eyedentify MMR 2.3.1  Release date: 2017/11/13
• Completely new binary models for vehicle classification with actual data for 2017/Q3.
• Binary models for vehicle classification now support view recognition feature. View type (frontal, rear) is now one of the vehicle classification outputs. Separate binary models for frontal and rear view are not the part of the distribution anymore, one binary model for both views can be used.
• Eyedentify shared library now supports explicit linking only. Example codes for explicit linking are the part of the release package. Static linking was removed.
• Added new structure ERImage for image manipulation which internally uses OpenCV library. ERImage API is defined in the er_image.h and er_explink.h header files. EdfImage structure was removed.
• Header file edf_utils.h removed. Contained functions for EdfCropParams structure manipulation (edfCropParamsAllocate, edfCropParamsWrap and edfCropParamsFree) moved to the edf.h header. Remaining functions from edf_utils.h were removed from the public API or replaced by the ERImage API functions.
• EdfCropImageConfig structure updated with antialiasing settings.
• Structure EdfClassifyConfig for classification configuration added. Function edfClassify can be configured using the EdfClassifyConfig instance on the input.
• New version of licensing software Sentinel LDK 7.6 used.
• Example image data moved to the package root folder data.
• Java Wrapper built with the Java version 1.8.0_151.

Eyedentify MMR 2.1.0  Release date: 2017/02/08
2 Product Description

Eyedentify MMR SDK is a cross-platform software library developed to provide vehicle recognition functionality. It defines an interface between the user’s software and our state of the art recognition solution. Eyedentify MMR SDK allows the user to recognize a vehicle located in digital image. The recognition output contains the information about the vehicle’s view, category, make, model and vehicle’s color.

Vehicle view can be frontal or rear. Vehicles are divided into 6 categories – bus, car, heavy truck, light truck, motorbike and van. The manufacturers of the vehicles are defined in the make output parameter – e.g. Toyota, Volkswagen, etc. The model parameter then distinguishes the bodywork of vehicles created by specific manufacturer – Avensis, Passat, etc.

2.1 Technical Details

Eyedentify MMR SDK consists of two libraries – base Eyedentify library and the recognition module. Both are cross-platform x86/x64 libraries with C interface. The base Eyedentify library is the only entry point, the user never uses the recognition module directly.

The recognition module is loaded and configured using the Eyedentify library. The configuration parameters are loaded from one the binary models, which are contained in the distribution.

The Eyedentify library provides following APIs:

- C native API
- Java JNI API
- C# API (Windows only)

Officially supported operating systems and platforms:

- Windows 7, 8, 8.1 and 10
  - 32 and 64 bit (Visual Studio 2015)
- Ubuntu 14.04 and higher
  - 32 and 64 bit
2.2 System Workflow

The workflow of the MMR system consists of image acquisition, vehicle license plate detection, input image cropping, vehicle descriptor computation and vehicle classification. The image acquisition and the vehicle license plate detection are not the part of this SDK and must be solved separately.

The process starts with the input image cropping with respect to the license plate detection. The image crop is done using the SDK, it crops and transforms the image in the way that only the vehicle of interest is contained in the cropped image. The crop is the input of the machine learning algorithm, which is contained in the SDK’s recognition module. The output is the descriptor – real number vector describing the input vehicle in the condensed form. The descriptor is then classified, where the output is the classification result – human readable output of the vehicle recognition.

1) Image acquisition
*Not the part of the SDK*

2) License plate detection
*Not the part of the SDK*

3) Image cropping
*Input image cropping around the vehicle of interest.*

4) Descriptor computation
*Vehicle descriptor computation from the vehicle image crop.*

5) Classification
*Vehicle descriptor classification and getting the MMR output.*

<table>
<thead>
<tr>
<th>View:</th>
<th>frontal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>CAR</td>
</tr>
<tr>
<td>Make:</td>
<td>VW</td>
</tr>
<tr>
<td>Model:</td>
<td>Golf</td>
</tr>
<tr>
<td>Color:</td>
<td>WHITE</td>
</tr>
</tbody>
</table>
3 Distribution Contents

The following list is an excerpt from the Eyedentify MMR SDK directory structure, highlighting the most important directories and files contained in the software distribution. A brief description of the items is given.

- [Eyedentify MMR SDK]/ ................................. distribution main folder
  - documentation .......................................................... documentation folder
    - 3rdparty-licenses ........................................ licenses to used 3rd party libraries
    - Eyedentify MMR SDK - Developer’s Guide.pdf ........................ SDK Developer’s Guide
    - Eyedentify MMR SDK - Technical sheet.pdf.......................... SDK Technical Sheet
  - examples ............................................................... examples source files
    - edf-explink ......................................................... explicit linking support sources
    - example-edf-explink .............................................. explicit linking example
    - example-vcl-API ................................................ basic MMR API example
    - gpu-devices ...................................................... utility for GPU devices listing
  - hasp ................................................................. software protection binaries
  - sdk ................................................................. Eyedentify MMR SDK mandatory data
    - include .......................................................... header files
    - lib ............................................................... Eyedentify library
    - modules .......................................................... recognition modules folder
      - edfdeep-vcl ...................................................... MMR recognition module
        - lib .......................................................... recognition module shared library
        - model ....................................................... recognition module binary models
  - wrappers .......................................................... API wrappers
    - java .............................................................. Java API wrapper
    - csharp ............................................................ C# API wrapper
  - LICENSE.txt ..................................................... SDK license
  - README.txt ..................................................... SDK Readme file
4 Hardware Requirements

4.1 Minimal Requirements

Processor: 1.0 GHz, single core, x86 platform, embedded (i.e. Intel Atom)
RAM: 2 GB
Hard disk: 1 GB free space

4.2 Recommended Requirements

Processor: 2.0 GHz, dual core, x86 platform (i.e. Intel i5)
RAM: 4 GB
Hard disk: 2 GB free space

4.3 Recommended Requirements (GPU version)

Processor: 3.0 GHz, quad core, x86 platform (i.e. Intel i7)
RAM: 8 GB
Hard disk: 2 GB free space
GPU: NVIDIA® GeForce® GTX 1080 and higher
Compute capability 3.0, 3.5, 5.0, 5.2, 6.0, 6.1, 7.0, 7.5.
Support of CUDA® 10.0.

4.4 Supported Operating Systems

4.4.1 Windows

- Microsoft Windows 7/8/8.1/10
  - Win32 and x64 platform
  - GPU version supports x64 platform only

4.4.2 Linux

- Ubuntu 14.04 and higher
  - i686 and x86_64 platform
  - GPU version supports x86_64 platform only
5 Installation Guide

Installation of software licensing daemon is the first step to start using Eyedentify 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 Eyedentify SDK to run, and to manage licenses using a web browser.

5.1 Pre-installation

Prior to the installation of the licensing software, all Sentinel Hardware Keys should be removed from the target computer based on the recommendation from 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).

5.2 Installation

5.2.1 Windows

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

- Start the command line cmd with Administrator privileges.
- Navigate to [EyedentifySDK]/hasp/ directory.
- Execute "dunst.bat" to uninstall any previous versions of the Sentinel License Manager.
- Execute "dinst.bat" to install the Sentinel License Manager.

5.2.2 Linux

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

- Start the command line and navigate to [EyedentifySDK]/hasp/ directory.
- On 64-bit Linux distributions, install the 32-bit compatibility binaries.
  - On Ubuntu 14.04 and higher:
    - Execute "sudo apt-get install libc6:i386".
- Execute "sudo ./dunst" to uninstall any previous versions of the Sentinel License Manager.
- Execute "sudo ./dinst" to install the Sentinel License Manager.

(Without compatibility binaries error “No such file or directory.” might appear.)
5.3 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 Eyedentify 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 relicense the software to another machine. The web page lists all the available license keys. Under the "Products" link in the left pane all available products are listed.

Illustration 1: Sentinel License Manager screenshot.

5.4 Installation Failures

On Windows, antivirus application might break the installation of Sentinel License Manager. If the installation failed, the client should disable the antivirus 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 antivirus. Port number 1947 must be also added to the exception list in the Windows firewall and also in the antivirus in case it uses its own firewall.

5.5 Managing Licenses

It is of the most importance that the client understands the licensing schemes used in Gemalto 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 the Chapter 9.
5.6 License Error Codes

The error codes are outputted to error stream of the application (typically `stderr`) using Eyedentify 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. (No license for the Eyedentify 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 Eyedentify 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.
6 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.

6.1 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 four color models are supported in the ERImage image structure. First is BGR color model, second is Gray color model, the third is YCbCr 4:2:0 color model and the fourth is BGRA 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.

6.1.1 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 \( \times \) height on coordinates \((x, y)\). Coordinates \(x, y\) and data array indices are 0-based.

\[
\begin{align*}
B(x,y) &= data(3 \times (width \times y + x) + 0) & \text{B component on } (x, y) \text{ coordinates} \\
G(x,y) &= data(3 \times (width \times y + x) + 1) & \text{G component on } (x, y) \text{ coordinates} \\
R(x,y) &= data(3 \times (width \times y + x) + 2) & \text{R component on } (x, y) \text{ coordinates}
\end{align*}
\]

6.1.2 Gray

One channel model Gray is used for storing grayscale image, which is composed from luminance values \((Y - \text{luminance})\). 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 \( \times \) height on coordinates \((x, y)\). Coordinates \(x, y\) and data array indices are 0-based.

\[
Y(x,y) = data(width \times y + x) & \text{ Y component on } (x, y) \text{ coordinates}
\]
6.1.3 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 \times height on coordinates (x, y). Coordinates x, y and data array indices are 0-based. All divisions in the formulas are integer divisions.

\[
Y(x, y) = data(width \times y + x) \\
|Y| = width \times height \\
Cb(x, y) = data\left(|Y| + \frac{y}{2} \times \frac{width}{2} + \frac{x}{2}\right) \\
|Cb| = |Cr| = \frac{width \times height}{4} \\
Cr(x, y) = data\left(|Y| + |Cb| + \frac{y}{2} \times \frac{width}{2} + \frac{x}{2}\right)
\]

6.1.4 BGRA

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

\[
B(x, y) = data(3 \times (width \times y + x) + 0) \\
G(x, y) = data(3 \times (width \times y + x) + 1) \\
R(x, y) = data(3 \times (width \times y + x) + 2) \\
A(x, y) = data(4 \times (width \times y + x) + 3)
\]
6.2 Application Interface

6.2.1 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.
- **ER_IMAGE_COLORMODEL_BGRA = 4**
  - Four channel BGRA color model. Four values per pixel stored 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.

6.2.2 Structures

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

ERImage

```c
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)
6.2.3 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**:

```c
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:**

```c
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:**

```c
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 *erImageAllocateBlank()* 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:**

```c
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:**

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

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:

```c
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:

```c
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:
```cpp
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:**

```cpp
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 `erImageGetDataTypeSize()` 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:
```cpp
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:**

```cpp
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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
```
7 SDK Application Interface

This chapter describes all the parts of the SDK’s public application interface for C/C++ programming language including defined Enumerators, Structures and all available Functions. It gives developer detailed overview of the SDK and helps to orientate during SDK integration.

7.1 Enumerators

This part contains all the information about enumerators used in the SDK’s public API. EdfComputationMode is used during the SDK initialization to specify the computation type.

7.1.1 EdfComputationMode

EdfComputationMode is used during the SDK instance initialization to specify the mode in which the SDK instance computation will operate.

- **EDF_COMPUTATIONMODE_CPU = 0**
  - CPU is exclusively used for all computations inside the SDK instance.
- **EDF_COMPUTATIONMODE_GPU = 1**
  - The most expensive computations are processed on the computer’s GPU (graphics processing unit), which gives significant processing speed-up.
  
  *This functionality is available in the GPU version of the SDK only.*
  
  NVIDIA® CUDA® capable graphics card is required.

7.2 Structures

Document section Structures covers all the information about structures used in the SDK’s public application interface.

EdfInitConfig is used during the SDK initialization, EdfDescriptor stores the unique description of the specified object, EdfPoints, EdfValues and EdfCropParams are used during input image preparation, EdfClassifyResultValue and EdfClassifyResult store the results of the classification process, EdfCropImageConfig specifies the configuration of the input image cropping, EdfComputeDescConfig specifies the configuration of the descriptor computation and EdfClassifyConfig specifies the configuration of the descriptor classification.

7.2.1 EdfInitConfig

```c
typedef struct {
    const char* module_path;
    const char* model_file;
    EdfComputationMode computation_mode;
    int gpu_device_id;
} EdfInitConfig;
```

EdfInitConfig represents the configuration parameters set used during SDK module initialization.
The structure contains following fields:

- **module_path**
  NULL terminated string with the path to the module.
  example: “C:/folder/Eyedentify-MMR-2.6.0/sdk/modules/edfdeep-mmr/”

- **model_file**
  NULL terminated string with model filename.
  example: “CNN_MMR_VCMM_BGR_PRECISE_2019Q2.dat”

- **computation_mode**
  Selected computation mode from EdfComputationMode enumerator.

- **gpu_device_id**
  Zero based integer representing GPU device identifier.
  *(used only when computation_mode == EDF_COMPUTATIONMODE_GPU)*

### 7.2.2 EdfDescriptor

```
typedef struct {
  unsigned int version;
  unsigned int size;
  unsigned char* data;
} EdfDescriptor;
```

*EdfDescriptor* contains the data which describes recognized object. The data is called the descriptor, which is generated as the output of the machine learning algorithms. The advantage of this approach is that the descriptor is usually smaller than input image and it is easier and more efficient to compare the descriptors than the input images.

The structure contains following fields:

- **version**
  Version of the binary model used for the descriptor computation.

- **size**
  Size of the descriptor data in bytes.

- **data**
  The descriptor data.

### 7.2.3 EdfPoints

```
typedef struct {
  int length;
  double* rows;
  double* cols;
} EdfPoints;
```

*EdfPoints* contains the 2D points defined by the row (y-axis) and column (x-axis) coordinates.

The structure contains following fields:

- **length**
  Number of the contained 2D points.

- **rows**
  Pointer to the array containing the row (y-axis) coordinates of the 2D points.

- **cols**
  Pointer to the array containing the column (x-axis) coordinates of the 2D points.
7.2.4 **EdfValues**

```c
typedef struct {
    int    length;
    double* values;
} EdfValues;
```

*EdfValues* contains real value array.

The structure contains following fields:

- **length**
  Number of contained real values.

- **values**
  Pointer to the array containing the real values.

7.2.5 **EdfCropParams**

```c
typedef struct {
    EdfPoints points;
    EdfValues values;
} EdfCropParams;
```

*EdfCropParams* contains the parameters used for image cropping with the function `edfCropImage`.

The contained parameters are internally stored in the *EdfPoints* and *EdfValues* structures.

The structure contains following fields:

- **points**
  Instance of the *EdfPoints* structure.

- **values**
  Instance of the *EdfValues* structure.

7.2.6 **EdfClassifyResultValue**

```c
typedef struct {
    char*    task_name;
    unsigned int task_name_length;
    char*    class_name;
    unsigned int class_name_length;
    int      class_id;
    float    score;
} EdfClassifyResultValue;
```

*EdfClassifyResultValue* represents the classification result value. The result value contains the name of the task (*classifier name*) and name of the class with the ID and the highest score.

The structure contains following fields:

- **task_name**
  NULL terminated string containing the name of the classified task.

- **task_name_length**
  Length of the *task_name* string without the terminating NULL character.

- **class_name**
  NULL terminated string containing the name of the resulting class.

- **class_name_length**
  Length of the *class_name* string without the terminating NULL character.

- **class_id**
  Identifier of the resulting class.

- **score**
  Classification score of the resulting class.
7.2.7 EdfClassifyResult

typedef struct {
    unsigned int num_values;
    EdfClassifyResultValue* values;
} EdfClassifyResult;

EdfClassifyResult contains the array with the EdfClassifyResultValue values as the result of the classification using the function edfClassify.

The structure contains following fields:

- **num_values**
  Number of the contained EdfClassifyResultValue values.

- **values**
  Array containing the EdfClassifyResultValue values.

7.2.8 EdfCropImageConfig

typedef struct {
    int full_crop;
    int color_normalization;
    int use_antialiasing;
    unsigned int antialiasing_kernel_size;
    float antialiasing_sigma;
} EdfCropImageConfig;

EdfCropImageConfig is used for image cropping configuration in the function edfCropImage.

Use this configuration only if you know what you are doing.

The structure contains following fields:

- **full_crop**
  Specifies whether the image is returned with the boundary. *(Default: 0)*
  Set to 1 to create the full crop (with the border).
  Set to 0 to create default crop.
  Set to -1 to create standard crop.

- **color_normalization**
  Specifies whether the color normalization is applied. *(Default: 1)*
  Set to 1 to use color normalization.
  Set to 0 to use color normalization default setting.
  Set to -1 to not use color normalization.

- **use_antialiasing**
  Specifies whether the antialiasing procedure is applied during the image transformation.
  Set to 1 to use antialiasing during image transformation.
  Set to 0 to use antialiasing default setting.
  Set to -1 to not use antialiasing during image transformation.
  *(Default antialiasing setting is loaded from the binary model and is specified for each task.)*

**IMPORTANT:** The antialiasing option can significantly improve recognition results, especially in the cases when the scale change during the cropping is high and aliasing occurs in the cropped image.

Cropping time can be significantly higher with antialiasing option enabled. It is caused by image filtering which is computationally expensive. The computation time depends on the scale change during the cropping and on the size of the image area to be cropped.
- **antialiasing_kernel_size**
  The size of the convolution kernel used during antialiasing.
  Set to 0 to use the default convolution kernel size (*computed from transformation scale*).
  Used only in combination with use_antialiasing == 1. *(Default: 0)*

- **antialiasing_sigma**
  The sigma parameter of the Gaussian distribution in the antialiasing convolution kernel.
  Set to 0.0f to use the default sigma size (*computed from kernel size*).
  Used only in combination with use_antialiasing == 1. *(Default: 0.0f)*

### 7.2.9 EdfComputeDescConfig

```c
typedef struct {
    unsigned int batch_size;
} EdfComputeDescConfig;
```

`EdfComputeDescConfig` configures the descriptor computation in the function `edfComputeDesc`.

The structure contains following fields:

- **batch_size**
  The size of the descriptors batch to compute.
  Set to 0 to disable batch processing.
  Set to 1-N to set the size of the batch (*value 1 has the same effect as 0*).

**IMPORTANT:** The advantage of the batch descriptor computation is the speed of the processing where the batch processing can be faster than sequential on some system configurations (*especially when using GPU computation mode*).

To use the batch descriptor computation in the function `edfComputeDesc()`, the input **image crops must be stored in the memory consecutively** (*like in an array or a vector*). In the same way **must be initialized the memory for storing the computed descriptors**.

**Example:**

```c
std::vector<EdfImage> imageCrops = { imageCrop1, imageCrop2, imageCrop3, imageCrop4 };
std::vector<EdfDescriptor> descriptors;
descriptors.resize(imageCrops.size());
EdfComputeDescConfig compDescConfig;
compDescConfig.batch_size = (unsigned int)imageCrops.size();
edfAPI.edfComputeDesc(&imageCrops[0], module_state, &descriptors[0], &compDescConfig);
```

### 7.2.10 EdfClassifyConfig

```c
typedef struct {
    int use_dependency_rules;
} EdfClassifyConfig;
```

`EdfClassifyConfig` is used for descriptor classification configuration in the function `edfClassify`.

Use this configuration only if you know what you are doing.

The structure contains following field:

- **use_dependency_rules**
  Specifies whether the dependency rules between the classifiers are applied. *(Default: 0)*
  When the dependency rules are not applied, **task_name** field in returned classification output structure `EdfClassifyResultValue` contains string with `_NODEP` suffix.
  Set to 0 to get classification results with dependency rules applied.
  Set to 1 to get classification results both with and without dependency rules applied.
  Set to -1 to get classification results without dependency rules applied.
7.3 Functions

This chapter contains the definition of the Eyedentify library functions which are present in the public API. The chapter is divided into five parts. First describes the main API functions, the other refers to the functions designed for the manipulation with the API public data structure `EdfCropParams`.

**IMPORTANT:** The Eyedentify library is not thread safe. Do not call the library instance from multiple threads but initialize an instance of the library for each thread separately.

### 7.3.1 Main API

This part defines the API functions which are designed to control the main part of the Eyedentify recognition library. The functions are: `edfInitEyedentify`, `edfFreeEyedentify`, `edfCropImage`, `edfComputeDesc`, `edfCompareDescs`, `edfClassify`, `edfModelVersion`, `edfAllocDesc`, `edfFreeDesc`, `edfFreeCropImage`, `edfFreeClassifyResult`. These functions are defined in the `edf.h` file.

**edfInitEyedentify**

Initializes the Eyedentify module using supplied parameters.

**Specification:**

```c
int edfInitEyedentify(const EdfInitConfig* init_config, void** module_state)
```

**Inputs:**

- `init_config`  
  `EdfInitConfig` structure containing the parameters needed for initialization.

**Outputs:**

- `module_state`  
  Pointer to the successfully initialized Eyedentify module instance.

**Returns:**

- `0` – Initialization was successful
- `other` – Error during initialization occurred.

**Error codes:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Module library was not found.</td>
</tr>
<tr>
<td>-2</td>
<td>Module library loading error.</td>
</tr>
<tr>
<td>-3, ..., -12</td>
<td>Module functions linking error.</td>
</tr>
<tr>
<td>-101</td>
<td>Binary model could not be opened.</td>
</tr>
<tr>
<td>-102</td>
<td>Module path not specified.</td>
</tr>
<tr>
<td>-103</td>
<td>Empty module path.</td>
</tr>
<tr>
<td>-104</td>
<td>Model name not specified.</td>
</tr>
<tr>
<td>-105</td>
<td>Empty model name.</td>
</tr>
<tr>
<td>-106</td>
<td>Binary model is not supported.</td>
</tr>
<tr>
<td>-107</td>
<td>Failed to allocate license subsystem.</td>
</tr>
<tr>
<td>-108</td>
<td>Binary model license is not supported.</td>
</tr>
<tr>
<td>-109</td>
<td>Failed to initialize license subsystem.</td>
</tr>
<tr>
<td>-110</td>
<td>License login failed.</td>
</tr>
<tr>
<td>-111</td>
<td>Login to perpetual count feature failed.</td>
</tr>
<tr>
<td>-112</td>
<td>Binary model contains invalid optional config.</td>
</tr>
<tr>
<td>-113</td>
<td>Binary model contains corrupted structure file.</td>
</tr>
<tr>
<td>-114</td>
<td>Failed to initialize loaded structure.</td>
</tr>
<tr>
<td>-115</td>
<td>Binary model contains corrupted metric.</td>
</tr>
<tr>
<td>-116</td>
<td>Binary model contains corrupted classifiers.</td>
</tr>
<tr>
<td>-117</td>
<td>Binary model contains invalid input dimensions.</td>
</tr>
<tr>
<td>-118</td>
<td>Mixing CPU/GPU modes in one process.</td>
</tr>
<tr>
<td>-67856865</td>
<td>GPU computation mode not supported.</td>
</tr>
<tr>
<td>-67856866</td>
<td>Error setting specified GPU device.</td>
</tr>
<tr>
<td>-67856867</td>
<td>Specified GPU device is busy.</td>
</tr>
<tr>
<td>-67856868</td>
<td>Error getting the count of GPU devices.</td>
</tr>
<tr>
<td>-67856869</td>
<td>No GPU device is installed on the system.</td>
</tr>
<tr>
<td>-67856870</td>
<td>Specified GPU device does not exist.</td>
</tr>
</tbody>
</table>

**IMPORTANT:** It is not possible to mix CPU and GPU computation mode in one process. Only one computation mode can be initialized within the process at a time.
SDK Application Interface

Description:
The function `edfInitEyedentify()` is used for initialization of the Eyedentify module. The initialization is required to be able to use the library for recognition tasks. The input of the function call is the structure `EdfInitConfig` and the output is the error code and the pointer to the initialized module.

**IMPORTANT:** The initialization of Eyedentify module is protected in the way it cannot be initialized parallely. In case of concurrent Eyedentify module initialization the instances are initialized serially using the internal mutexing.

**GPU version only:** To initialize the GPU version of the library, the ID of the GPU hardware must be supplied. The ID of the GPU is zero-based number *(similar to the numbering of the installed CUDA® devices used by NVIDIA software)*. To get the list of the installed CUDA® devices on your system, command line utility [EyedentifySDK]/examples/gpu-devices/gpu-devices can be used. *If only one CUDA® GPU is installed ID 0 is used for initialization.*

Example:

```c
void* module_state = NULL;
EdfInitConfig initConfig;
initConfig.module_path          = EDF_MODULE_PATH;
initConfig.model_file           = MODEL_NAME;
initConfig.computation_mode     = EDF_COMPUTATIONMODE_CPU;
initConfig.gpu_device_id        = 0;
int initResult = edfInitEyedentify(&initConfig, &module_state);
if (initResult != 0) {
    // Handle errors
}
```

**edfFreeEyedentify**

Frees initialized Eyedentify module.

**Specification:**

```c
void edfFreeEyedentify(void** module_state)
```

**Inputs:**

- **module_state**
  Pointer to the initialized Eyedentify module instance.

**Description:**

The function `edfFreeEyedentify()` is used for freeing the Eyedentify module. When the module is not needed anymore, for example at the end of the program, all underlying structures must be deallocated. The input of the function call is the pointer `module_state`, which was created using `edfInitEyedentify` function during module initialization.

**IMPORTANT:** Always free the module when it is not needed anymore otherwise your program will have memory leaks.

Example:

```c
void* module_state = NULL;
edfInitEyedentify(&initConfig, &module_state);
// ...
edfFreeEyedentify(&module_state);
```
**edfCropImage**

Prepares/crops the input image for processing.

**Specification:**

```c
int edfCropImage(const ERImage* image_in, EdfCropParams* params, void* module_state, ERImage* cropped_image, EdfCropImageConfig* config)
```

**Inputs:**

- **image_in**
  Input image stored in the *ERImage* structure.
- **params**
  Pointer to the cropping parameters stored in the *EdfCropParams* structure.
- **module_state**
  Pointer to the initialized Eyedentify module instance.
- **config**
  Cropping configuration parameters stored in the *EdfCropImageConfig* structure. 
  *For default configuration set to NULL.*

**Outputs:**

- **cropped_image**
  Pointer to the successfully cropped and transformed image in the *ERImage*.

**Returns:**

- **0** – Cropping was successful.
- **other** – Error during cropping occurred.
  - **-1** – Invalid module pointer, input image, cropping parameters or *cropped_image* pointer.
  - **-2** – Unsupported *image_in* color model or data type.
  - **-3** – Invalid number of channels of the *image_in*.
  - **-4** – Wrong number of points or values in the *params*.

**Description:**

The function *edfCropImage()* crops and transforms the input image according to the input cropping parameters, Eyedentify module and binary model requirements. It is used before the function *edfComputeDesc* to prepare the input image for the descriptor computation.

The input of the function call is the pointer to the input image structure *ERImage*, void pointer *module_state*, cropping parameters defined using *EdfCropParams* structure and the cropping configuration in the *EdfCropImageConfig* structure. Specification of the configuration is optional, for default configuration use NULL pointer. The output is the error code and the pointer to the *ERImage* structure containing cropped image.

Cropping parameters defined in the *EdfCropParams* structure are dependent on the Eyedentify module and recognition task. They allow to define 2D points and double values. For more detailed information see the *Examples* chapter and the example source codes included in the SDK package.

**Example:**

```c
EdfCropParams cropParams;
allocateEdfCropParams(1, 1, &cropParams);
cropParams.points.cols[0] = 123.4;
cropParams.points.rows[0] = 56.7;
cropParams.values.values[0] = 8.9;
ERImage cropImage;
int cropResult = edfCropImage(&image, &cropParams, module_state, &cropImage, NULL);
if (cropResult != 0) {
    // Handle errors
}
```
**edfComputeDesc**

Computes descriptor from input image cropped by *edfCropImage* function.

**Specification:**

```c
int edfComputeDesc(const ERImage* img, const void* module_state,
                    EdfDescriptor* descriptor, EdfComputeDescConfig* config)
```

**Inputs:**

- **img**
  Input image cropped by *edfCropImage* function.
- **module_state**
  Pointer to the initialized Eyedentify module instance.
- **config**
  Descriptor computation configuration in *EdfComputeDescConfig* structure. 
  *For default configuration set to NULL.*

**Outputs:**

- **descriptor**
  Descriptor raw data and metadata saved in the *EdfDescriptor* structure.

**Returns:**

- **0** – Computation was successful.
- **other** – Error during computation occurred.
  - **-1** – Invalid module, license or wrong input image format
  - **-2, -3, -4, -5** – Internal error
  - **-6** – Mixed computation mode (CPU, GPU) in same process

**Description:**

The function *edfComputeDesc()* is used for machine learning method descriptor computation. The descriptor contains information describing input image in the condensed form, which is designed for efficient classification and matching.

The input of the function call is the pointer to the cropped input image structure *ERImage* created with *edfCropImage*, *void* pointer *module_state*, which was created using *edfInitEyedentify* function during module initialization and the computation configuration in the *EdfComputeDescConfig* structure. Specification of the configuration is optional, for default configuration use NULL pointer. The output is the error code and the pointer to the *EdfDescriptor* structure.

**IMPORTANT:** Descriptor computation is next to the initialization the most time and resource consuming process which can last from tens to hundreds of milliseconds on standard x86 computer, depending on the recognition task complexity.

**Example:**

```c
edfCropImage(&image, &cropParams, module_state, &cropImage, NULL);
// ...
EdfDescriptor descriptor;
int computeDescResult = edfComputeDesc(cropImage, module_state, &descriptor, NULL);
if (computeDescResult != 0) {
    // Handle errors
}
```
**edfCompareDescs**

Compares two descriptors and returns similarity score.

**Specification:**

```c
int edfCompareDescs(const EdfDescriptor* desc_A, const EdfDescriptor* desc_B, void const* module_state, float* score)
```

**Inputs:**

- **desc_A**
  First descriptor for comparison stored in the `EdfDescriptor` structure.

- **desc_B**
  Second descriptor for comparison stored in the `EdfDescriptor` structure.

- **module_state**
  Pointer to the initialized Eyedentify module instance.

**Outputs:**

- **score**
  Pointer to the `float` where similarity score is saved on success.

**Returns:**

- **0**
  – Comparison was successful.

- **other**
  – Empty or different descriptors or internal error during comparison occurred.

**Description:**

The function `edfCompareDescs()` compares two descriptors generated by `edfComputeDesc` with the same Eyedentify module and same binary model. Comparison function is not defined generally but is dependent on selected recognition task. The result of the comparison is the similarity score.

For similarity score the following rule is valid: the higher the score is the more similar two descriptors are. The similarity score minimal and maximal possible values are dependent on the Eyedentify module, binary model and on the recognition task, therefore cannot be defined in general.

The input of the function call are the pointers to the descriptors to compare and the `void` pointer `module_state`. The output is the error code and the pointer to the similarity score stored as `float`.

**IMPORTANT:** The function `edfCompareDescs()` requires the data in the `EdfDescriptor` structure to be stored in the `data` field which points to the aligned memory, because SSE instructions are used for computation speedup. To ensure that the memory is aligned, use for descriptor allocation `edfComputeDesc` or `edfAllocDesc` functions only.

**Example:**

```c
edfComputeDesc(cropImageA, module_state, &descA, NULL);
edfComputeDesc(cropImageB, module_state, &descB, NULL);
// ...
float score = 0.0f;
int compareResult = edfCompareDescs(&descA, &descB, module_state, &score);
if (compareResult != 0) {
    // Handle errors
}
```
edfClassify

Classifies descriptor to the classes defined in the binary model.

Specification:

```c
int edfClassify(const EdfDescriptor* desc, void* module_state,
                 EdfClassifyResult** classify_result, EdfClassifyConfig* config)
```

Inputs:
- `desc`
  Descriptor for classification stored in the `EdfDescriptor` structure.
- `module_state`
  Pointer to the initialized Eyedentify module instance.
- `config`
  Descriptor classification configuration in `EdfClassifyConfig` structure.
  - For default configuration set to `NULL`.

Outputs:
- `classify_result`
  Double pointer to the `EdfClassifyResult`, where classification result is saved on success.

Returns:
- `0` – Classification was successful.
- `other` – Empty or corrupted descriptor or internal error during classification occurred.

Description:
The function `edfClassify()` classifies the descriptor to the classes defined in the binary model for the specific recognition task. The result of the classification is the name and the ID of one or more classes to which the object specified by the descriptor belongs.

The input of the function call is the pointer to the descriptor to classify and the `void` pointer `module_state` and the classification configuration in the `EdfClassifyConfig` structure. Specification of the configuration is optional, for default configuration use `NULL` pointer. The output is the error code and the double pointer to the classification result stored in the `EdfClassifyResult` structure.

**IMPORTANT:** Some modules and binary models does not support classification. In these cases, the result of the classification is `NULL`.

Example:

```c
edfComputeDesc(cropImage, module_state, & descriptor, NULL);
// ...
EdfClassifyResult* classify_result = NULL;
int resultCode = edfClassify(&descriptor, module_state, &classify_result, NULL);
if (resultCode == 0) {
    // Print the results to the console
    for (unsigned int i = 0; i < classify_result->num_values; i++) {
        char* task_name = classify_result->values[i].task_name;
        char* class_name = classify_result->values[i].class_name;
        int class_id = classify_result->values[i].class_id;
        float score = classify_result->values[i].score;
        std::cout << task_name << "\": " << class_name << "(" << class_id << ")" << std::endl;
    }
} else {
    // Handle errors
}
```
**edfModelVersion**

Returns the version of the loaded binary model.

**Specification:**

```c
unsigned int edfModelVersion(const void* module_state)
```

**Inputs:**
- `module_state`
  Pointer to the initialized Eyedentify module instance.

**Returns:**
Function returns the model version on success, otherwise 0 is returned.

**Description:**
The function `edfModelVersion()` is used for getting the version of the loaded binary model. The version is the date of the binary model in the format YYYYMMDD. The input of the function call is the `void` pointer `module_state`. The output is the version number or 0 on an error.

**Example:**
```c
void* module_state = NULL;
edfInitEyedentify(&initConfig, &module_state);
// ...
int modelVersion = edfModelVersion(module_state);
if (modelVersion == 0) {
    // Handle errors
}
```

**edfAllocDesc**

Allocates the `EdfDescriptor` structure.

**Specification:**

```c
void edfAllocDesc(EdfDescriptor* desc, unsigned int size, unsigned int version)
```

**Inputs:**
- `desc`
  Pointer to the initialized `EdfDescriptor` structure instance.
- `size`
  Size of the descriptor data to be allocated in bytes.
- `version`
  Version of the binary model used for descriptor computation (see `edfModelVersion`).

**Description:**
The function `edfAllocDesc()` is used for allocating descriptor structure `EdfDescriptor`. Function is used in the cases when the descriptor data are loaded from the external storage, because functions `edfCompareDescs` and `edfClassify` requires aligned memory for the descriptor. The input of the function call is the pointer to the `EdfDescriptor`, size of the data in bytes and the version number.

**Example:**
```c
char* desc_data; // Data loaded from external storage
EdfDescriptor* descriptor = new EdfDescriptor();
edfAllocDesc(descriptor, desc_size, desc_version);
memcpy(descriptor->data, desc_data, desc_size);
```
**edfFreeDesc**

Frees the descriptor *EdfDescriptor* structure.

**Specification:**

```c
void edfFreeDesc(EdfDescriptor* desc)
```

**Inputs:**

- `desc`
  
  Pointer to the initialized *EdfDescriptor* structure instance.

**Description:**

The function `edfFreeDesc()` is used for freeing the allocated descriptor structure *EdfDescriptor*. This special function is required to correctly free the descriptor’s aligned memory. The input of the function call is the pointer to the allocated *EdfDescriptor*.

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

**Example:**

```c
EdfDescriptor* descriptor = new EdfDescriptor();
edfAllocDesc(descriptor, desc_size, desc_version);
// ...
edfFreeDesc(descriptor);
delete descriptor;
```

**edfFreeCropImage**

Frees the *ERImage* structure from the `edfCropImage` function.

**Specification:**

```c
void edfFreeCropImage(ERImage* cropped_image)
```

**Inputs:**

- `cropped_image`
  
  Pointer to the *ERImage* structure instance created using `edfCropImage` function.

**Description:**

The function `edfFreeCropImage()` is used for freeing the allocated image structure *ERImage* created using function `edfCropImage`. The input of the function is the pointer to the allocated *ERImage*.

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

**Example:**

```c
ERImage cropImage;
edfCropImage(&image, &cropParams, module_state, &cropImage, NULL);
// ...
edfFreeCropImage(&cropImage);
```
**edfFreeClassifyResult**

Frees the *EdfClassifyResult* structure from the *edfClassify* function.

**Specification:**

```c
int edfFreeClassifyResult(EdfClassifyResult** classify_result, void* module_state)
```

**Inputs:**

- *classify_result*
  
  Double pointer to the *EdfClassifyResult* structure instance created using *edfClassify* function.

- *module_state*
  
  Pointer to the initialized Eyedentify module instance.

**Returns:**

- 0 — Freeing was successful.
- *other* — Error during freeing occurred.

**Description:**

The function *edfFreeClassifyResult()* is used for freeing the allocated structure *EdfClassifyResult* created using function *edfClassify*. The input of the function is the double pointer to the allocated *EdfClassifyResult* and the *void* pointer *module_state*.

**IMPORTANT:** The function DOES delete the *EdfClassifyResult* pointer because the pointer is created by the *edfClassify* function.

**Example:**

```c
EdfClassifyResult* classify_result = NULL;
edfClassify(&descriptor, module_state, &classify_result);
// ...
edfFreeClassifyResult(&classify_result, module_state);
```
7.3.2 EdfCropParams

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

- **Allocation**: `edfCropParams`
- **Wrapping**: `edfCropParams`
- **Freeing**: `edfCropParams`

These functions are defined in the `edf_utils.h` file.

**edfCropParamsAllocate**

Allocates `EdfCropParams` structure content.

**Specification:**

```c
void edfCropParamsAllocate(int size_points, int size_values, EdfCropParams* params);
```

**Inputs:**

- **size_points**
  Number of points to allocate.
- **size_values**
  Number of values to allocate.
- **params**
  Pointer to the `EdfCropParams` structure instance.

**Description:**

The function `edfCropParamsAllocate()` is used to allocate the cropping parameters contained in the `EdfCropParams` structure instance. The input is the size of the points and values to allocate and the pointer to the `EdfCropParams` structure instance.

**Example:**

```c
EdfCropParams* params = new EdfCropParams();
edfCropParamsAllocate(1, 1, params);
params->points->rows[0] = 12.3;
params->points->cols[0] = 45.6;
params->values->values[0] = 12.3;
```

**edfCropParamsWrap**

Wraps `EdfCropParams` structure over the data arrays.

**Specification:**

```c
void edfCropParamsWrap(int size_points, double* rows, double* cols,
                       int size_values, double* values_data, EdfCropParams* params);
```

**Inputs:**

- **size_points**
  Number of points to wrap.
- **rows**
  Pointer to the row (y-axis) point data to wrap.
- **cols**
  Pointer to the column (x-axis) point data to wrap.
- **size_values**
  Number of values to wrap.
• **values_data**  
  Pointer to the values data to wrap.
• **params**  
  Pointer to the `EdfCropParams` structure instance.

**Description:**
The function `edfCropParamsWrap()` is used to wrap the `EdfCropParams` structure instance over the supplied data arrays. The input is the size of the points, the pointers to the 2D points data arrays to wrap, the size of the values, the pointers to the values data arrays to wrap and the pointer to the `EdfCropParams` structure instance.

**Example:**
```c
double* rows = {12.3, 45.6};
double* cols = {78.9, 10.0};
double* values_data = {12.3, 45.6};

EdfCropParams* params = new EdfCropParams();
edfCropParamsWrap(2, rows, cols, 2, values_data, params);
// params->points->rows[0] == 12.3
// params->points->cols[1] == 10.0
// params->values->values[0] == 12.3
```

**edfCropParamsFree**

Frees the `EdfCropParams` structure content.

**Specification:**
```c
void edfCropParamsFree(EdfCropParams* params);
```

**Inputs:**
• **params**  
  Pointer to the `EdfCropParams` structure instance.

**Description:**
• The function `edfCropParamsFree()` is used to free the 2D points and values data arrays contained in the `EdfCropParams` structure instance. The input is the pointer to the `EdfCropParams` instance.

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

**Example:**
```c
EdfCropParams* params = new EdfCropParams();
edfCropParamsAllocate(10, 5, params);
// ...
edfCropParamsFree(params);
delete params;
```
8 Examples

This chapter contains the examples description which are contained in the SDK package. The examples are used to demonstrate the functionality of the SDK, the source codes are included in the package and are in detail described in this chapter.

8.1 Eyedentify MMR SDK Example

The package Eyedentify MMR SDK contains the example which is used to demonstrate the basic functionality of the MMR SDK on several input images. The example uses known position of the vehicles in the images to recognize their category, make, model and color using the SDK library. This chapter describes in detail the example together with references to important parts of this document.

The example is in the folder [EyedentifyMMRSDK]/examples/example-vcl-API/. The folder contains all needed source codes and files for successful build. In case of Windows Visual Studio 2015 project is included, in case of Linux Makefile is included.

8.1.1 Initialization

First thing to do is the Eyedentify MMR SDK module initialization using the EdfInitConfig structure and the edfInitEyedentify function. Starting with initialization of the MMR module (example below) and continuing with the initialization of the Color recognition module, which has separate binary model.

```c
const char* EDF_MODULE_PATH = "./";
const char* MODEL_NAME = "CNN_MMR_CMM_FRONT_BGR_2017Q1.dat";
// ...
void* module_state = NULL;
EdfInitConfig initConfig;
initConfig.module_path = EDF_MODULE_PATH;
initConfig.model_file = MODEL_NAME;
initConfig.computation_mode = EDF_COMPUTATIONMODE_CPU;
initConfig.gpu_device_id = 0;
int initResult = edfInitEyedentify(&initConfig, &module_state);
if (initResult != 0) {
    EDF_LOG_ERR("Error during Eyedentify module initialization!");
    return 1;
}
```

The EdfInitConfig structure is filled first with the path to the Eyedentify MMR module, then the name of the binary model to use and finally the computation mode and the ID of the GPU are specified. In case of CPU computation, gpu_device_id parameter is not used (e.g. set to 0).

If the initialization was successful, zero code is returned from the edfInitEyedentify function. For other return codes refer the function reference: edfInitEyedentify.

8.1.2 Input image loading

Before the MMR engine could be used, image data must be loaded and decoded. The example uses the Eyedea Recognition’s custom image structure ERImage to manipulate with the images. The image is loaded to the ERImage structure using the erImageRead function.

```c
// Create the ERImage.
ERImage image;
// Read the input image
int image_read_code = edfAPI.erImageRead(&image, IMAGE_FILENAME);
// Check whether the image was loaded.
if (image_read_code != 0) {
    // Handle errors
}
```
8.1.3 Cropping the input image

The engine requires the input image to be cropped and transformed to get the object to recognize aligned and to have the image in correct resolution and color and data format. For such image transformations, the `edfCropImage` function is used.

```cpp
EdfCropParams cropParams;
allocateEdfCropParams(EDF_VCL_CROP_POINTS, EDF_VCL_CROP_VALUES, &cropParams);
// Set license plate center in the input image.
EDF_LP_CENTER_X(cropParams) = 475.0;
EDF_LP_CENTER_Y(cropParams) = 573.0;
// Set license plate resolution in pixels per meter.
// -> License plate has 134 pixels in the image and
// Czech LP is 0.52 m wide: 134/0.52 = 257.7
EDF_LP_SCALE_PX_PER_M(cropParams) = 257.7;
// Set license plate rotation compensation in degrees.
EDF_LP_ROTATION(cropParams) = 2.0;
```

First the cropping parameters in the `EdfCropParams` must be set. The cropping parameters are allocated with the `edfCropParams` function. MMR task requires the center of the license plate in the image, scale in pixels per meter and rotation compensation to be set. The 2D center of the vehicle’s license plate in the image is set using the macro `EDF_LP_CENTER_X` for the x coordinate and the macro `EDF_LP_CENTER_Y` for the y coordinate. The scale of the vehicle in pixels per meter is obtained by dividing the width of the license plate in pixels by the width of the license plate in meters (for details see the example code above) and set using the macro `EDF_LP_SCALE_PX_PER_M`. The last parameter - rotation compensation is set using the macro `EDF_LP_ROTATION`. The rotation compensation parameter defines the in-plane rotation of the input image with the center of the rotation in the center of the license plate.

With the cropping parameters prepared, the input image can be processed with the function `edfCropImage` which crops and transforms the input image with respect to the cropping parameters defined above.

While only the cropped `ERImage` is required for further processing the original `ERImage` with the input image data can be deleted using the function `erImageFree`. As mentioned in the chapter `Input image loading` the input image is loaded with the OpenCV framework to the `cv::Mat` structure and the image data are deleted with that structure, therefore the `data` array of the `ERImage` must be set to NULL to avoid the OpenCV’s image data deletion with `erImageFree` function.
8.1.4 Descriptor computation

The core of the recognition process and most time-consuming operation is done during the descriptor computation. The descriptor is the condensed representation of the recognized object and it is optimized for efficient comparison.

The descriptor computation is done using the `edfComputeDesc` function. The function requires the cropped input image (from the chapter *Cropping the input image*). The computation takes from tens to hundreds of milliseconds depending on the binary model and hardware used (for more information about the computation times see the document *Eyedentify MMR SDK - Technical sheet*).

The image crop is not needed after the descriptor is computed, it can be deleted. The function `edfFreeCropImage` must be used for image crop deletion, because the crop was generated inside the module, must be there also freed. Together with the image crop data the pointer to the `ERImage` structure is freed.

```c
// Create the EdfDescriptor structure.
EdfDescriptor descriptor;
// Compute the descriptor. Crop image is used as an input, // the output is copied to the EdfDescriptor structure.
int computeDescResultCode = edfComputeDesc(cropImage, module_state, &descriptor, NULL);
// Free the crop image data. The crop is not needed anymore.
edfFreeCropImage(&cropImage);
// Check the descriptor computation result.
if (computeDescResultCode != 0) {
    // Handle error.
}
```

8.1.5 Classification

The classification is the process of getting the corresponding classes from the computed descriptor. Also, it is the process of getting the human readable results of the vehicle recognition.

The only input of the classification is the descriptor combined with the module which was used for the descriptor computation. The process of classification is done using the function `edfClassify` which stores the result to the `EdfClassifyResult` structure.

The `EdfClassifyResult` structure contains the array of `EdfClassifyResultValue` values. The values contain the best result for each class. In case of MMR the classes are *category*, *make*, *model* and *color*. The result contains the name of the task (i.e. “make”), the result class name (i.e. “VW”), the ID of the result class (i.e. 43) and the score of the result (i.e. 0.95041). The availability of the classes is dependent on used model (for more information about the current binary models see the document *Eyedentify MMR SDK - Technical sheet*).
The descriptor can be deleted after the classification is done. The deletion of the descriptor is done using the SDK API function `edfFreeDesc`. When the classification result `EdfClassifyResult` is not needed anymore (for example after the results were saved to the DB or printed out) it must be deleted using the `edfFreeClassifyResult` function. The function goes through the contained result values and deletes them all together with the pointer to the structure.

8.1.6 Cleaning up

At the end, when the work with the Eyedentify MMR SDK instance is finished (for example at the end of the program), it must be deleted together with all underlying structures. To delete it use the API function `edfFreeEyedentify`, which is designed for such purpose.

```c
// Free the module. All module internal structures
// will be deleted and program can be finished.
edfFreeEyedentify(&module_state);
```

During the function run the significant amount of memory is freed, because the whole binary model, which is loaded in the memory after the instance initialization, is freed.
9 Eyedentify SDK Licensing

Eyedentify SDK uses the 3rd party framework developed by Gemalto for software protection and licensing. The SDK is protected against reverse engineering and unlicensed execution using the hardware USB keys. The SDK cannot be used without the USB license key with the valid license.

9.1 License Key Types

The SDK allows to load the license from several hardware keys types, which are listed in the following table. The keys differ in number of licenses which can be contained (Pro and Max versions), by physical dimensions, ability to contain timed licenses (Time versions) and ability to distribute the licenses over the network (Net versions).

<table>
<thead>
<tr>
<th>SKU</th>
<th>Product</th>
<th>SKU</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>SH-PRO</td>
<td>Sentinel HL Pro</td>
<td>SH-BRD</td>
<td>Sentinel HL Max (Board form factor)</td>
</tr>
<tr>
<td>SH-MAX</td>
<td>Sentinel HL Max</td>
<td>SH-TIM</td>
<td>Sentinel HL Time</td>
</tr>
<tr>
<td>SH-MIC</td>
<td>Sentinel HL Max (Micro form factor)</td>
<td>SH-NET</td>
<td>Sentinel HL Net</td>
</tr>
<tr>
<td>SH-CHP</td>
<td>Sentinel HL Max (Chip form factor)</td>
<td>SH-NTT</td>
<td>Sentinel HL NetTime</td>
</tr>
</tbody>
</table>

9.2 Licenses Overview

Several licenses are available for the Eyedentify SDK. The licenses differ in the type of the binary models which can be loaded, the time when the license is valid and the number of allowed recognition function executions.

9.2.1 Perpetual license

The perpetual license is the less restricted license available. It allows the user to use the license in one instance for unlimited time and unlimited number of executions. This license type is used for products which will be deployed to the end-user.

9.2.2 Time limited license

The time limited license allows to add the restriction on time when the license is valid. The date of the end of the license validity or the number of the days for which the license is valid after the first usage can be set. This license can be set on Time keys only (see License Key Types). This type of license is used mainly in the Developer package license.
9.2.3 Execution counting

The execution counting license allows to count how many times was the license logged in. The SDK is designed in the way it logs in the license every time the descriptor is computed. It allows to limit the number of descriptor computation executions with the license. This type of license is used mainly in the Developer package license.

9.3 License Management

The license protection software provides the web interface for license management. The web interface can be found on the address [http://localhost:1947](http://localhost:1947) opened in the common web browser. It allows the user to list the connected license keys, see the details of the arbitrary license key, update the license and several other functions.

9.3.1 Connected license keys

On the address [http://localhost:1947/_int_/devices.html](http://localhost:1947/_int_/devices.html) the list of license keys currently plugged in the computer is available. The list contains basic information about each key which includes: location of the key (Local or IP/name of the remote machine), Vendor ID, Key ID, Key Type, Configuration, Version and the number of connected Sessions. Each key also allows to list the contained license products, features and sessions using the buttons Products, Features and Sessions. The USB key LED can be blinked to easy identification using the button Blink on. The unique key identification file can be downloaded using the button C2V.

9.3.2 License key details

The detailed info about the plugged key can be get by clicking on the button Features in the Connected license keys list or on the address http://localhost:1947/_int_/features.html?haspid=KEYID, where the KEYID is the ID of the key. The web page contains information about the licenses contained on the key. The set of the Features represents the license. Each Feature controls different part of the SDK workflow (initialization, binary model selection, descriptor computation, ...).

9.4 License Update

The license is updated using the special *.v2c file which is emitted by the licensor of the software. The license update file is generated for specific license key ID and only that key can be updated using the file. There are two ways of updating the license: Web interface and Command line.

The license update must be done on the computer, where the protection software supplied with the SDK package is installed. For more information about the protection software installation see the chapter Installation Guide.

**IMPORTANT:**

Hardware protection key dongle with license to update needs to be connected to the machine where the license update file is applied.

9.4.1 Web interface

First option allows user to update the license using the web interface of the license management software Sentinel Admin Control Center. The web interface which can be opened in all modern browsers is located at http://localhost:1947/_int_/checkin.html.

![Sentinel Admin Control Center](image)


How to update the license:

2. Click on Choose File button and select *.v2c file which you want to use for update.
3. Click on Apply File button.
4. Webpage with the result of the license update is shown.
9.4.2 Command line

Second option to update the license is using Windows command line or Linux console. This approach can be very useful when applying update remotely or on many devices. It is also suitable for automating the license update procedure. This option requires basic knowledge of Windows command line or Linux console. The license update file *.v2c is applied using the hasp_update utility from the folder hasp/ located in the corresponding SDK package root.

**Windows command line**

Run the hasp_update utility with following parameter and *.v2c file on the selected machine:

```
C:\product\hasp> ./hasp_update u /path/to/v2c/license.v2c
```

When command runs with no error, the license is updated.

**Linux console**

Run the hasp_update utility with following parameter and *.v2c file on the selected machine:

```
eyedea@eyepc:~/product/hasp$ ./hasp_update u /path/to/v2c/license.v2c
```

When command runs with no error, the license is updated.
10 Third Party Software

The Eyedentify SDK uses third party software libraries, in accordance with their licenses. The licenses can be found under [EyedentifySDK]/documentation/3rdparty-licenses.

Here is a complete list of all libraries used, in alphabetical order:

- Boost
- Caffe
- CUDA
- OpenBLAS
- OpenCV
- OpenSSL
- Protobuf
- 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/)."