Anonymizer SDK

Developer's Guide

Version 6.x

eyedea RECOGNITION

ADVANCED COMPUTER VISION SOLUTIONS

Contact:

Address:

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

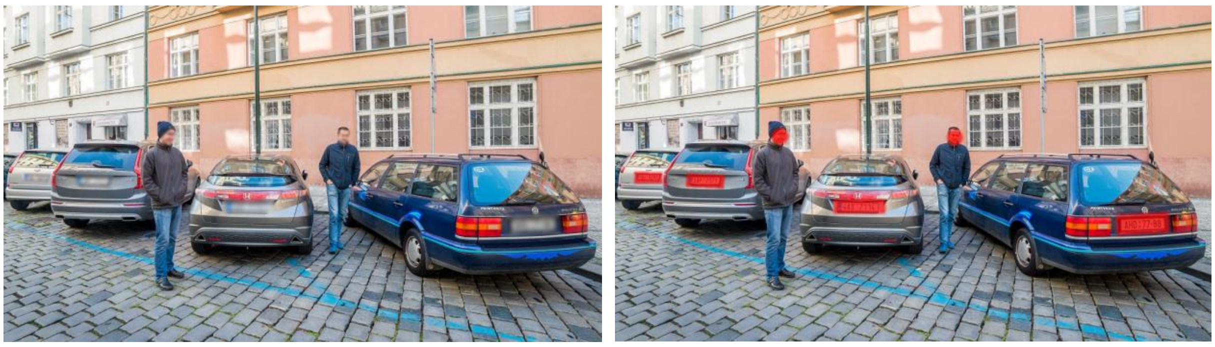
web: http://www.eyedea.cz

email: info@eyedea.cz

Table of Contents

1 Product Description

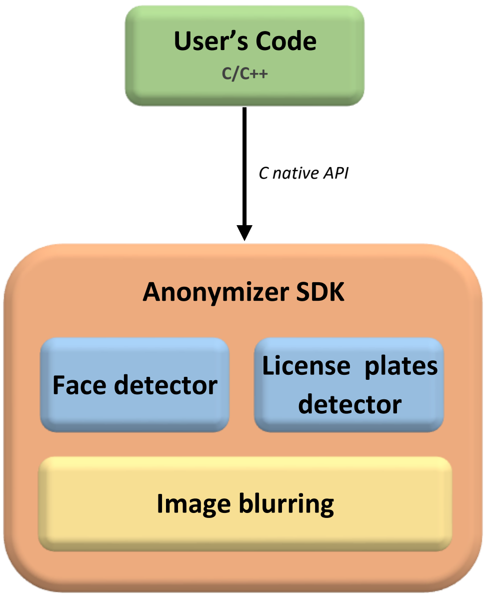
Anonymizer SDK is a cross-platform software library designed to provide easy anonymization of RGB images. The software detects and blurs faces and/or car license plates in various scales and orientations, with support for high resolution spherical images. Package includes command line application for batch processing of images. Both one-line and two-lines license plates with EU size or similar are supported (520x110mm, 280x200mm, 320x160mm and similar). Detection of other types of license plates on request.



Example of anonymized image (left) and image with highlited detections for better inspection (right).

1.1 Technical Details

Anonymizer SDK consists of three parts – face detector, license plate detector and image blurring. Face detector or license plate detector parts can each contain multiple detectors for better results for example one detector for one-line plates and one for two-line plates. Areas found by detectors are then seamlessly blurred or they can be highlighted for better visual inspection of results.



The Anonymizer library provides following APIs:

- C native API

Officially supported operating systems and platforms:

- Windows 7, 8, 8.1 and 10
  - 64-bit (Visual Studio 2015)
- Ubuntu 16.04 and higher
  - 64-bit
- Other platforms on request

2 Distribution Contents

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

- [Anonymizer SDK]/ ……………………………………………………………… *Distribution main folder*
    - AnonymizerSDK ……………………………………………………… *Anonymizer engine folder*
        - include ……………………………………………………… *Anonymizer header files folder*
        - lib ……………………………………………………………… *Anonymizer libraries folder*
        - models ……………………………………………… *Anonymizer detectors models folder*
    - applications ……………………………………………… *Anonymizer applications folder*
        - anonymizer-cmd ……………………………………… *Batch processing application folder*
    - examples ……………………………………………………… *Anonymizer examples folder*
        - example-files ……………………………………… *Files processing example folder*
        - example-buffers ………………………………… *Buffers processing example folder*
    - hasp ……………………………………………………… *License management software folder*
    - docs ……………………………………………………………… *SDK documentation folder*
    - data ………………………………………………………………………… *Example data folder*
    - License.txt ……………………………………………………………………… *SDK license file*
    - ReadMe.txt …………………………………………………………………… *SDK readme file*
    - ReleaseNotes.txt …………………………………………………… *SDK release notes file*

3 Hardware Requirements

3.1 Minimal Requirements

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

3.2 Recommended Requirements

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

3.3 Supported Operating Systems

3.3.1 Windows

- Microsoft Windows 7/8/8.1/10
  - x64 platform

3.3.2 Linux

- Ubuntu 16.04 and higher
  - x86_64 platform

4 Installation Guide

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

4.1 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).

4.2 Installation

4.2.1 Windows

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

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

4.2.2 Linux

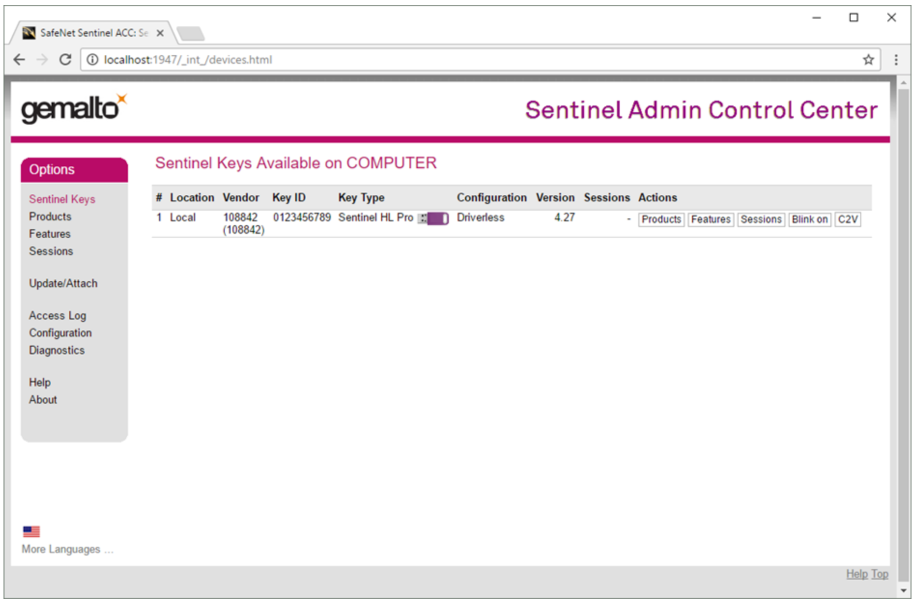
Follow these steps to install Sentinel License Manager on a Linux machine:

- Start the command line and navigate to **[Anonymizer_SDK]/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.)

4.3 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 Anonymizer 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.



Sentinel Keys Available on COMPUTER

#	Location	Vendor	Key ID	Key Type	Configuration	Version	Sessions	Actions
1	Local	108842 (108842)	0123456789	Sentinel HL Pro	Driverless	4.27	-	Products Features Sessions Blink on C2V

Illustration 1: Sentinel License Manager screenshot.

4.4 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 in the antivirus in case it uses its own firewall.

4.5 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 *Anonymizer SDK Licensing*.

4.6 License Error Codes

The error codes are outputted to error stream of the application (typically *stderr*) using Anonzmizer 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 Anonymizer 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 Anonymizer SDK is encrypted for enhanced software protection. On the other hand, on Linux systems, 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.

5 SDK Application Interface

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

5.1 Structures

Document section Structures covers all the information about structures used in the SDK's public application interface. Structure *AnROI* is used to define image area, *AnParams* is used to store anonymization parameters, *AnBuffer* is used to enclose buffers data and *AnState* is used as Anonymizer engine handle.

5.1.1 AnState

```
typedef void* ANState;
```

Type *AnState* is used as a handle to Anonymizer SDK library.

5.1.2 AnBuffer

```
typedef struct {
    unsigned char*    raw_data;
    size_t            length;
} AnBuffer;
```

Structure *AnBuffer* is used to enclose image buffer. The structure contains following fields:

- **raw_data**
  Byte array of data (raw image BGR data, JPEG encoded data).
- **length**
  Length of buffer in bytes.

5.1.3 AnROI

```
typedef struct {
    int             x;
    int             y;
    unsigned int    width;
    unsigned int    height;
} AnROI;
```

AnROI (Region Of Interest) structure defines image area where anonymization is applied. The ROI is defined by its top left point and width and height. The active pixels of regions are defined by closed ranges <x,x+width-1>, <y,y+height-1>. The structure contains following fields:

- **x**
  Defines top left column of ROI.
- **y**
  Defines top left row of ROI.
- **width**
  Defines width of ROI.
- **height**
  Defines height of ROI.

5.1.4 AnParams

```
typedef struct {
    AnROI  ROI;
    int    mode;
    int    showDetections;
    int    jpegQuality;
    int    keepExif;
    int    panoramic;
    double faceConfidenceThr;
    double faceDetSizeFactor;
    int    faceNumBlurPass;
    double lpConfidenceThr;
    double lpDetSizeFactor;
    int    lpNumBlurPass;
    int    verbose;
```

Structure *AnParams* contains set of parameters controlling anonymization process. This structure allows to define various parameters in runtime. Anonymization consists of two detection modules (face and licence plate), for both the control of balance between true positive detections and false alarms can be set by **faceConfidenceThr**, **lpConfidenceThr** parameters. Higher value means less false detections and lower number of true detections, lower value means more true detections as well as more false detections. Anonymization area and anonymization strength are controled by **faceDetSizeFactor** and **faceNumBlurPass** (**lpDetSizeFactor** and **lpNumBlurPass** respectively). For visualisation purposes the **showDetections** flag can be used. Detections are then highlighted insted of blurred. Parameter **keepExif** is used to copy EXIF information from original file or JPG buffer to output file or JPG buffer. The structure contains following fields:

- **ROI**
  Region of interest - the area scanned for detections, default area is from_col=0, to_col=MAX_INT, from_row=0, to_row=MAX_INT. Values higher than size of processed image are converted to nearest applicable value.
- **mode**
  Flag indicating whether to anonymize faces or license plates or both. Default value is ANONYMIZE_FACE | ANONYMIZE_LP (both faces and license plates are anonymized).
- **showDetections**
  Debugging flag indicating, that detections will be highlighted instead of blurred. Default value is 0.
- **jpegQuality**
  Output image JPEG quality 0..100, default value is 90.
- **keepEXIF**
  Copy EXIF and other non-image information from original file or JPG buffer to output file or JPG buffer. This option does nothing when file is not JPG or when used for image buffer anonymization. Note that EXIF image thumbnail is NOT anonymized and that this may not work on nonstandard EXIFs. Default value is 0.
- **panoramic**
  Enable copying of left and right borders to detect objects on the edges of panoramic photos. Default value is 0.
- **faceConfidenceThr**
  The minimal confidence of a face to be anonymized. (Detector return confidences > 0). Default value is 10.0.
- **faceDetSizeFactor**
  Parameter to enlarge (or shrink) faces anonymized area. Value lower than 1.0 means anonymized area will be smaller than detection, value higher than 1.0 mean that anonymized area will be larger than detection. Default value is 1.0.
- **faceNumBlurPass**
  Number of blur passes over face detection. Higher number means stronger blur. Default value is 1.
- **lpConfidenceThr**
  The minimal confidence of a license plate to be anonymized. (Detector return confidences > 0). Default value is 6.0.
- **lpDetSizeFactor**
  Parameter to enlarge (or shrink) license plates anonymized area. Value lower than 1.0 means anonymized area will be smaller than detection, value higher than 1.0 mean that anonymized area will be larger than detection. Default value is 1.0.
- **lpNumBlurPass**
  number of blur passes over license plate detection. Higher number means stronger blur. Default value is 1.
- **verbose**
  Enables logging of anonymization progress on to **stdout**. Default value is 0.

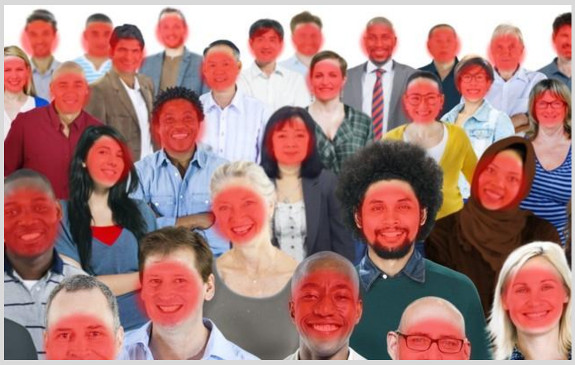
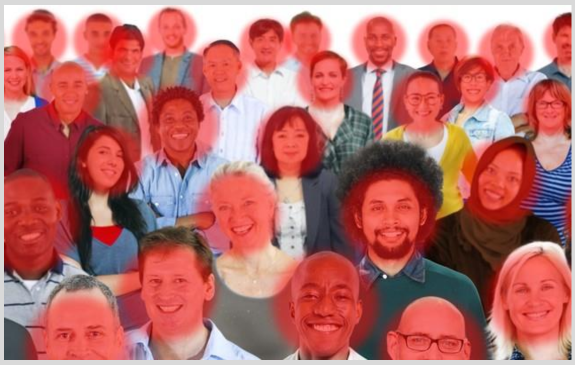
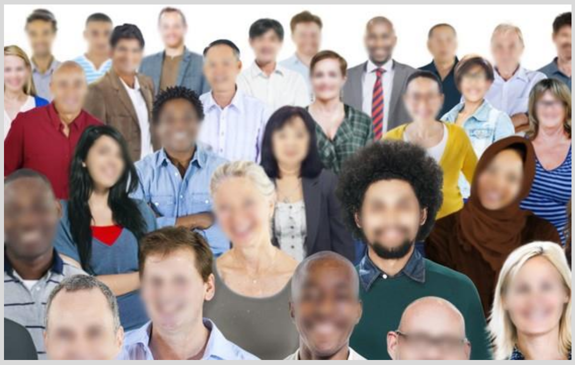
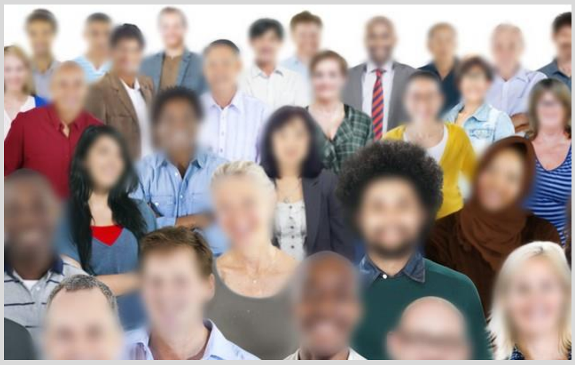
	dFaceDetSizeFactor=1.0	dFaceDetSizeFactor=1.5
iShowDetections=1		
iShowDetections=0		

Table 1: Comparison of normal blurring vs. showing detections in red and comparison of different sizes of blurring regions

iFaceNumBlurPass=1	iFaceNumBlurPass=2	iFaceNumBlurPass=3

Table 2: Comparison of different blur strength

5.2 Functions

This chapter contains the definition of the Anonymizer library functions which are present in the public API. The chapter is divided into three parts. First part describes function for manipulation with Anonymizer engine, second part describes functions for image anonymization and third section describes all other functions.

5.2.1 Engine manipulation functions

This part defines the API functions which are designed to initialize Anonymizer engine and to free Anonymizer engine as well as to get engine version. The functions are: *anInit(), anFree()* and *anVersion().* These functions are defined in the **Anonymizer.h** file.

anInit

Initializes the Anonymizer engine and loads and set-ups all detection modules.

Specification:

```
int anInit(const char* sdk_directory, const char* ini_filename, ANState* state)
```

Inputs:

- **sdk_directory**
  Path of the AnonymizerSDK directory.
- **ini_filename**
  Config file name (if NULL, default "**config.ini**" is used).

Outputs:

- **state**
  Pointer to *AnState* structure.

Returns:

- 0 – Anonymizer was successfully initialized.
- *other* – Error while initializing Anonymizer.

Description:
The function *anInit()* initializes Anonymizer engine and loads detection modules for face detection and license plates detection and load their configuration files. Input parameters are path to AnonymizerSDK directory where detection modules and configuration files are located and filename of main config. Function returns zero on success or error code if it fails.

Example:

```
ANState  state;      // Anonymizer state handler
int ern;
if ((ern = anInit("../../AnonymizerSDK/", "config.ini", &state)) != 0)
{
    // error handling
}
```

anFree

Frees initialized Anonymizer engine.

Specification:

```
void anFree(ANState state);
```

Inputs:

- **state**
  Pointer to the initialized Anonymizer engine instance.

Description:
The function *anFree()* is used for freeing the Anonymizer engine. When the engine 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 *AnState* which was created using *anInit()* function during engine initialization.

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

Example:

```
ANState  state;      // Anonymizer state handler
anInit("../../AnonymizerSDK/", "config.ini", &state);  // Anonymizer init

    // working with state

anFree(state);       // Free the Anonymizer state
```

anVersion

Returns the Anonymizer engine version.

Specification:

```
const char* anVersion(int verbose);
```

Inputs:

- **verbose**
  Verbosity flag. If enabled, function will return more information about SDK.

Returns:

- Anonymizer engine version.

Description:
The function *anVersion()* returns string with version of Anonymizer engine e. g. "Anonymizer v5.0.0.8694"

Example:

```
printf("Version: %s\n\n", anVersion(0));      // print Anonymizer version
```

5.2.2 Anonymization functions

This part defines the API functions which are designed to anonymize images. Function *anAnonymize()* anonymizes image files defined by its filename, function *anAnonymizeImageBuffer()* anonymizes raw image data supplied in buffer and *anAnonymizeJpegBuffer()* anonymize JPEG encoded image data supplied in buffer. Function *anFreeBuffer()* frees previously allocated buffer and function *anGetDefaultParams()* returns default anonymization parameters values of *AnParams* structure. These functions are defined in the **Anonymizer.h** file.

anAnonymize

Loads image file, runs anonymization and save result as JPEG file.

Specification:

```
int anAnonymize(const char* src_image_filename, const char* dst_image_filename,
                AnParams* params, ANState state);
```

Inputs:

- **src_image_filename**
  Input image filename. Anonzmizer can load JPG, PNG, TIF and BMP files.
- **dst_image_filename**
  Output image filename (in JPEG format).
- **Params**
  Pointer to AnParams structure with Anonymization parameters.
  Use NULL for default parameters.
- **State**
  AnState Anonymizer state.

Returns:

- 0 – Successfully anonymized.
- *other* – Error during anonymization.

Description:

The function *anAnonymize()* loads image file specified by **src_image_filename** parameter, run anonymization with parameters specified by **Params** parameter and save the anonymized image as JPEG file specified by **dst_image_filename**.

Example:

```
if (anAnonymize("image.jpg", "image_anonymized.jpg", NULL, state)!=0)
{
    // error handling
}
```

anAnonymizeImageBuffer

Runs Anonymization on raw image buffer and outputs result as raw image buffer.

Specification:

```
int anAnonymizeImageBuffer(AnBuffer srcBuffer, unsigned int width, unsigned int height,
                              AnParams* params, ANState state, AnBuffer* dstBuffer);
```

Inputs:

- **srcBuffer**
  Input structure with BGR data buffer, row-wise, 3 bytes (unsigned chars) per pixel,
  index = 3*col + row*3*width.
- **width**
  Image width.
- **height**
  Image height.
- **Params**
  Pointer to *AnParams* structure with Anonymization parameters.
  Use NULL for default parameters.
- **State**
  AnState Anonymizer state. See *anInit()* function.

Outputs:

- **dstBuffer**
  Anonymized image in data buffer (BGR).

Returns:

- 0 – Successfully anonymized.
- *other* – Error during anonymization.

Description:

The function *anAnonymizeImageBuffer()* runs anonymization on raw BGR image data supplied in buffer aligned row by row and returns result in buffer with same format. Output buffer **dstBuffer** must be freed when it is no longer needed. See *Anonymizer Example – buffers* for more information.

Example:

```
AnBuffer srcBuffer, dstBuffer;  // input and output buffers

// fill srcBuffer with some image data

if (fcnAnAnonymizeImageBuffer(srcBuffer, img_width, img_height, NULL, state, &dstBuffer) != 0)
{
    // error handling
}
```

IMPORTANT: Always free **dstBuffer** structure with result when it is not needed anymore using *anFreeBuffer()* function otherwise your program will have memory leaks.

anAnonymizeJpegBuffer

Runs Anonymization on JPEG image buffer and outputs result as JPEG image buffer.

Specification:

```
int anAnonymizeJpegBuffer(AnBuffer srcBuffer, AnParams* params, ANState state,
                             AnBuffer* dstBuffer);
```

Inputs:

- **srcBuffer**
  Input structure with JPEG encoded data buffer.
- **Params**
  Pointer to *AnParams* structure with Anonymization parameters.
  Use NULL for default parameters.
- **State**
  AnState Anonymizer state. See *anInit()* function.

Outputs:

- **dstBuffer**
  Anonymized image in JPEG data buffer.

Returns:

- 0 – Successfully anonymized.
- *other* – Error during anonymization.

Description:

The function *anAnonymizeJpegBuffer()* runs anonymization on JPEG image data (JPEG encoded data) supplied in buffer and returns result in buffer with same format. Output buffer **dstBuffer** must be freed when it is no longer needed. See *Anonymizer Example – buffers* for more information.

Example:

```
AnBuffer srcBuffer, dstBuffer;  // input and output buffers

// fill srcBuffer with some image data

if (ern=fcnAnAnonymizeJpegBuffer(srcBuffer, NULL, state, &dstBuffer) != 0)
{
    // error handling
}
```

IMPORTANT: Always free **dstBuffer** structure with result when it is not needed anymore using *anFreeBuffer()* function otherwise your program will have memory leaks.

anFreeBuffer

Frees the image buffer filled by Anonymizer SDK functions.

Specification:

```
void anFreeBuffer(AnBuffer buffer);
```

Inputs:

- **buffer**
  AnBuffer structure to free.

Description:

The function *anFreeBuffer()* frees data buffer previously allocated by *anAnonymizeImageBuffer()* or *anAnonymizeJpegBuffer()* function.

Example:

```
fcnAnFreeBuffer(dstBuffer); // free buffer allocated by Anonymizer
```

anGetDefaultParams

Fills *AnParams* structure with default values.

Specification:

```
int anGetDefaultParams(AnParams* parameters);
```

Outputs:

- **parameters**
  Pointer to *AnParams* structure.

Returns:

- 0 – Success.
- *other* – Failure.

Description:

The function *anGetDefaultParams()* fills *AnParams* structure with default values. Pass pointer to statically or dynamically allocated *AnParams* structure to be filled with default values.

Example:

```
AnParams params;
fcnAnGetDefaultParams(&params);
```

5.2.3 Other functions

This part defines other API functions. Function *anGetErrorMsg()* is used to get error message for error code. These functions are defined in the **Anonymizer.h** file.

anGetErrorMsg

Gets a message string related to the given error code.

Specification:

```
const char* anGetErrorMsg(int ern);
```

Inputs:

- **ern**
  An error code of the message to be retrieved.

Returns:

- The null-terminated string containing the error message.

Description:
The function *anGetErrorMsg()* returns an error message of the error specified by error code.

Example:

```
int erno; // variable to save return value

// run some function and save return value to erno

if(erno != 0)
{
    printf("Function failed: %s\n", anGetErrorMsg(erno)); // Print error message
}
```

6 Examples

This chapter contains description of examples which are contained in the SDK package. 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. First example demonstrates how to anonymize image files directly. Second example demonstrates how to anonymize images when you already have image data in memory either encoded in JPEG or as raw BGR data.

6.1 Anonymizer Example – files

This basic example demonstrates how to use Anonymizer SDK to anonymize image files directly from filesystem and write result back to filesystem. File reading and writing is handled by Anonymizer SDK.

The example is in the folder **[Anonymizer_SDK]/examples/example-files/**. 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.

6.1.1 Initialization of Anonymizer engine

First thing to do is the Anonymizer engine initialization using the *anInit()* function. Parameters of this function are directory where AnonzmizerSDK modules are located (e. g. **../../AnonymizerSDK** for default package directory structure) and name of main configuration file (or NULL for default **config.ini** file). Almost all API functions has integer return value and returns 0 on success or error code. To see the error message use a *anGetErrorMsg()* function. More verbose error logs are printed to stderr.

```
#define ANONYMIZER_SDK_DIR "../../AnonymizerSDK/"   // define path to AnonymizerSDK
#define ANONYMIZER_INI "config.ini"                 // define filename of main config
ANState  state;      // Anonymizer state handler
int ern;
if ((ern = anInit((char*)ANONYMIZER_SDK_DIR, (char*)ANONYMIZER_INI, &state)) != 0)
{
    // error handling
}
```

After successful initialization of engine, anonymization functions can be used.

6.1.2 Setting anonymization parameters

The anonymization parameters are set using *AnParams* structure. Example of *AnParams* initialization:

```
AnParams parameters;
/* Region Of Interest (detection area) */
parameters.ROI.x              = 0;
parameters.ROI.y              = 0;
parameters.ROI.width          = 1000000;
parameters.ROI.height         = 1000000;
/* anonymize faces and License Plates */
parameters.mode        = ANONYMIZE_FACE | ANONYMIZE_LP;
/* Output image quality */
parameters.jpegQuality        = 90;
/* The minimal detection confidence of a face/LP to be anonymized. */
parameters.faceConfidenceThr = 15.;
parameters.lpConfidenceThr   = 15.;
/* Parameter to enlarge anonymized area (1.0 == exactly detection) */
parameters.faceDetSizeFactor = 1.;
parameters.lpDetSizeFactor   = 1.;
/* Number of blur passes over face/LP detection */
parameters.faceNumBlurPass   = 1;
parameters.lpNumBlurPass     = 1;
/* Debugging flag indicating, that detections will be highlighted instead of blurred */
parameters.showDetections    = 1;
parameters.verbose           = 0;
parameters.panoramic         = 0;
parameters.keepEXIF          = 0;
```

The **ROI** parameters define area where the anonymization is applied. (If the values of **ROI** are higher than the image dimensions then these higher values are ignored and the whole image is processed.)

Parameters **faceConfidenceThr** and **lpConfidenceThr** define detection thresholds for face detection and license plate detection, respectively. The lower values result in more detections together with more false positives. The higher values decrease number of false positives and increase number of false negative. The recommended range is <0,25>, the recommended value is 15.

Parameters **faceDetSizeFactor** and **lpDetSizeFactor** define the size of the blurred area with respect to the size of detected object. Values below 1 decrease the area size, values above 1 increase the area size (see table 1 in description of *AnParams* structure).

Parameters **faceNumBlurPass** and **lpNumBlurPass** denote number of blurring passes over the detected object, i.e. they determine the blurring strength (see table 2 in description of *AnParams* structure).

The **showDetections** flag is used highlighted detections for testing purposes (see table 1 in description of *AnParams* structure).

The **verbose** flag switches on/off progress information to the **stdout**.

The **panoramic** flag switches on/off copying of left and right edges of images which is useful for anonymizing of 360 degrees panoramic images where image on the left edge continues at right side of image. Depending on other parameters, this setting increase processing time by about 5-10%.

The **keepEXIF** flag can be used for JPG files or buffers to copy EXIF data from source to output file/JPG buffer. Note that EXIF thumbnail is NOT anonymized. May not work on nonstandard EXIFs.

6.1.3 Image anonymization

To anonymize image located on drive function *anAnonymize()* is used. Parameters are input and output image filenames, anonymization parameters and Anonymizer state. Input file can be JPG, PNG, BMP or TIF file, output file is in JPG format.

```
string input  = "../../data/image.jpg";
string output = "image_anonymized.jpg";
// run anonymization - if anonymization parameters are NULL, default values are used
if (anAnonymize((char*)input.c_str(), (char*)output.c_str(), parameters, state)!=0)
{
    // error handling
}
```

6.1.4 Cleaning up

At the end, when the work with the Anonymizer instance is finished (for example at the end of the program), it must be freed. To free Anonymizer, use the API function *anFree()*, which is designed for such purpose.

```
anFree(state);   // Free the Anonymizer state
```

6.2 Anonymizer Example – buffers

This basic example demonstrates how to use Anonymizer SDK to anonymize images in buffers. This can be useful when data are already in memory. Anonymizer can process raw images data in BGR format or JPEG encoded data.

The example is in the folder **[Anonymizer_SDK]/examples/example-buffers/**. 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.

6.2.1 Initialization of Anonymizer engine

First thing to do is the Anonymizer engine initialization using the ***anInit****()* function. Parameters of this function are directory where AnonzmizerSDK modules are located (e. g. **../../AnonymizerSDK** for default package directory structure) and name of main configuration file (or NULL for default **config.ini** file). Almost all API functions has integer return value and returns 0 on success or error code. To see the error message use a ***anGetErrorMsg****()* function. More verbose error logs are printed to **stderr**.

```
#define ANONYMIZER_SDK_DIR "../../AnonymizerSDK/"   // define path to AnonymizerSDK
#define ANONYMIZER_INI "config.ini"                 // define filename of main config
ANState  state;      // sdk state handler
int ern;
if ((ern = anInit((char*)ANONYMIZER_SDK_DIR, (char*)ANONYMIZER_INI, &state)) != 0)
{
    // error handling
}
// initialization ok
```

After successful initialization of engine, anonymization functions can be used.

6.2.2 JPEG buffer anonymization

To anonymize buffer with JPEG encoded image, function ***anAnonymizeJpegBuffer****()* is used. Parameters are input and output buffers, anonymization parameters and Anonymizer state. Functions **fread_buffer()** and **fwrite_buffer()** are simple functions to read and write binary files. Codes of these functions are included in example in file **example-buffers.cpp**.

```
// Anonymization of JPEG buffer
string input  = "../../data/image.jpg";
string output = "image_anonymized.jpg";
AnBuffer srcBuffer, dstBuffer;
if (fread_buffer((char*)input.c_str(), &(srcBuffer.raw_data), &(srcBuffer.length)) != 0)
{
    // error handling
}
if (ern=fcnAnAnonymizeJpegBuffer(srcBuffer, NULL, state, &dstBuffer) != 0)
{
    // error handling
}
if (fwrite_buffer((char*)((output + ".JpegBuffer.jpg").c_str()), dstBuffer.raw_data,
dstBuffer.length) != 0)
{
    // error handling
}
delete [] srcBuffer.raw_data;
fcnAnFreeBuffer(dstBuffer);
```

6.2.3 Image buffer anonymization

To anonymize buffer with raw BGR image, function ***anAnonymizeImageBuffer()*** is used. Parameters are input and output buffers, image width and height, anonymization parameters and Anonymizer state. OpenCV library is used in this example for image reading and writing.

```
// anonymization of image buffer
string input  = "../../data/image.jpg";
string output = "image_anonymized.jpg";
cv::Mat imageBGR, imageAnonymized;
imageBGR = cv::imread(input.c_str(), cv::IMREAD_COLOR); // Read the file - BGR order
srcBuffer.raw_data = imageBGR.data;
unsigned char* anonymizedBuffer = NULL; // anonymized buffer - ouptut of anonymization
cv::Size s = imageBGR.size();
if (fcnAnAnonymizeImageBuffer(srcBuffer, s.width,s.height, NULL, state, &dstBuffer) != 0)
{
        // error handling
}
memcpy(imageBGR.data, dstBuffer.raw_data, imageBGR.total()*imageBGR.elemSize());// set anonymized
data back to BGR image
cv::imwrite(output + ".Buffer.jpg", imageBGR); // and write

fcnAnFreeBuffer(dstBuffer); // free buffer allocated by Anonymizer
```

6.2.4 Cleaning up

At the end, when the work with the Anonymizer instance is finished (for example at the end of the program), it must be freed. To free Anonymizer, use the API function ***anFree()***, which is designed for such purpose.

```
anFree(state);   // Free the Anonymizer state
```

7 Anonymizer command line application

Anonymizer SDK package contains anonymizer-cmd application for batch anonymization of images located at **[Anonymizer_SDK]\applications\anonymizer-cmd**. This command-line application uses simple interface, user sets input list of images to be processed, and output directory for anonymized images. Optional parameters are JPEG output quality, mode to visualize detections instead of blurring, possibility to blur only registration plates or only faces and possibility to anonymize only selected area of images as well as few other options to control blurring parameters.

Usage of anonymizer is:

```
anonymizer [options] image_list
```

Where options can be:

```
-o=directory  sets the output directory for anonymized images
-q=n  jpeg quality n = 1..100 (90 default)
--keep_exif copy EXIF from original JPG file to output file
--lp  run registration plates anonymization only
--lp_thr= threshold on plates detections
--face  run face annonymization only
--face_thr= threshold on face detection
--roi=[x,y,width,height] anonymization ROI (region of interest)
--to_row= position of bottom row of rectangle to process
--from_col= position of left column of rectangle to process
--to_col= position of right column of rectangle to process
--face_num_blur_pass= strength of blurring of faces (1, 2, 3, ...)
--lp_num_blur_pass= strength of blurring of license plates (1, 2, 3, ...)
--face_size_factor= size of blurred area relative to size of detected face
--lp_size_factor= size of blurred area relative to size of detected lp
--show  colorize detections instead of blurring them
--panoramic  anonymize faces and plates divided by left and right edge
--verbose  more verbose run
```

These options can be printed by running `anonymizer --help`

Examples:

```
./anonymizer -o=output_dir image_list.txt
```

This command will anonymize both license plates and faces in images in **image_list.txt** and writes them to directory **output_dir**.

```
./anonymizer -o=o_dir –face --face_thr=20 image_list.txt
```

This command will anonymize only faces with score above 20 in images in **image_list.txt** and writes them to directory **o_dir**.

Where **image_list.txt** is text file with paths to image files to be processed, one file per line. Filepaths can be absolute or relative to anonymizer binary. Script **make_imagelist** is included in directory of **anonymizer-cmd**. Example of image list can be:

```
/local/data/images/DSC_5243.jpg
/local/data/images/DSC_5244.jpg
/local/data/images/DSC_5245.jpg
```

8 Anonymizer SDK Licensing

Anonymizer 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.

8.1 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).

SKU	Product	SKU	Product
SH-PRO	Sentinel HL Pro	SH-BRD	Sentinel HL Max (Board form factor)
SH-MAX	Sentinel HL Max	SH-TIM	Sentinel HL Time
SH-MIC	Sentinel HL Max (Micro form factor)	SH-NET	Sentinel HL Net
SH-CHP	Sentinel HL Max (Chip form factor)	SH-NTT	Sentinel HL NetTime

8.2 Licenses Overview

Several licenses are available for the Anonymizer 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.

8.2.1 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 image anonymization is executed. It allows to limit the number of processed images with the license. This is standard type of license for Anonymizer.

8.2.2 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 for anonymization of large image quantities.

8.2.3 Perpetual license

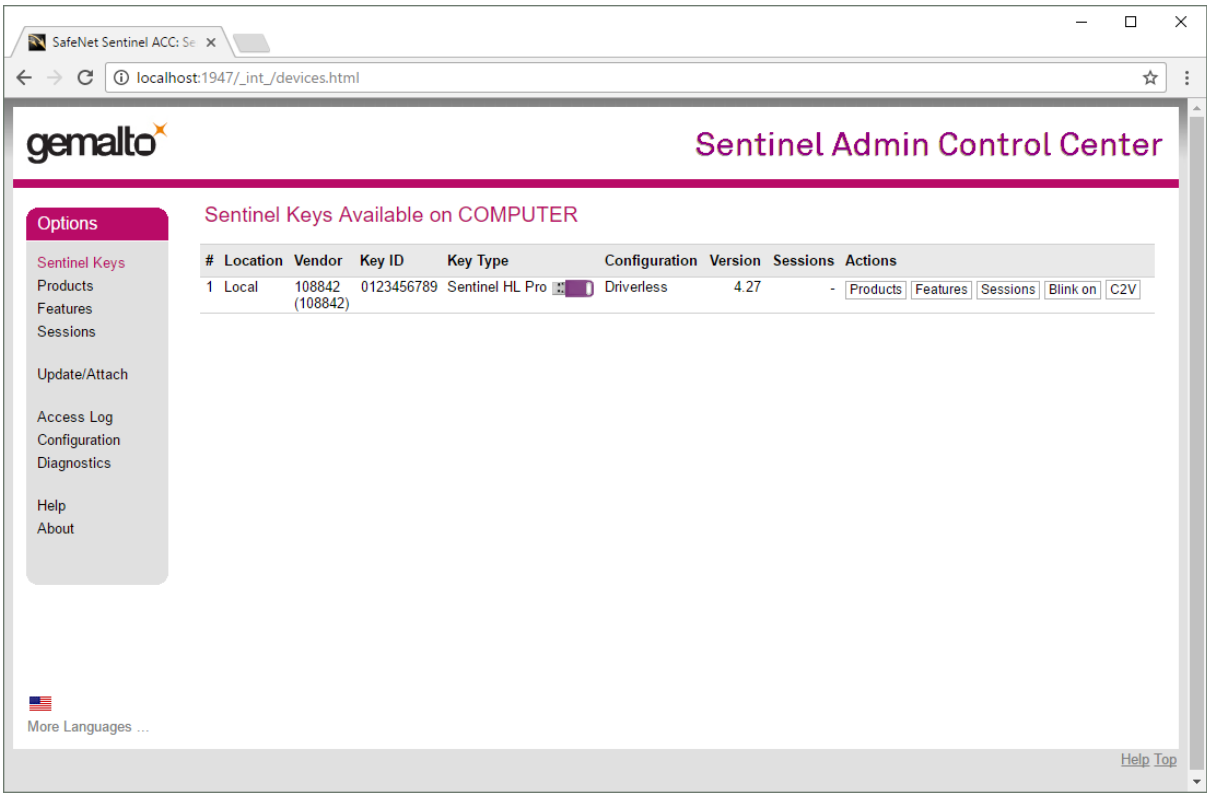
The perpetual license is the least restricted license available. It allows the user to use the license in one instance for unlimited time and unlimited number of executions. This license type is used for anonymization of very large image quantities.

8.3 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 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.

8.3.1 Connected license keys

On the address 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.



Sentinel Keys Available on COMPUTER

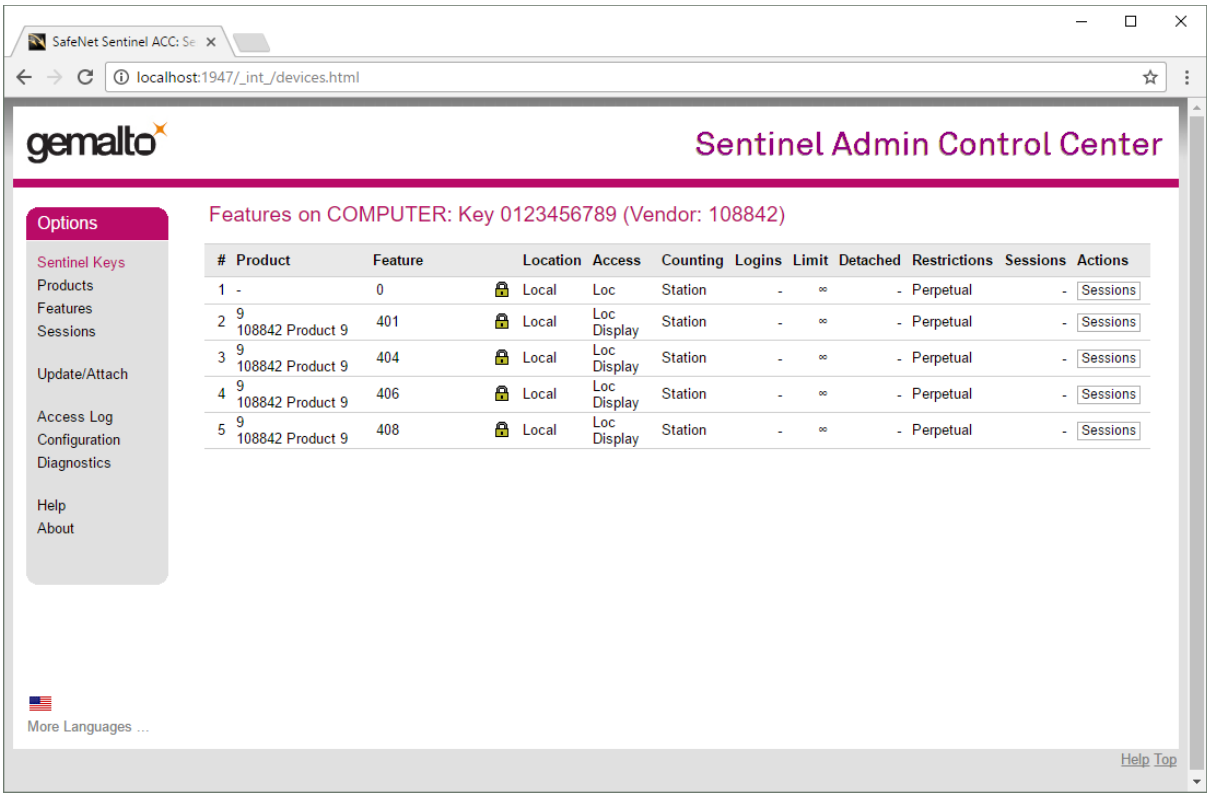
#	Location	Vendor	Key ID	Key Type	Configuration	Version	Sessions	Actions
1	Local	108842 (108842)	0123456789	Sentinel HL Pro	Driverless	4.27	-	Products Features Sessions Blink on C2V

Illustration 2: Web interface with list of plugged keys on http://localhost:1947/_int_/devices.html

8.3.2 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 can control different part of the SDK workflow (initialization, detection, anonymization, ...).



Sentinel Admin Control Center

Features on COMPUTER: Key 0123456789 (Vendor: 108842)

#	Product	Feature		Location	Access	Counting	Logins	Limit	Detached	Restrictions	Sessions	Actions
1	-	0	🔒	Local	Loc	Station	-	∞	-	Perpetual	-	Sessions
2	9 108842 Product 9	401	🔒	Local	Loc Display	Station	-	∞	-	Perpetual	-	Sessions
3	9 108842 Product 9	404	🔒	Local	Loc Display	Station	-	∞	-	Perpetual	-	Sessions
4	9 108842 Product 9	406	🔒	Local	Loc Display	Station	-	∞	-	Perpetual	-	Sessions
5	9 108842 Product 9	408	🔒	Local	Loc Display	Station	-	∞	-	Perpetual	-	Sessions

Illustration 3: Web interface with key 0123456789
details on http://localhost:1947/_int_/features.html?haspid=0123456789

8.4 License Update

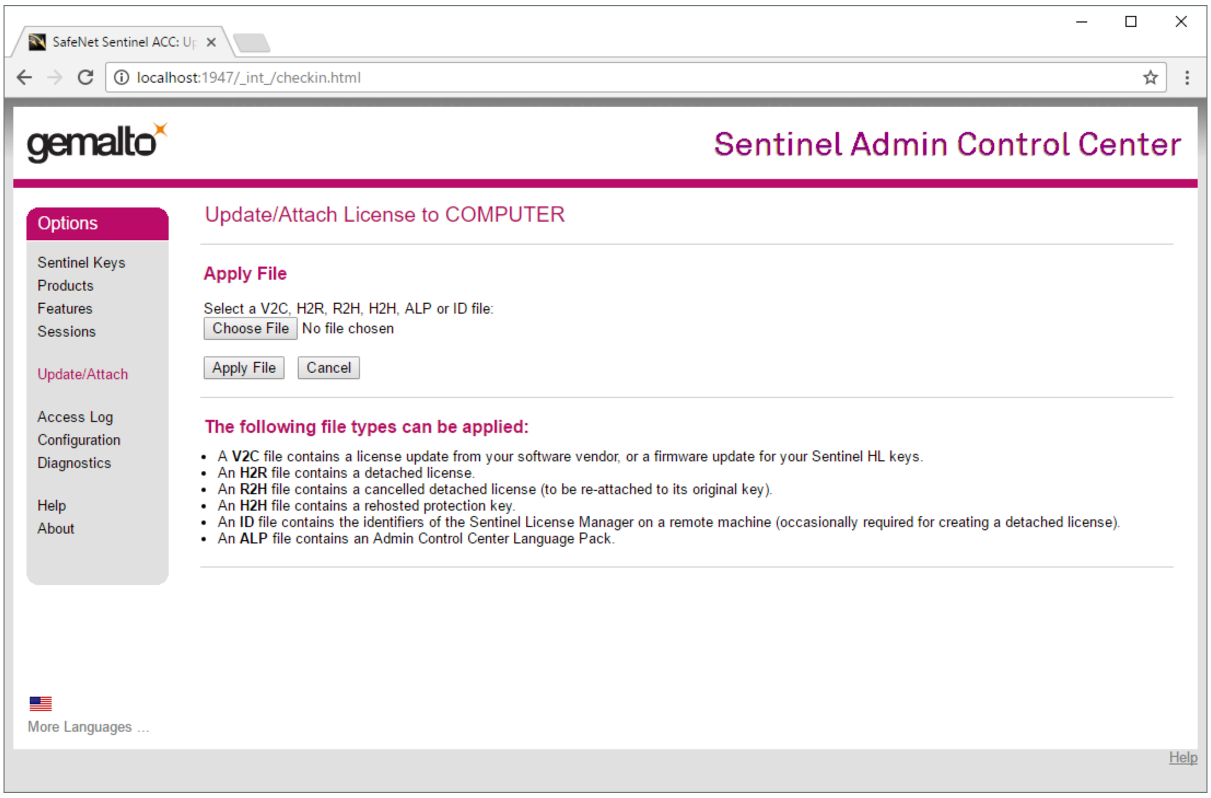
The license is updated using the special **.v2c* file which is emitted by the licensor of the software. The license update file is generated for specific license key ID and only that key can be updated using the file. There are two ways of updating the license: *Web interface* and *Command line*.

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.

8.4.1 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.



Update/Attach License to COMPUTER

Apply File

Select a V2C, H2R, R2H, H2H, ALP or ID file:

No file chosen

The following file types can be applied:

- A **V2C** file contains a license update from your software vendor, or a firmware update for your Sentinel HL keys.
- An **H2R** file contains a detached license.
- An **R2H** file contains a cancelled detached license (to be re-attached to its original key).
- An **H2H** file contains a rehosted protection key.
- An **ID** file contains the identifiers of the Sentinel License Manager on a remote machine (occasionally required for creating a detached license).
- An **ALP** file contains an Admin Control Center Language Pack.

Illustration 4: Web interface for license update on http://localhost:1947/_int_/checkin.html

How to update the license:

1. Open the link http://localhost:1947/_int_/checkin.html in the web browser.
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.

8.4.2 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 **[Anonymizer_SDK]/hasp/**.

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.

9 Third Party Software

The Anonymizer SDK uses third party software libraries, in accordance with their licenses. The licenses can be found under **[Anonymizer_SDK]/docs/3rdparty-licenses**.

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

- Iniparser
- OpenCV