

Eyedeia Anonymzier

Developer's guide

Version 7.0



ADVANCED COMPUTER VISION SOLUTIONS

Copyright © 2024, Eyedea Recognition s. r. o.

All rights reserved

Eyedea Recognition s. r. o. is not liable for any damage or loss caused by incorrect or inaccurate results or unauthorized use of the Anonymzier software.

Thales, the Thales logo, are trademarks and service marks of Thales S.A. and are registered in certain countries. Sentinel, Sentinel Admin Control Center and Sentinel Hardware Key are registered trademarks of Thales S.A..

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.

Microsoft Windows, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10, Windows 11, Windows logo and Visual Studio are registered trademarks of Microsoft Corporation in the United States and/or other countries.

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

Ubuntu and Canonical are registered trademarks of Canonical Ltd.

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

Python is a registered trademark of The Python Software Foundation. The Python logos (in several variants) are use trademarks of The Python Software Foundation as well.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

Other names may be trademarks of their respective owners.

All personal information in photos in this document were either anonymized or altered to avoid possibility of direct or indirect identification of any person.

Contact:

Address:	Eyedea Recognition, s.r.o. Vyšehradská 320/49 128 00, Prague 2 Czech Republic
web:	www.eyedea.cz
email:	info@eyedea.cz

Table of Contents

1	Product Description	4
1.1	Technical Details	4
2	Distribution Contents	5
3	Installation Guide	6
3.1	Pre-installation	6
3.2	Sentinel LDK Installation	6
3.3	Verification of Installation	6
3.4	Installation Failures	7
3.5	Managing Licenses	7
3.6	License Error Codes	8
3.7	TensorRT	8
3.8	OpenGL Prerequisites	9
4	SDK Application Interface	10
4.1	Structures	10
4.2	Functions	13
5	Examples	19
5.1	Anonymizer Example – files	19
5.2	Anonymizer Example – buffers	21
6	Anonymizer command line interface	23
7	Anonymzier Licensing	25
7.1	License Key Types	25
7.2	Licenses Overview	25
7.3	License Management	26
7.4	License Update	27
8	Third Party Software	29



1 Product Description

The Anonymizer SDK is a versatile, cross-platform software library designed for seamless anonymization of RGB images. It detects and blurs faces and car license plates at various scales and orientations, including support for high-resolution spherical images. The package includes a command-line application for batch image processing. It supports both single-line and multi-line license plates of EU and North America (Mexico, US, Canada) sizes or similar dimensions. Detection of other types of license plates is available upon request.



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

1.1 Technical Details

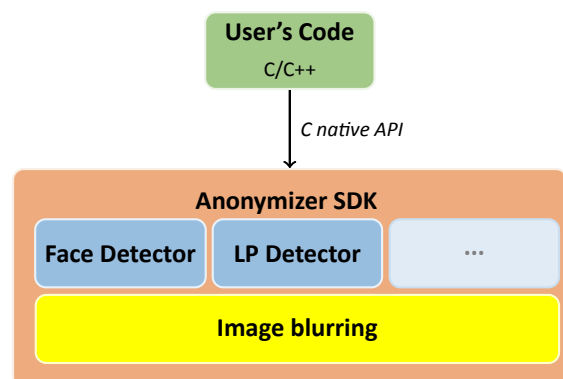
The Anonymizer SDK is a comprehensive tool that now includes multiple detection modules for enhanced privacy and data security in images. It comprises a face detector and an advanced detector for license plates and other objects such as car boxes and windshields. This modular approach allows for the adaptability of the SDK, with the potential to include additional detection modules as needed. Detected areas are then processed according to the user's requirements: they can be seamlessly blurred for privacy or highlighted to facilitate a thorough visual inspection of the results.

The Anonymizer library provides following APIs:

- C native API

Officially supported operating systems and platforms:

- Windows 10 and 11, 64 bit
- Ubuntu 20.04 and higher, 64 bit
- Other platforms on request



2 Distribution Contents

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

- 📁 [AnonymizerSDK] *distribution main folder*
 - 📁 sdk *Anonymizer engine folder*
 - 📁 include *Anonymizer header files*
 - 📁 lib *Anonymizer libraries*
 - 📁 plugins *Anonymizer backend plugins for detection*
 - 📁 models *Anonymizer detectors models folder*
 - 📁 applications *Anonymizer applications folder*
 - 📁 anonymizer-cli *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*
 - 📁 documentation *SDK documentation folder*
 - 📁 3rdparty-licenses *Licenses of used 3rd party backend libraries*
 - 📁 data *Example data folder*
 - 📁 tools *Folder with utilities for GPU devices listing, TensorRT model conversion*
 - 📄 LICENSE.txt *SDK license*
 - 📄 RELEASE_NOTES.txt *file with release notes for each SDK version*
 - 📄 README.txt *SDK readme file*



3 Installation Guide

Installation of the software licensing daemon is the first step to start using the Anonymzier. The library comes equipped with a standard third-party software licensing solution, Sentinel LDK by Thales. 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 execution of the encrypted Anonymzier binaries, and management of licenses using a web browser.

3.1 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 Thales. 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.

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

3.2 Sentinel LDK Installation

3.2.1 Windows

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

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

3.2.2 Linux

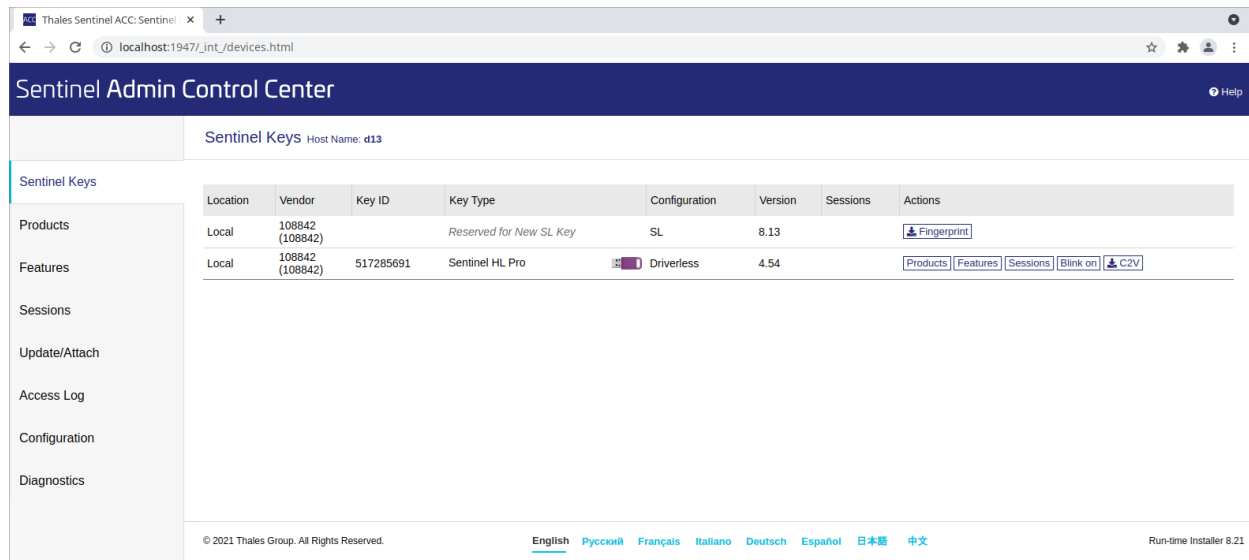
Follow these steps to install Sentinel License Manager on a Linux machine:

- Start the command line and navigate to the **[sdk]/hasp/** directory.
- On 64-bit Linux distributions, install the **32-bit** compatibility binaries.
 - On Ubuntu 18.04 and higher: Execute “**sudo apt-get install libc6:i386**”.
- Execute “**sudo ./dunst**” to uninstall any previous versions of Sentinel License Manager.
- Execute “**sudo ./dinst**” to install Sentinel License Manager.
 - Without compatibility binaries, error “*No such file or directory.*” might appear.

3.3 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 the image below. The client must check that the trial licenses were installed properly, and that the Anonymzier 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 a licensing failure and additional costs to relicense the software to another machine. The web page lists all available license keys. Under the “**Products**” link

in the left pane all available products are listed.



Sentinel License Manager screenshot.

3.4 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, 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 of the Windows firewall, and also to the antivirus exception list, if it uses its own firewall.

3.5 Managing Licenses

It is of the utmost importance that the client understands the licensing schemes used in the Thales Sentinel LDK software protection framework. Otherwise, unreparable 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 *Anonymzier Licensing*.

3.6 License Error Codes

Error codes are outputted to the error stream of the application (typically *stderr*) using Anonymzier. The user needs to check the error stream for 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 Anonymzier 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 Anonymzier is encrypted for enhanced software protection. However, in case of failure, the application does not terminate, but crashes after a few calls to the library; this is a security measure against reverse engineering but may confuse the users. The client needs to make sure they monitor the error codes outputted by the error stream to distinguish between programming errors and licensing problems.

3.7 TensorRT

The Anonymzier can use TensorRT to run detection and OCR models, SDK package contains data files and command-line utility which can be used to generate TensorRT model for specific target device.

3.7.1 TensorRT Anonymzier Models

For Nvidia Jetson devices, when TensorRT GPU mode is set, the classifiers cannot be prepared in advance and the folder

```
[AnonymizerSDK]/sdk/models
```

does not include prebuilt .dat files, but only their prototypes. Before running the software for the first time on a specific Nvidia Jetson device type, the .dat files must be created using an utility called **edftrt_dat_encoder** which should be located in the **tools** directory. For example, if the client has 100 identical devices, they only need to follow this process once and then share the created .dat files among the devices.

To run the **edftrt_dat_encoder** utility, the client needs to make sure the relevant Nvidia TensorRT libraries are visible in the system, which can be checked using **ldd** utility as "**ldd edftrt_dat_encoder**". If not found, the Nvidia TensorRT need to be added to the library path using the following command:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/aarch64-linux-gnu/
```

The **edftrt_dat_encoder** utility must be executed when there is no other process utilizing resources on the target device, otherwise the created .dat files will not give the best possible performance. By default, the generated .dat files use float32 (FP32) computation mode. Using float16 (FP16) computation mode can improve speed, but the effect on accuracy needs to be verified. Use parameter "-h" with the **edftrt_dat_encoder** utility to see all options, run the utility without any options to use defaults. Conversion can take about 30 minutes depending on the specific device type. Warnings might appear during the generation which can be ignored.

3.7.2 Generating Device Specific Models

Here is an example of a command that can be used from inside the models directory:

```
./edftrt_dat_encoder -p=./ -w=2048 -q=FP16
```

The “-p” argument denotes the path in which the utility will look for model prototypes (file triples with extensions .dat.pre, .dat.net, .dat.post) to make optimized .dat files from, “-q” sets the quantization, and “-w” sets the workspace size - see the official NVIDIA TensorRT documentation (docs.nvidia.com/deeplearning/tensorrt/api/c_api/) for the function `IBuilderConfig::setMaxWorkspaceSize` for more information about this parameter.

3.7.3 Known Issues

As of Nvidia TensorRT 8.2, there are still documented known issues in Nvidia TensorRT library that can cause the generated .dat files to lose accuracy or completely misbehave. It is up to the customer to verify the newly created .dat files give expected performance, for example by comparing with the results of Anonymzier CPU version.

3.8 OpenGL Prerequisites

For Nvidia Jetson devices, we also provide Anonymzier with Tensorflow Lite backend, which utilizes OpenGL for GPU processing. To be able to use GPU, the Jetson SD card image must be installed with **nvidia-l4t-3d-core** package, described as “*NVIDIA GL EGL Package*”. This package is installed during the default installation of Nvidia JetPack. When using a remote shell to connect to a device where the client wants to be using OpenGL GPU mode, X forwarding must be turned off.



4 SDK Application Interface

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

4.1 Structures

Document section Structures covers all the information about structures used in the SDK's public application interface. Structure *ERoI* 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.

ANState

```
typedef void* ANState;
```

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

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.

AnDetectionParams

```
typedef struct {  
    float threshold;  
    int    min_length;  
    int    max_length;  
    float blur_size_factor;  
    int    blur_num_passes;  
} AnDetectionParams;
```

AnDetectionParams are control parameters to select/filter objects to be anonymized. Anonymization area and anonymization degree is controlled by *blur_size_factor* and *blur_num_passes*. The structure contains the following fields:

- **threshold**
Detection threshold to balance between recognition of objects and amount of false positive detections. Default value is 0.0
- **min_length**
minimal length of the detected object to be accepted in pixels. Default value is 0
- **max_length**
maximal length of the detected object to be accepted in pixels. Default value is INT_MAX
- **blur_size_factor**
Parameter to enlarge anonymized area (< 1.0 - smaller than detection, ==1.0 - exactly as detection;

>1.0 - larger than detection). Default value is 1.0

- **blur_num_passes**
Number of blur passes over the detection. Default value is 1

AnObjectType

```
enum AnObjectType{
    AN_FACE = 0,
    AN_LP,
    AN_NUM_OBJECT_TYPES
};
```

AnObjectType is an enumeration of objects to anonymize. It is used for setting and accessing anonymization parameters of objects (see *AnDetectionParams*) in the anonymization parameters *AnParams*.

AnParams

```
typedef struct
{
    ERROI          roi;
    int            mode;
    int            show;
    int            jpeg_quality;
    int            keep_exif;
    int            panoramic;
    int            verbose;
    int            num_object_types;
    AnDetectionParams object_params[AN_NUM_OBJECT_TYPES];
} AnParams;
```

AnParams is a set of parameters controlling anonymization process. *AnParams* enables to define parameters in runtime like ROI definition, anonymized object type, output image quality etc. For visualisation purposes the *show* flag is used. Detections are then highlighted instead of blurred. The structure contains the following fields:

- **roi**
Region of interest - the area scanned for detections, default area is full image (defined as *ERoI*{*x=0, .y=0, .width=-1, .height=-1*})
- **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).
- **show**
Debugging flag indicating, that detections will be highlighted instead of blurred.
- **jpeg_quality**
Output image JPEG quality 0..100, default value is 90.
- **keep_exif**
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.
- **verbose**
Enables logging of anonymization progress on to stdout. Default value is 0
- **num_object_types**

Arrat size of object_params see *AnObjectType*

- **object_params**

Array of anonymization parameters indexed using *AnObjectType*, see *AnDetectionParams*



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



Comparison of different blur strength



4.2 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.

4.2.1 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* type.

Returns:

- **0** on success, **non-zero** value otherwise

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);
```

Input:

- **state**
Pointer to the initialized Anonymizer engine instance created by *anInit()* function.

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);
```

Input:

- **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
```

4.2.2 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. Anonymizer 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** on success, **error code** otherwise.

Description:

The function *anAnonymize()* loads image file specified by *src_image_filename* parameter, run anonymization with parameters specified by *AnParams* params 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 src_buffer, unsigned int width, unsigned int height,
                          AnParams* params, ANState state, AnBuffer*
                          dst_buffer);
```

Inputs:

- **src_buffer**
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.

Output:

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



Returns:

- **0** on success, **error code** otherwise.

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 *dst_buffer* must be freed when it is no longer needed. See *Anonymizer Example – buffers* for more information.

Example:

```
AnBuffer src_buffer, dst_buffer;
// input and output buffers
// fill src_buffer with some image data
if (fcnAnAnonymizeImageBuffer(src_buffer, img_width, img_height, NULL, state, &dst_buffer) != 0)
{
    // error handling
}
```

IMPORTANT: Always free *dst_buffer* 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 src_buffer, AnParams* params, ANState state, AnBuffer*
dst_buffer);
```

Inputs:

- **src_buffer**
Input structure with JPEG encoded data buffer.
- **params**
Pointer to *AnParams* structure with Anonymization parameters. Use NULL for default parameters.
- **state**
ANState Anonymizer state.

Output:

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

Returns:

- **0** on success, **error code** otherwise.

Description:

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

Example:

```
AnBuffer src_buffer, dst_buffer;
// input and output buffers
// fill src_buffer with some image data
if (ern=anAnonymizeJpegBuffer(src_buffer, NULL, state, &dst_buffer) != 0)
{
    // error handling
}
```


IMPORTANT: Always free `dst_buffer` 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);
```

Input:

- **buffer**
AnBuffer structure to free.

Description:

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

Example:

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

anGetDefaultParams

Fills *AnParams* structure with default values.

Specification:

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

Output:

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

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);
```

4.2.3 Other functions

This part defines other API functions. Function `anGetErrorMsg()` is used to get error message for error code. Function `anPlotLayouts()` serves as helping function to for better setup of detection parameters. 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 errn);
```



Input:

- **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
}
```

anPlotLayouts

Save image(s) with current tiles' layout(s) for detection of given type.

Specification:

```
int anPlotLayouts(ANState state, const char* image_path, AnObjectType type);
```

Input:

- **state**
ANState Anonymizer state.
- **image_path**
The path to the image, the tiles will be plotted on.
- **type**
AnObjectType object detector type, whose detection layouts will be plotted.

Returns:

- **0** on success, **error code** otherwise.

Description:

Object detection using convolutional neural networks is limited by the architecture of the network (by the fixed input image size to the network). To process large images (e.g. street-view images) the image should be splitted to "tiles" which are processed by the network independently. How the tiles are arranged is defined by layouts. Layouts are generated based on parameters or manually defined in .ini files of detectors. The function *anPlotLayouts()* serves for better setup of layouts/layouts' parameters. The function plots tiles to the input image as colored rectangles. The images with tiles are saved to the current working directory with name prefix defined by *AnObjectType* and by the layout setup.

Example:

```
// given initialized ANState state and path to image, the function will generate image(s) with
// tile layouts for face detection.
anPlotLayouts(state, image_path, AnObjectType::AN_FACE);
```



5 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.

5.1 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 project is included, in case of Linux Makefile is included.

5.1.1 Initialization of Anonymizer engine

First thing to do is the Anonymizer engine initialization using the `anInit()` function. Parameters of this function are directory where Anonymizer SDK 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.

5.1.2 Setting anonymization parameters

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

```
AnParams parameters;

parameters.roi = ERRoI{0,0,-1,-1}; /* Region Of Interest (detection area) see ERRoI */
parameters.mode = ANONYMIZE_FACE | ANONYMIZE_LP; /* anonymize faces and licence plates */
parameters.jpeg_quality = 90; /* output JPEG file/buffer quality */
parameters.show = 0; /* if show>0 than the detections in ouput image are highlighted insted
of blurred */
parameters.panoramic = 0; /* set panoramic=1 if you have 360° panoramatic images to detect
objects on left and right edges */
parameters.keep_exif = 0; /* set keep_exif=1 to copy EXIF of processed JPG image NOTE that
thumbnail is not modified and that nonstandard EXIFs are not preserved. Works only for JPG
files or JPG buffers. */
parameters.verbose = 0; /* if verbose>0 than Anonymizer engine prints to the stdout the
progress state */

/* The face detection and blurr parameters */
```

```

parameters.face.threshold = 0.; /* detection threshold to balance between recognition of
    objects and amount of false positive detections */
parameters.face.min_length = 0; /* minimal length of the detected object to be accepted
    in pixels */
parameters.face.max_length = INT_MAX; /* maximal length of the detected object to be accepted
    in pixels */
parameters.face.blur_size_factor = 1.; /* factor to enlarge anonymized area with respect to size
    of detection */
parameters.face.blur_num_passes = 1; /* number of blurr passes over anonymized area */

/* The license plate detection and blurr parameters */
parameters.lp.threshold = 0.;
parameters.lp.min_length = 0;
parameters.lp.max_length = INT_MAX;
parameters.lp.blur_size_factor = 1.;
parameters.lp.blur_num_passes = 1;

```

The parameter `.roi` defines Region Of Interest where the anonymization is applied.

Parameters `parameter.face.threshold` and `parameter.lp.threshold` define detection thresholds for face 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 value range is $\langle 0,1 \rangle$.

Parameters `parameter.face.blur_size_factor` and `parameter.lp.blur_size_factor` 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 `parameter.face.blur_num_passes` and `parameter.lp.blur_num_passes` 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 parameter `.show` flag is used highlighted detections for testing purposes (see table 1 in description of *AnParams* structure).

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

The parameter `.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 parameter `.keep_exif` 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.

5.1.3 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
}

```

5.1.4 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
```

5.2 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 project is included, in case of Linux Makefile is included. version.

5.2.1 Initialization of Anonymizer Engine

5.2.2 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 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);
```

5.2.3 Image buffer anonymization

To anonymize buffer with raw BGR image, function `anAnonymizeJpegBuffer()` 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.

IMPORTANT: The example implicitly suppose, that returned opencv image has continuous image data buffer! This should be checked with `cv::Mat::isContinuous()` function.

```
// 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 - output 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
```

5.2.4 *Cleaning up*



6 Anonymizer command line interface

Anonymizer SDK package contains anonymizer-cli application for batch anonymization of images located at `[Anonymizer_SDK]/applications/anonymizer-cli`. 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-cli:

Linux:

```
./anonymizer-cli [options] image_list
```

Windows:

```
anonymizer-cli.exe [options] image_list
```

Options:

```
-h, --help      this help
-o=directory, --output=directory
                 sets the output directory for anonymized images (default 'anonymized')
-q=n, --quality=n
                 sets jpeg quality n = 1..100 (0 default)
--keep-exif
                 Copy EXIF from original JPG file to output file.
                 NOTE that EXIF thumbnail is not anonymized and that some nonstandard EXIFs may not be
                 preserved.
                 You may regenerate JPG thumbnails with some suitable tool after anonymization (jhead,
                 exiftran).
--no-lp         don not run LP anonymization
--no-face       don not run FACE anonymization
--cpu           override coputation mode to cpu
--gpu          override computation mode to gpu
--gpu-id=ID    use gpu with given ID for gpu computation
--num-threads=X number of threads for CPU computation parts
--roi=[x,y,width,height]
                 anonymization ROI (region of interest) negative value of width/height represents full
                 dimension of image
                 default, [0 0 -1 -1]
--show         colorize detections insted of blurring them
--panoramic
                 copy left and right edges of panoramatic images to be able to detect on edges of 360
                 degree images
                 not active with ROI which is not full width of image
-v, --verbose
                 more verbose run.

--lp-thr=threshold
                 sets the threshold on LP detections (default 1.000000)
--lp-min-length=value
                 sets the length filter, the minimal length of the lp to be accepted
                 (default 832)
--lp-max-length=value
                 sets the length filter, the maximal length of the lp to be accepted
                 (default 0)
--lp-num-blur-pass=n
                 sets licence plates blur amount (default 832)
--lp-size-factor=r
                 sets size of blurred area relative to lps detection size (default
                 0.000000)

--face-thr=threshold
                 sets the threshold on face detections (default nan)
--face-min-length=value
                 sets the length filter, the minimal size of the face to be accepted
                 (default 1065353216)
--face-max-length=value
                 sets the length filter, the maximal size of the face to be accepted
                 (default 1)
--face-num-blur-pass=n
                 sets faces blur amount (default 0)
```

```
--face-size-factor=r    sets size of blurred area relative to faces detection size (default
                        0.000000)

--plot-layouts
    plot colored tiles into image based on layouts used detected object. For each object
    used layouts are plotted.
    Expected image filename as an input!
```

Examples (Linux):

```
./anonymizer-cli -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-cli -o=o_dir -face --face-thr=0.8 image_list.txt
```

This command will anonymize only faces with score above 0.8 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-cli**.

Example of image list can be:

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

```
./anonymizer-cli --plot-layouts DSC_5243.jpg
```

This command will produce images named by anonymized object and currently used layouts to generate detector's input tiles. The tiles are displayed in the input image (DSC_5243) as colored rectangles.











7 Anonymzier Licensing

Anonymzier uses the third-party framework developed by Thales for software protection and licensing. The SDK is protected against reverse engineering and unlicensed execution using hardware USB keys. The SDK can not be used without a USB license key with a valid license except in trial version, which uses software key instead.

7.1 License Key Types

The SDK allows loading a license using various hardware key types which are listed in the following table. The keys differ by the number of licenses they can contain (Pro and Max versions), by physical dimensions, ability to contain time-limited licenses (Time versions) and ability to distribute 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 

7.2 Licenses Overview

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

7.2.1 Perpetual License

A perpetual license is the least restrictive license available. It allows the user to use the license in specified number of instances for unlimited time and unlimited number of executions. This license type is used for products which will be deployed to the end-user.

7.2.2 Time-Limited License

A time-limited license allows to set a restriction on the time for which the license is valid. The license validity end date or the number of the days for which the license is valid after the first use 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.

7.2.3 Execution Counting

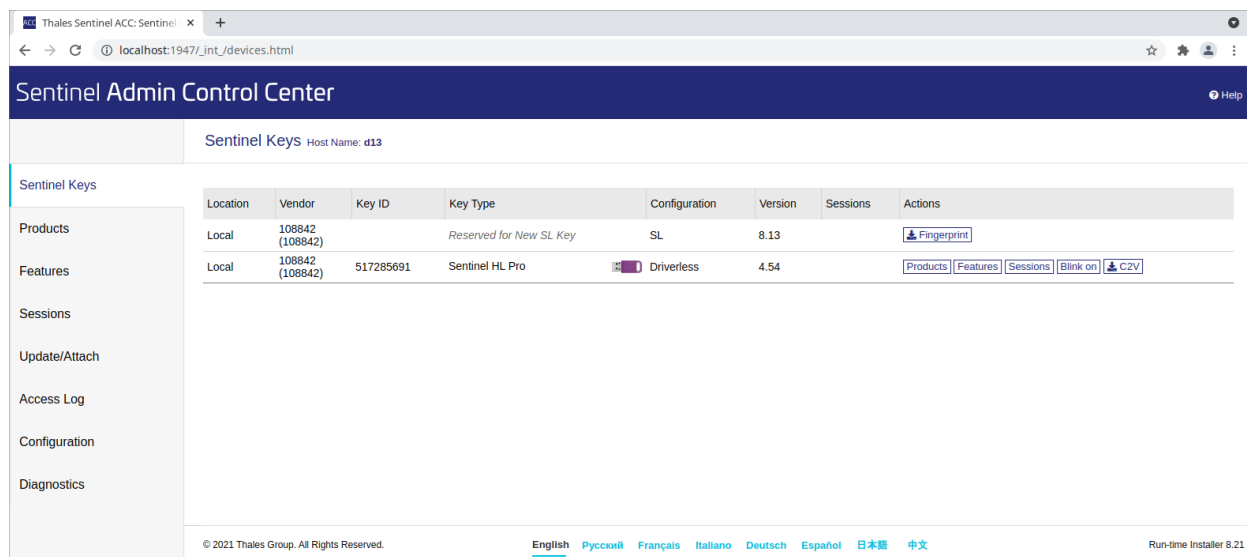
An execution counting license allows counting the number of times the license was logged in. The SDK is designed in such a way that it logs in the license every time a specified SDK function is called. It allows limiting the number of executions with the license. This type of license is used mainly in the Developer package.

7.3 License Management

The license protection software provides a 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.

7.3.1 Connected License Keys

The list of license keys currently plugged in the computer is available at http://localhost:1947/_int/_devices.html. The list contains basic information about each key, including the 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. For each key, it is possible to list the contained license products, features and sessions using the buttons Products, Features and Sessions. For easy identification, the USB key LED can be blinked using the **Blink On** button in the Actions column. The unique key identification file can be downloaded using the **C2V** button.



The screenshot shows the Sentinel Admin Control Center web interface. The main content area displays a table titled "Sentinel Keys" with the following data:

Location	Vendor	Key ID	Key Type	Configuration	Version	Sessions	Actions
Local	108842 (108842)		Reserved for New SL Key	SL	8.13		Fingerprint
Local	108842 (108842)	517285691	Sentinel HL Pro	Driverless	4.54		Products Features Sessions Blink on C2V

The interface also includes a sidebar with navigation options: Sentinel Keys, Products, Features, Sessions, Update/Attach, Access Log, Configuration, and Diagnostics. The footer contains copyright information for Thales Group and a language selector.

Web interface with list of plugged keys on http://localhost:1947/_int/_devices.html

7.3.2 License Key Details

Detailed information about a key can be acquired by clicking on the **Features** button in the *Connected License Keys* list or at 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 all the features represents the whole license. Each Feature controls a different part of the SDK workflow (initialization, binary model selection, descriptor computation, ...).

The screenshot shows the Sentinel Admin Control Center interface. The main content area displays a table titled "Features Available" for Host Name: d13. The table is filtered by Key: 517285691 and Vendor: 108842. The table has columns for Product, Feature, Location, Access, Counting, Logins, Limit, Detached, Restrictions, Sessions, and Actions. The table lists 12 features, each with a unique ID and associated product information. The "Sessions" column for all features is set to "Perpetual".

Product	Feature	Location	Access	Counting	Logins	Limit	Detached	Restrictions	Sessions	Actions
0		Local	Loc	Station		∞		Perpetual		Sessions
108842 Product 4	1001	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 4	1000	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	10003	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	10000	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	425	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	423	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	417	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	416	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	413	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	412	Local	Loc Display	Station		∞		Perpetual		Sessions
108842 Product 6	301	Local	Loc Display	Station		∞		Perpetual		Sessions

1-12 of 28 | Next Last

© 2021 Thales Group. All Rights Reserved. English Русский Français Italiano Deutsch Español 日本語 中文 Run-time Installer 8.21

Web interface with key 517285691

details on http://localhost:1947/_int_/features.html?haspid=517285691

7.4 License Update

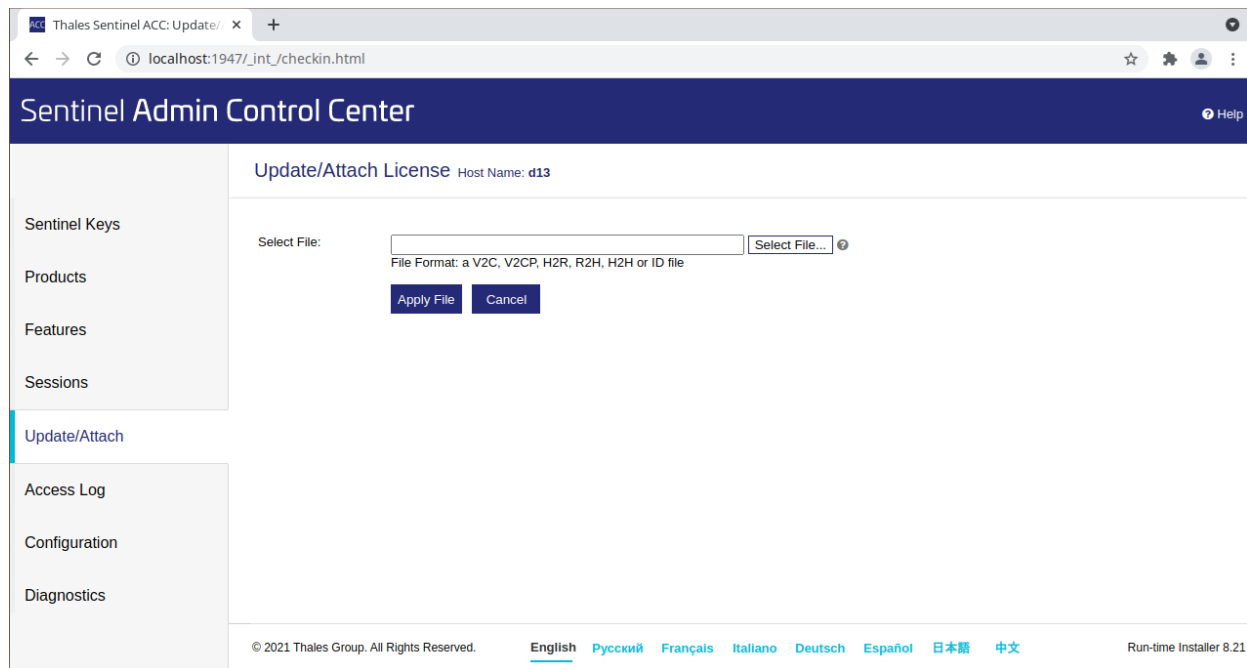
The license can be updated using a special *.v2c file, which is emitted by the licensor of the software. The license update file is generated for a specific license key ID and only that key can be updated using the file. There are two ways of updating the license: *Web Interface* and *Command Line*.

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: The hardware protection key dongle with the license to be updated needs to be connected to the machine where the license update will be applied.

7.4.1 Web Interface

The first option allows the 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.



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 the Select File button and select the *.v2c file which you want to use for the update.
3. Click on the Apply File button.
4. A webpage with the result of the license update is shown.

7.4.2 Command Line

The second method of updating the license is by using the Windows command line or a Linux console. This approach can be very useful when applying the update remotely or on many devices. It is also suitable for automating the license update procedure. This option requires basic knowledge of the Windows command line or some 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 the *.v2c file path on the selected machine:

```
hasp_update u /path/to/v2c/license.v2c
```

If the command runs without any errors, the license has been updated successfully.

Linux console

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

```
./hasp_update u /path/to/v2c/license.v2c
```

If the command runs without any errors, the license has been updated successfully.

8 Third Party Software

The LPM SDK uses third party software libraries in accordance with their licenses. The licenses can be found under [\[LPMSDK\]/documentation/3rdparty-licenses](#).

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

- Iniparser
- OpenCL
- OpenCV
- OpenSSL
- TensorFlow Lite
- 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/>)."

