Eyedea LPM SDK

Developer's guide   Version 7.7

eyedea
RECOGNITION

ADVANCED COMPUTER VISION SOLUTIONS

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

The LPM SDK is a cross-platform software library designed to provide a comfortable detection of car license plates, ADR and Trash plates, and/or cars via bounding boxes, as well as optical character recognition (OCR) of plates including plate type and physical size recognition from input images. It defines an interface between the client's software and our state-of-the-art detection and recognition modules. This special API allows simple module administrations and updates without any need for changes to the client's software.

Each client receives an FTP account automatically created at Eyedea Recognition's server. This FTP access serves as two-way communication between the client and Eyedea Recognition, s. r. o.. Clients have an easy way to regularly upload data samples (or problematic data) to the FTP server, and subsequently receive the corresponding updates of LPM modules. This systematic approach makes it possible to verify result statistics and continuously adapt the LPM modules to the client's specific data, ensuring the best possible performance.

1.1 Technical Details

License plate

Multi-line license plates

Car bounding box

ADR plate

Trash plate
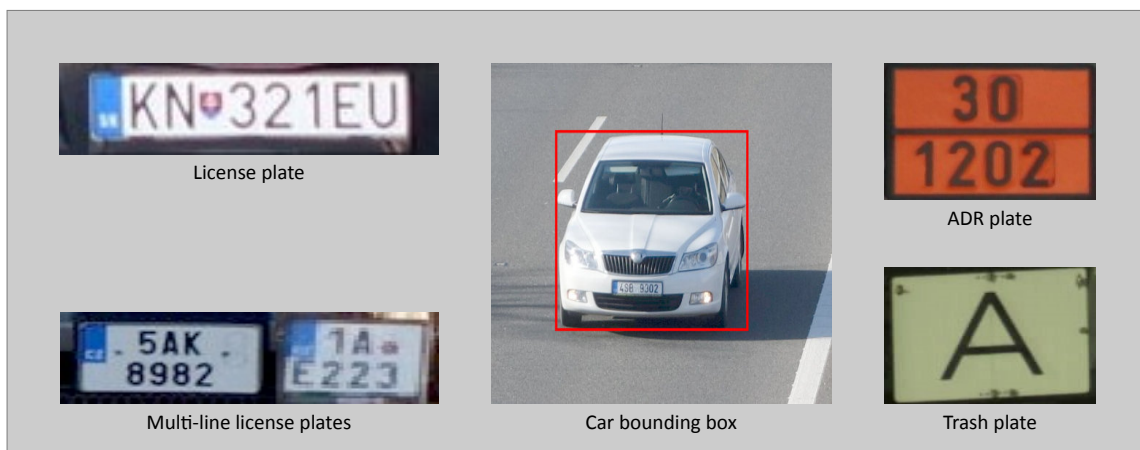
LPM SDK consists of two parts – base LPM engine and detection/recognition modules. Both are cross platform libraries with C interface. The base LPM library is the only entry point, the user never uses the detection/recognition LPM modules directly. The module is loaded, configured, and executed using the LPM library. Each module can contain a detection routine, an OCR routine, or both.

User's Code
C/C++

C native API

LPM Library

LPM module   LPM module   LPM module
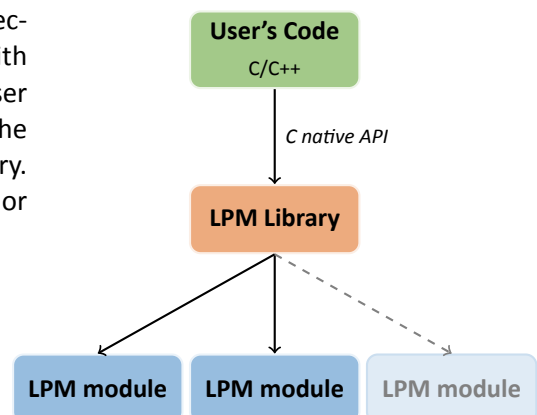
The LPM library provides the following APIs:

- C native API
- Python wrapper
- Java wrapper

Officially supported operating systems and platforms:

- Windows 7, 8, 8.1, 10 and 11
  32-bit and 64-bit (Visual Studio 2022)
- Ubuntu 18.04 and higher
  64-bit and aarch64
- Other platforms on request

1.2 System Workflow

The workflow of the LPM system consists of: image acquisition, plates or bounding boxes detection, and OCR of detected plates (where applicable). The image acquisition is not part of this SDK and must be solved separately.

The process starts with detection of license plates or ADR/Trash plates or car bounding boxes. Some types of detections can then be supplied to the OCR stage which returns hypotheses of the plate text and plate type, together with their confidences. There is no need to crop the detected plates for the OCR stage, as the OCR stage takes the whole input image and the detection results.

1) Image acquisition
Not part of the SDK
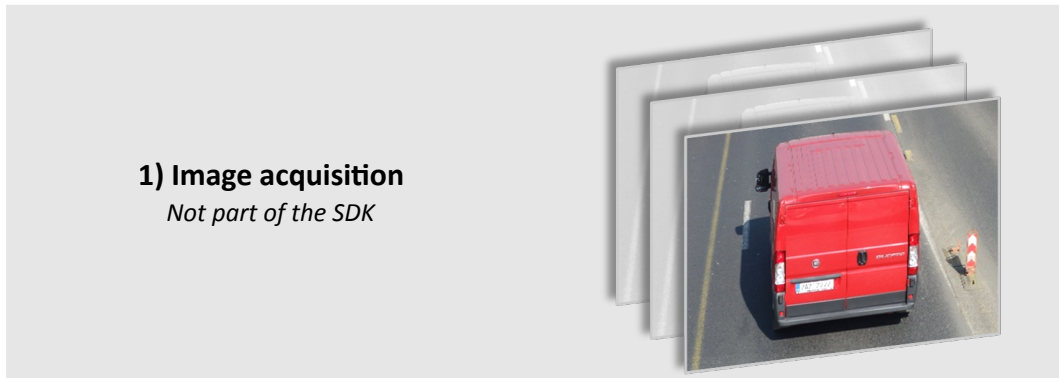
For every single input image:

2) Plates and objects detection
Detect plates and/or other objects in the input image

3) Plates OCR
Plate text and type recognition for detected license plates or ADR tables

| TEXT: 2A22222 |
| SIZE: 520x110mm |

| **TEXT:** 2A22222 |
| **TYPE:** CZ |
| **SIZE:** 520x110mm |

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.

📂 LPM SDK . . . . *distribution main folder*

    📂 LPM . . . . *LPM engine folder*

        🗀 include . . . . *LPM header files*

        🗀 lib . . . . *LPM libraries*

    📂 examples . . . . *LPM examples folder*

        📂 example-anpr-implink . . . . *example of implicit LPM library link*

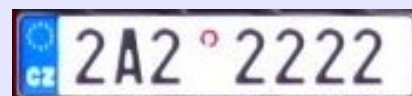   🗀 images . . . . *example images folder*

            📄 example-anpr-implink.vcxproj . . . . *Visual Studio project (only Windows version)*

            📄 example.cpp . . . . *example source code*

            📄 Makefile . . . . *example makefile (only Linux version)*

    📂 modules-v7 . . . . *LPM modules*

        🗀 x64 . . . . *modules for appropriate architecture*

        📄 config_camera_view.ini . . . . *camera view parameters file*

  🗀 hasp . . . . *license management software folder*

  🗀 documentation . . . . *SDK documentation folder*

  🗀 wrappers . . . . *SDK wrappers folder*

  📄 LICENSE.txt . . . . *SDK license*

  📄 WhatsNew.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 LPM SDK. 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 LPM SDK 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 **[LPM_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 **[LPM_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 LPM 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 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.
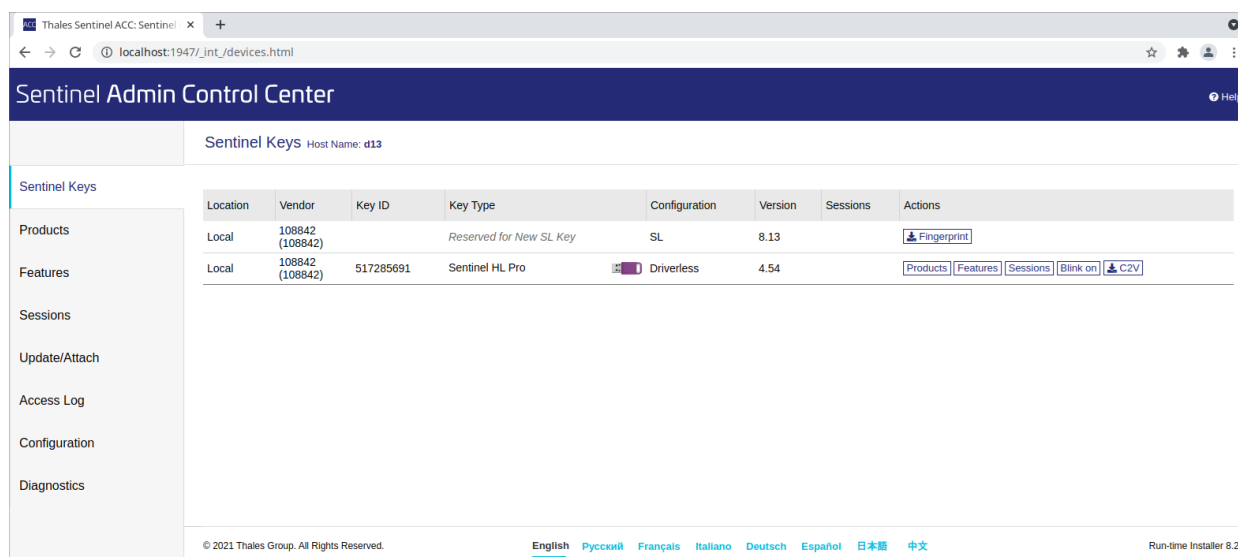
Sentinel License Manager screenshot.

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

$$C:\backslash Windows\backslash system32\backslash hasplms.exe$$

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, 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 *LPM SDK Licensing*.

3.6   License Error Codes

Error codes are outputted to the error stream of the application (typically *stderr*) using LPM SDK. 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 LPM 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 LPM SDK 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 LPM SDK 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 LPM SDK Models

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

```
[LPM_SDK]/modules-v*/aarch64/module-name/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 **models** 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 LPM SDK CPU version.

3.8   OpenGL Prerequisites

For Nvidia Jetson devices, we also provide LPM SDK 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 *Enumerators*, *Structures* and all available *Functions*. It gives the developer a detailed overview of the SDK and can help orientate the developer during SDK integration.

For information about image structure and image manipulation functions used by SDK see chapter *ERImage*.

4.1   Enumerators

This section defines the API enumerators which are used in the LPM SDK:

LpmViewType

LpmViewType is used to specify the type of the camera view in the *LpmCameraViewParams* structure.

- **LPM_VIEW_FRONTAL = 0**
  Frontal images of cars (e. g. overhead installation on motorway gantries).
- **LPM_VIEW_GENERIC = 1**
  Generic images of cars (e. g. camera in a moving vehicle).

LpmDetectionLabel

LpmDetectionLabel is used to specify the type of detection input for the *LpmRunOcr*() function.

- **LPM_LABEL_DEFAULT = 0**
  Default label value for generic usage.
- **LPM_LABEL_PERSON = 200**
  Generic person object.
- **LPM_LABEL_LP = 1000**
  Generic license plate.
- **LPM_LABEL_LP_EU_ONE_LINE = 1001**
  European license plate.
- **LPM_LABEL_LP_EU_MULTI_LINE = 1002**
  European multiline license plate.
- **LPM_LABEL_LP_NORTH_AMERICA = 1200**
  North American license plate.
- **LPM_LABEL_LP_ASIA_PACIFIC = 1300**
  Asian license plate.
- **LPM_LABEL_ADR = 2000**
  ADR (the European Agreement on International Carriage of Dangerous Goods by Road).
- **LPM_LABEL_ADR_STRING = 2001**
  ADR with text.

- **LPM_LABEL_ADR_EMPTY = 2002**
  Empty ADR.
- **LPM_LABEL_TRASH = 2100**
  Plate indicating trash load.
- **LPM_LABEL_SPEED_LIMIT = 2200**
  Speed limit sticker.
- **LPM_LABEL_OVERSIZE_LOAD = 2210**
  Oversize load sign e. g. "CONVOI EXCEPTIONNEL".
- **LPM_LABEL_VIGNETTE = 2300**
  Vignette sticker.
- **LPM_LABEL_VEHICLE = 3000**
  General vehicle bounding box
- **LPM_LABEL_VEHICLE_FRONT = 3001**
  Frontal vehicle bouding box
- **LPM_LABEL_VEHICLE_REAR = 3002**
  Rear vehicle bounding box
- **LPM_LABEL_VEHICLE_WINDSHIELD = 3010**
  Vehicle windshield
- **LPM_LABEL_VEHICLE_WHEEL = 3020**
  Vehicle wheel.

4.2   Structures

This section covers information about the structures used in the SDK's public application interface. *LpmModuleInfo*, *LpmPropertyFlags*, *LpmLicenseInfo*, and *LpmDateTime* structures are used to get information about available modules and their properties, *LpmCameraViewParams* contains information about the camera view and is used during loading of modules, *LpmBoundingBox* is used to store different bounding boxes coordinates, *LpmDetResult*, *LpmDetection*, *LpmOcrHypothesis*, *LpmLpDimensions* and *LpmTextLine* store all data about detections or OCR results, *LpmModuleConfig* is used for configuration of loaded modules. In version 7.3 and higher, structures *LpmDetResult_extension1*, *LpmDetection_extension1*, and *LpmModuleConfig_extension1* are used to store additional information.

LpmModuleInfo

```
typedef struct {
    char            name[LPM_MAX_STR_LEN];
    int             id;
    char            date[LPM_MAX_STR_LEN];
    char            path[LPM_MAX_PATH_LEN];
    int             version;
    int             subversion;
    char            det_type[LPM_MAX_STR_LEN];
    char            obj_type[LPM_MAX_STR_LEN];
    char            rcg_type[LPM_MAX_STR_LEN];
    char            input_img_type[LPM_MAX_STR_LEN];
    double          pxl_aspect_ratio;
    char            lp_countries[LPM_MAX_STR_LEN];
    int             lp_min_mean_max_width[3];
    int             lp_min_mean_max_height[3];
    double          lp_min_mean_max_rotation[3];
    int             is_active;
    LpmPropertyFlags  prop;
    LpmLicenseInfo   *license_info;
} LpmModuleInfo;
```

LpmModuleInfo represents all information you can get about a module by using the *lpmGetModuleInfo* function. The structure contains the following fields:

- **name**
  Full name of the module.
- **id**
  Id of the module.
- **date**
  Release date of the module in "*YYYY-mm-dd*" format.
- **path**
  Full path to the module.
- **version**
  Version number of the module.
- **subversion**
  Subversion number of the module.
- **det_type**
  Char array with the detector type ("*frontal*", "*generic*", "*lfrontal*").
- **obj_type**
  Char array with the type of the detected object ("*license plates*", "*adr plates*", ...).
- **rcg_type**
  Char array with the recognition type ("*ceu3*", "*cz*", "*adr*", "*vcl*", ...).
- **input_img_type**
  Input image type (e.g. "*ERImage*").
- **pxl_aspect_ratio**
  Desired pixels aspect ratio of input images.
- **lp_countries**
  Supported LP countries codes returned as a comma separated list (e.g. "*CZ,SK,A*").
- **lp_min_mean_max_width**
  Required LP width range.
- **lp_min_mean_max_height**
  Required LP height range.
- **lp_min_mean_max_rotation**
  Range of LP inplane rotation.
- **is_active**
  Switch indicating whether the module is active or not.
- **prop**
  Module properties in the *LpmPropertyFlags* structure. See header file **lpm_types.h** for more information.
- **license_info**
  Information about the license in the *LpmLicenseInfo* struct.

LpmPropertyFlags

```
typedef long long LpmPropertyFlags;
```

LpmPropertyFlags contains bit flags which describe the properties of the module, detector type, object type, OCR type and recognition type. For specific values, see the **lpm_types.h** header file.

LpmLicenseInfo

```
typedef struct {
    int            is_valid;
    LpmDateTime    expiration_date;
    int            is_using_counter;
    unsigned long  executions_left;
} LpmLicenseInfo;
```

LpmLicenseInfo contains information about the module license. The structure contains the following fields:

- **is_valid**
  Flag determining whether the license is valid or not. Zero means invalid, otherwise valid.
- **expiration_date**
  License expiration date.
  Note: the license is time-unlimited if all fields of LpmDateTime structure are zeros.
- **is_using_counter**
  The counter is enabled if non-zero.
- **executions_left**
  Number of module executions left. License is execution-unlimited if *is_using_counter* is zero.

LpmDateTime

```
typedef struct {
    unsigned int   year;
    unsigned char  month;
    unsigned char  day_of_month;
    unsigned char  hour;
    unsigned char  minute;
    unsigned char  second;
} LpmDateTime;
```

LpmDateTime contains date and time fields. The structure contains the following fields:

- **year**
  Year in 4-digit format.
- **month**
  Month 1-12.
- **day_of_month**
  Day 1-31.

- **hour**
  Hour 0-23.
- **minute**
  Minute 0-59.
- **second**
  Second 0-59.

LpmCameraViewParams

```
typedef struct {
    LpmViewType    view_type;
    unsigned int   min_horizontal_resolution;
    unsigned int   max_horizontal_resolution;
    float          density_ratio;
} LpmCameraViewParams;
```

LpmCameraViewParams contains the configuration of camera view parameters which are sent to every module during module initialization using *lpmLoadModule*(). The structure contains the following fields:

- **view_type**
  LpmViewType with values LPM_VIEW_FRONTAL or LPM_VIEW_GENERIC.
- **min_horizontal_resolution**
  Minimal horizontal resolution in number of pixels per meter.
- **max_horizontal_resolution**
  Maximal horizontal resolution in number of pixels per meter.
- **density_ratio**
  Camera pixel density ratio which is defined as vertical pixel density / horizontal pixel density. For standard cameras with square pixels, use value 1.

> **IMPORTANT**: Camera view parameters do NOT apply when using GPU capable detection modules which work in a different way than previous modules.

LpmModuleConfig

```
typedef struct {
    int                         compute_on_gpu;
    int                         gpu_device_id;
    LpmModuleConfig_extension1  *extras;
} LpmModuleConfig;
```

LpmModuleConfig contains the parameters for module initialization which are sent to every module during module initialization using *lpmLoadModule*(). Note that in version 7.3 some fields were deprecated, and a new extension structure was introduced.

The structure contains the following fields:

- **compute_on_gpu**
  DEPRECATED specifies whether the computation should be done on a CPU=0 or a GPU=1.
- **gpu_device_id**
  DEPRECATED GPU device identifier (used only when the computation is running on a GPU).
- **extras**
  Extension of the configuration structure, **must be NULL if not in use**. Used in version 7.3 and higher.

LpmModuleConfig_extension1

```
typedef struct {
    const char  *lpm_config_filename;
    int          ocr_compute_on_gpu;
    int          ocr_gpu_device_id;
    int          ocr_num_threads;
    int          disable_ocr;
    const char  *det_config_filename;
    int          det_compute_on_gpu;
    int          det_gpu_device_id;
    int          det_num_threads;
    int          disable_det;
    void        *extras;
} LpmModuleConfig_extension1;
```

Extension of the configuration structure for module initialization used in version 7.3 and higher. Structure contains the following fields:

- **lpm_config_filename**
  Filename of the module's configuration file (config.ini by default if NULL).
- **ocr_compute_on_gpu**
  Specifies if the OCR computation should be done on a CPU=0 or a GPU=1.

- **ocr_gpu_device_id**
  GPU device identifier (used only when the computation is running on a GPU) for the OCR.
- **ocr_num_threads**
  Specifies the number of threads available for the OCR (used only when the computation is running on a CPU). Uses approximately 90% of logical processors if set to 0 or negative.
- **disable_ocr**
  If set to 1, the OCR submodule will not be loaded and will not be available.
- **det_config_filename**
  Filename of the detector's configuration file (config-det.ini by default if NULL).
- **det_compute_on_gpu**
  Specifies if the computation should be done on a CPU=0 or a GPU=1 for the detector.
- **det_gpu_device_id**
  GPU device identifier (used only when the computation of the detector is running on a GPU).
- **det_num_threads**
  Specifies number of threads available for the detector (used only when the computation is running on a CPU). Use approximately 90% of logical processors if set to 0 or negative.
- **disable_det**
  If set to 1, the detection submodule will not be loaded and will not be available.
- **extras**
  General void pointer allocated for future use, **must be NULL if not in use**.

> **IMPORTANT**: Be careful when setting the number of threads by **ocr_num_threads** or **det_num_threads**, as setting a high value (same as number of logical processors) can lead to poor performance because LPM threads may block other processes, including the system processes needed by LPM.

LpmBoundingBox

```c
typedef struct {
    float  top_left_col;
    float  top_left_row;
    float  top_right_col;
    float  top_right_row;
    float  bot_left_col;
    float  bot_left_row;
    float  bot_right_col;
    float  bot_right_row;
} LpmBoundingBox;
```

LpmBoundingBox represents bounding box with zero based coordinates starting from top left corner of image. Bounding boxes are used either for positions of detections or for specification of the detection area.

The structure contains the following fields:

- **top_left_col**
  Top left column.
- **top_left_row**
  Top left row.
- **top_right_col**
  Top right column.
- **top_right_row**
  Top right row.
- **bot_left_col**
  Bottom left column.
- **bot_left_row**
  Bottom left row.
- **bot_right_col**
  Bottom right column.
- **bot_right_row**
  Bottom right row.

LpmDetResult

```
typedef struct {
    int                      lpm_id;
    int                      lpm_idx;
    int                      num_detections;
    LpmDetection            *detections;
    LpmDetection_extension1 *extras;
} LpmDetResult;
```

LpmDetResult structure holds an array of detections.

The structure contains the following fields:

- **lpm_id**
  ID of the the used LPM module.
- **lpm_idx**
  Index of the used LPM module.
- **num_detections**
  Number of detections.
- **detections**
  Array of detections.
- **extras**
  Additional details for detections, NULL if not in use. Used in version 7.3 and higher.

LpmDetResult_extension1

```
typedef struct {
    LpmDetection_extension1  *detections;
    void                     *extras;
} LpmDetResult;
```

LpmDetResult_extension1 structure holds an array of additional information for detections in version 7.3 or higher.

The structure contains the following fields:

- **detections**
  An array of additional information for detections.
- **extras**
  General void pointer allocated for future use, NULL if not in use.

LpmDetection

```
typedef struct {
    double              confidence;
    LpmBoundingBox      position;
    LpmDetectionLabel   label;
    ERImage             image;
    double              affine_mapping[6];
} LpmDetection;
```

LpmDetection contains data related to a single license plate or ADR plate or carbox detection.

The structure contains the following fields:

- **confidence**
  Plate detection confidence factor.
- **position**
  License plate position.

- **label**
  Detection type label.
- **image**
  The image crop of the detection. Please note that generation of this image can be disabled in configuration files.
- **affine_mapping**
  Array with affine mapping from plate image coordinates to source image coordinates. The array contains the first two rows of the affine transformation matrix, saved row-wise.

LpmDetection_extension1

```
typedef struct {
    float   occlusion;
    int     truncated;
    int     cluster_id;
    double  cluster_confidence;
    void    *extras;
} LpmDetection_extension1;
```

LpmDetection_extension1 contains additional data related to a single detection in version 7.3 and higher.

The structure contains the following fields:

- **occlusion**
  Specifies how much the detection is occluded. Negative value - not known, 0.0f - not occluded, 1.0f - fully occluded.
- **truncated**
  Contains -1 if not known, 0 if the detection is not truncated, 1 if it is truncated (the bbox does not cover the whole object).
- **cluster_id**
  ID of the cluster, to which this detection belongs. -1 if the cluster is not known, 0 means undefined, ID starts generally at 1. Detections of objects which are physically connected have same cluster_id, for example bounding box and license plate of same car will have same cluster_id.
- **cluster_confidence**
  Confidence factor for cluster_id prediction.
- **extras**
  General void pointer allocated for future use, NULL if not in use.

LpmOcrResult

```
typedef struct {
    int                lpm_id;
    int                lpm_idx;
    int                num_hypotheses;
    LpmOcrHypothesis   *hypotheses;
} LpmOcrResult;
```

LpmOcrResult structure holds an array of OCR hypotheses related to a single detection.

The structure contains the following fields:

- **lpm_id**
  ID of the used LPM module.
- **lpm_idx**
  Index of the used LPM module.

- **num_hypotheses**
  Number of OCR-hypotheses.
- **hypotheses**
  Array of OCR-hypotheses.

LpmOcrHypothesis

```
typedef struct {
    double          confidence;
    unsigned int    num_lines;
    LpmTextLine     *text_lines;
    char            *plate_type;
    double           plate_type_confidence;
    LpmLpDimensions  lp_dimensions;
    double          *lp_dimensions_confidence;
    void            *extras;
} LpmOcrHypothesis;
```

LpmOcrHypothesis structure holds one OCR hypothesis.

The structure contains the following fields:

- **confidence**
  Confidence factor for the current OCR result.
- **num_lines**
  Number of license/ADR plate text lines.
- **text_lines**
  Array of text lines of type *LpmTextLine* of the current license/ADR plate.
- **plate_type**
  A NULL-terminated string pointing to international license plate code.
  Note: When reading ADR plates, the value is "ADR" or "TRASH". If the value is "UNK", then it was recognized as a false positive detection.
- **plate_type_confidence**
  Confidence for the plate type prediction.
- **lp_dimensions**
  Predicted physical dimensions of the license plate.
- **lp_dimensions_confidence**
  Confidence for the dimensions prediction.
- **extras**
  General void pointer allocated for future use, NULL if not in use.

LpmLpDimensions

```
typedef struct {
    unsigned int  physical_width;
    unsigned int  physical_height;
} LpmLpDimensions;
```

LpmLpDimensions structure holds a physical width and height of a license plate in mm.

The structure contains the following fields:

- **physical_width**
  Physical width of the license plate in mm.
- **physical_height**
  Physical height of the license plate in mm.

LpmTextLine

```
typedef struct {
    double        line_confidence;
    unsigned int  length;
    int          *characters;
    double       *characters_confidence
} LpmTextLine;
```

LpmTextLine structure holds all data about one line of the license/ADR plate text.

The structure contains the following fields:

- **line_confidence**
  Output confidence for the whole line.
- **length**
  Text length (i.e. number of characters).
- **characters**
  Text in Unicode (UTF-32) of length-many characters.
- **characters_confidence**
  Array of length-many items containing the confidence for each character.

4.3   Functions

This chapter contains information about the LPM library functions present in the public API. The chapter is divided into four parts. The first part describes the *functions for handling the LPM engine*, the second part describes *functions for handling the LpmCameraViewParams structure*, the third section describes all *functions related to loading/unloading/running LPM modules* and the fourth section describes *error logging related functions*.

4.3.1   Main LPM engine functions

This part defines the API functions which are designed to initialize the LPM engine and to free the LPM engine, as well as to get the engine version and compilation date. The functions are: *lpmInit*(), *lpmFree*(), *lpmVersion*() and *lpmCompilationDate*(). These functions are declared in the **lpm.h** file.

lpmInit

Initializes the LPM engine and searches the given directory for installed LPM modules.

Specification:

```
int lpmInit(const char *lpm_directory, LPMState *lpm_state)
```

Input:

- **lpm_directory**
  LPM module base directory (e.g. "*../../modules-v7/x64*").
- **lpm_state**
  LPM state structure (LPM context) to be initialized.

Returns:

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

Description:

The function *lpmInit*() initializes the LPM engine and searches the given directory (e.g. "*../../modules-v7/x64*") for installed modules while assigning them unique, zero-based indices. Assigned indices range

from zero to the number of installed modules - 1. Function returns 0 on success.

Example:

```
LPMState lpm_state;
int ret_code;

if ((ret_code = lpmInit("../../modules-v7/x64", &lpm_state)) != 0)
{
// error handling
return -1;
}

// lpm_state can be used here
```

lpmFree

Frees the initialized LPM engine.

Specification:

```
void lpmFree(LPMState *lpm_state);
```

Input:

- **lpm_state**
  The LPM state created by *lpmInit*() function.

Description:

The function *lpmFree*() is used for freeing the LPM engine. When the SDK 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 a pointer to the structure *LPMState*, which was initialized using the *lpmInit*() function during engine initialization.

> **IMPORTANT**: Always free the LPM engine when it is not needed anymore, as otherwise your program will have memory leaks.

Example:

```
LPMState lpm_state;
lpmInit("../../modules-v6/x64", &lpm_state);
// code using lpm_state
// ...
lpmFree(&lpm_state);
```

lpmVersion

Returns the LPM engine version.

Specification:

```
unsigned long lpmVersion(void);
```

Returns:

- LPM engine version.

Description:

The function lpmVersion() returns the version of the LPM engine coded into one unsigned long integer. The least significant byte stores the subversion number and the second least significant byte stores the version number.

Example:

```
unsigned long longversion = lpmVersion();
unsigned char version = (unsigned char)(lpmVersion() >> CHAR_BIT);
unsigned char subversion = (unsigned char)(lpmVersion());
```

lpmCompilationDate

Returns the compilation date of the LPM engine.

Specification:

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

Returns:

- LPM compilation date.

Description:

The function *lpmCompilationDate*() returns the compilation date of the LPM engine in Mmm-dd-yyyy format.

Example:

```
char const *compilation_date = lpmCompilationDate();
printf("Compilation date: %s\n", compilation_date);
```

4.3.2 Camera view configuration functions

This part defines the API functions which are designed for handling the *LpmCameraViewParams* structure. The function *lpmLoadViewConfig*() reads this structure from a file and *lpmWriteViewConfig*() writes this structure to a text file. These functions are declared in the lpm.h file.

lpmLoadViewConfig

Loads the camera view parameters from file.

Specification:

```
int lpmLoadViewConfig(const char *filename, LpmCameraViewParams *camera_view_params);
```

Input:

- **filename**
  Path to a file, from which the config should be loaded, or NULL to use default parameters.
- **camera_view_params**
  Structure to be loaded with parameters from a given file.

Returns:

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

Description:

The function *lpmLoadViewConfig*() loads camera view parameters from a file specified by the filename parameter into the parameter camera_view_params. If the parameter filename is NULL, then default camera view parameters are returned. Camera view parameters stored in *LpmCameraViewParams* are used while loading LPM modules by the *lpmLoadModule*() function.

Example:

```
LpmCameraViewParams camera_view_params;
if (lpmLoadViewConfig("config_camera_view.ini", &camera_view_params) != 0)
{
    // Error handling
}
```

IMPORTANT: Camera view parameters do NOT apply when using GPU capable modules which work in a different way than previous modules.

lpmWriteViewConfig

Writes the camera view parameters to a given file.

Specification:

```
int lpmWriteViewConfig(const char *filename, LpmCameraViewParams camera_view_params);
```

Input:

- **filename**
  Path to the file where camera view parameters will be written.
- **camera_view_params**
  Pointer to *LpmCameraViewParams* to write.

Returns:

- **0** – File was successfully written.
- **other** – Error while saving file.

Description:

The function *lpmWriteViewConfig*() writes the structure *LpmCameraViewParams* with camera view parameters to the given file.

Example:

```
LpmCameraViewParams cvp;
cvp.view_type = LPM_VIEW_FRONTAL;
cvp.camera_aspect = 1.f;
cvp.min_horizontal_resolution = 175;
cvp.max_horizontal_resolution = 360;

lpmWriteViewConfig("../../modules-v7/config_camera_view.ini", cvp);
```

4.3.3   LPM modules handling functions

This part defines the API functions which are designed to handle LPM modules - to load them, run them, get information about them and to free them. The function *lpmLoadModule*() is used to load a LPM module, *lpmFreeModule*() to free a module, *lpmRunDet*() and *lpmRunOcr*() to run detection and OCR modules respectively, *lpmFreeDetResult*() and *lpmFreeOcrResult*() to free the returned results, *lpmGetNumAvlbModules*() to get the number of available LPM modules, *lpmGetModuleIndex*() and *lpmGetModuleIndexByName*() to get the module index by ID or name, *lpmGetModuleInfo*() to get all information about modules. These functions are declared in the **lpm.h** file.

lpmLoadModule

Loads an LPM module with a given index.

Specification:

```
int lpmLoadModule(LPMState lpm_state, int module_index,
                  LpmCameraViewParams *camera_view_params,
                  const LpmModuleConfig *module_config);
```

Input:

- **lpm_state**
  The LPM state created by the *lpmInit*() function.
- **module_index**
  Index of the LPM module you wish to load.
  Note that module index and module ID are two different things.
- **camera_view_params**
  Pointer to optional camera view parameters. Use NULL for default parameters.
- **module_config**
  Pointer to optional module configuration parameters. Use NULL to load values from configuration file.

Returns:

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

0 – the module was successfully initialized.

other – Error while initializing the module.

Description:

The function *lpmLoadModule*() initializes the LPM module with the given index. Module functions can be called after successful initialization. The LPM layer allows the user to activate and work with multiple modules simultaneously. The appropriate module index can be retrieved from the module ID and version using the *lpmGetModuleIndex*() function or the *lpmGetModuleIndexByName*() function, if the module name is known. The third parameter with camera view config is optional - pass in NULL to use default values. The fourth parameter with module config is also optional, pass in NULL to load the values from the module-specific configuration file.

Example:

```
int idx = 1;
LpmCameraViewParams camera_view_params;
// Initialize camera_view_params here...
LpmModuleConfig module_config;
// Initialize module_config here...

if (lpmLoadModule(lpm_state, idx, &camera_view_params, &module_config) != 0)
{
    // Error handling
}
// Use module functions here
```

IMPORTANT: Always free the LPM module when it is not needed anymore using *lpmFreeModule*(), otherwise your program will have memory leaks.

lpmFreeModule

Frees previously loaded LPM module with the given index.

Specification:

```
void lpmFreeModule(LPMState lpm_state, int module_index);
```

Input:

- **lpm_state**
  The LPM state created by the *lpmInit*() function.
- **module_index**
  Index of the LPM module to free.

Description:

The function *lpmFreeModule*() frees the previously loaded LPM module.

Example:

```
lpmLoadModule(lpm_state, idx, NULL, NULL);    // Init module
// Use module here
// ...
lpmFreeModule(lpm_state, idx);                // Free module
```

lpmRunDet

Runs license/ADR plate detection on the given image.

Specification:

```
LpmDetResult *lpmRunDet(LPMState lpm_state, int module_index,
                        ERImage image, const LpmBoundingBox *bounding_box);
```

Input:

- **lpm_state**
  The LPM state created by *lpmInit*() function.
- **module_index**
  Index of LPM module to use. *Note that module index and module ID are two different things.*
- **image**
  ERImage structure containing the input image for detection.
- **bounding_box**
  The bounding box of a detection area.

Returns:

- **NULL**
  Error during computation occurred.
- **other**
  LpmDetResult structure with all detections.

Description:

The function *lpmRunDet*() runs LPM detection module specified by module index idx on supplied image. Scanning area is specified by *LpmBoundingBox* structure. For a more detailed example, see the License/ADR plates detection part in the *Examples* section.

Example:

```
LpmBoundingBox bb;  // Variable holding bounding box of detector area
bb.top_left_col = 0;
bb.top_left_row = 0;
bb.bot_right_col = er_image.width - 1;
bb.bot_right_row = er_image.height - 1;

LpmDetResult *det_result = NULL;  // A pointer to the detection result structure
if ((det_result = lpmRunDet(lpm_state, idx, er_image, &bb)) == NULL) // Run detection
{
    // Error handling
}
// Working with result
```

IMPORTANT: With new GPU capable detection models, size and aspect ratio of bounding box can affect detection performance. Be careful when using very wide or very high bounding boxes.

IMPORTANT: Always free the structure with the detection result when it is not needed anymore using *lpmFreeDetResult*(), otherwise your program will have memory leaks.

lpmFreeDetResult

Frees detection result structure generated by *lpmRunDet*().

Specification:

```
void lpmFreeDetResult(LPMState lpm_state, LpmDetResult *detection_result);
```

Input:

- **lpm_state**
  The LPM state created by *lpmInit*() function.
- **detection_result**
  Pointer to the detection result structure to be freed.

Description:

The function *lpmFreeDetResult*() frees the detection result structure generated by *lpmRunDet*().

Example:

```
LpmDetResult *det_result = lpmRunDet(lpm_state, idx, er_image, &bb);  // Run detection
// Working with the result
// ...
lpmFreeDetResult(lpm_state, det_result);  // Free detection result
```

lpmRunOcr

Runs OCR on the given image.

Specification:

```
LpmOcrResult *lpmRunOcr(LPMState lpm_state, int module_index,
                        ERImage image, const LpmBoundingBox *detection_position,
                        LpmDetectionLabel detection_label);
```

Input:

- **lpm_state**
  The LPM state created by *lpmInit*().
- **module_index**
  Index of the LPM module to use. *Note that module index and module ID are two different things.*

- **image**
  ERImage structure containing the input image.
- **detection_position**
  The 4-point position of the detection.
- **detection_label**
  The detection label specifying the type of detection; can be obtained from the *LpmDetection* structure if using the *lpmRunDet*() function.

Returns:
- **NULL** – Error during computation occurred.
- **other** – *LpmOcrResult* structure with all detections.

Description:
The function *lpmRunOcr*() runs OCR on a supplied image. The detection area is specified by the *LpmBoundingBox* structure and the detection label by the *LpmDetectionLabel* structure. Usually, the bounding box and detection label are supplied to the OCR function from the detection output of *lpmRunDet*() function.

Example:
```
LpmOcrResult *ocr_result = NULL;  // A pointer to the OCR result structure
if ((ocr_result = lpmRunOcr(lpm_state, idx, er_image,
                            &(det_result->detections[j].position),
                            det_result->detections[j].label)) != NULL)
{
    // Error handling
}
```

> **IMPORTANT:** Always free the structure with the OCR result when it is not needed anymore using *lpmFreeOcrResult*(), otherwise your program will have memory leaks.

lpmFreeOcrResult

Frees the detection result structure generated by *lpmRunOcr*().

Specification:
```
void lpmFreeOcrResult(LPMState lpm_state, LpmOcrResult *ocr_result);
```

Input:
- **lpm_state**
  The LPM state created by the *lpmInit*() function.
- **ocr_result**
  Pointer to the OCR result structure to be freed.

Description:
The function *lpmFreeOcrResult*() frees the detection result structure generated by *lpmRunOcr*().

Example:
```
LpmOcrResult *ocr_result;  // A pointer to the OCR result structure
if ((ocr_result = lpmRunOcr(lpm_state, idx, er_image,
                            &(det_result->detections[j].position),
                            det_result->detections[j].label)) != NULL)
{
    // Error handling
}
// Working with the result
lpmFreeOcrResult(lpm_state, ocr_result);  // Free OCR result
```

lpmGetNumAvlbModules

Gets the number of available LPM modules.

Specification:
```
int lpmGetNumAvlbModules(LPMState lpm_state);
```

Input:

- **lpm_state**
  The LPM state created by the *lpmInit*() function.

Returns:

- **-1**   – Error during computation occurred.
- **other** – Number of LPM modules.

Description:

The function *lpmGetNumAvlbModules*() returns the number of available LPM modules.

Example:
```
int num_available_modules = lpmGetNumAvlbModules(lpm_state);
```

lpmGetModuleIndex

Gets the LPM module index (handle) from the module ID and its version.

Specification:
```
int lpmGetModuleIndex(LPMState lpm_state, int module_id, int version, int subversion);
```

Input:

- **lpm_state**
  The LPM state created by the *lpmInit*() function.
- **module_id**
  ID of the module.
- **version**
  Version of the module.
- **subversion**
  Subversion of the module.

Returns:

- **-1**   – Error during computation occurred.
- **other** – Module index.

Description:

The function *lpmGetModuleIndex*() returns the LPM module index (handle) from the module ID and its version. Module indices can vary with each program execution because they depend on the search order of the given module directory. Set the version and subversion to zero to get the index of the latest available module with the provided ID.

Example:
```
int idx = -1;
if ((idx = lpmGetModuleIndex(lpm_state, MODULE_ID, 0, 0)) == -1)
{
    // Error handling
}
// Working with the idx
```

lpmGetModuleIndexByName

Gets the LPM module index (handle) from the module name.

Specification:

```
int lpmGetModuleIndexByName(LPMState lpm_state, const char *module_name);
```

Input:

- **lpm_state**
  The LPM state created by the *lpmInit*() function.
- **module_name**
  Name of the given module. See *lpmGetModuleInfo*().

Returns:

- **-1** – Error during computation occurred.
- **other** – Module index.

Description:

The function *lpmGetModuleIndexByName*() returns the LPM module index (handle) from the module name. Module indices can vary with each program execution because they depend on the search order of the given module directory.

Example:

```
int idx = -1;
if ((idx = lpmGetModuleIndexByName(lpm_state, "001-frontal.adr-adr-v7.0")) == -1)
{
    // Error handling
}
// Working with the idx
```

lpmGetModuleInfo

Retrieves information about the LPM module.

Specification:

```
LpmModuleInfo *lpmGetModuleInfo(LPMState lpm_state, int module_index);
```

Input:

- **lpm_state**
  The LPM state created by the *lpmInit*() function.
- **module_index**
  Index of the LPM module to use. *Note that module index and module ID are two different things.*

Returns:

- **NULL** – Error during computation occurred. Call *lpmGetLastError*() to get error code.
- **other** – *LpmModuleInfo* structure with all information about the module.

Description:

The function *lpmGetModuleInfo*() returns all information about the module with the desired index.

Example:

```
int idx = 1;
LpmModuleInfo *lmi = lpmGetModuleInfo(lpm_state, idx);
// Each module has its own ID and this is a way how to get it.
printf(" Module ID: %d\n", lmi->id);
// Another module properties ...
printf(" Module name: %s\n", lmi->name);
printf(" Module path: %s\n", lmi->path);
printf(" Module date: %s\n", lmi->date);
printf(" Module version: %d.%d\n\n", lmi->version, lmi->subversion);
```

4.3.4   Error logging functions

This part defines the API functions which are designed for error logging and easier debugging. The *lpmGetLastError*() function returns the error code of the last error. These functions are defined in the **lpm.h** file.

lpmGetLastError

Gets the code of the last occurred error.

Specification:

```
int lpmGetLastError(void);
```

Returns:

- The ID of the last occurred error.

Description:

The function *lpmGetLastError*() returns the error code of the last error.

Example:

```
int err_code = lpmGetLastError();
```

5 Examples

This chapter describes the example which is contained in the SDK package. The example is used to demonstrate the functionality of the SDK. The source code is included in the package and is described in detail.

5.1 LPM SDK Example

The LPM SDK package contains an example which is used to demonstrate the basic functionality of the LPM SDK on several input images. The example detects license/ADR plates on multiple images and runs OCR on these detections. It also demonstrates how to load LPM modules and get information about the available modules. This chapter describes in detail the example together with references to important parts of this document.

The example is in the folder **[LPM_SDK]/examples/example-anpr-implink/**. The folder contains all source code and files needed for building the example. In Windows packages, a Visual Studio 2019 project is included; in Linux packages, a Makefile is included.

5.1.1 Initialization of the LPM engine

The first thing to do is initializing the LPM engine using the *lpmInit*() function. The parameters of this function are the directory where the LPM modules are located (e. g. **../../modules-v7/x64** for default package directory structure on Windows x64 system), and a LpmState pointer.

```
#define MODULES_DIR "../../modules-v7/x64"
LpmState lpm_state;                              // A void pointer to the lpm state variable
int ret_code;
if ((ret_code = lpmInit(MODULES_DIR, &lpm_state)) != 0)
{
    // Error handling
}
printf("LPM v%u.%u initialized.\n\n", (unsigned char)(lpmVersion() >> CHAR_BIT),
        (unsigned char)(lpmVersion()));
```

After successful initialization of the engine, modules can be loaded, and the engine can be used.

5.1.2 Listing of available LPM modules

To start working with LPM modules, their indexes are needed as handles. Indexes are numbers from zero to the number of available modules -1. Code bellow illustrates how to get the number of modules and list information about them:

```
LpmModuleInfo *module_info = NULL;   // A pointer to the module info structure
// Get the number of available lpm modules and print some basic information about them
int num_available_modules = lpmGetNumAvlbModules(lpm_state);

printf("Listing %d modules:\n"}, num_available_modules);
for (int i = 0; i < num_available_modules; i++)
{
    module_info = lpmGetModuleInfo(lpm_state, i);

    // Each module has its own ID and this is a way how to get it
    printf("  Module ID     : %d\n", module_info->id);
    // Module name, version, date and others are available as defined in LpmModuleInfo
    printf("  Module name   : %s\n", module_info->name);
    printf("  Module path   : %s\n", module_info->path);
    printf("  Module date   : %s\n", module_info->date);
    printf("  Module version: %d.%d\n\n", module_info->version, module_info->subversion);
}
```

5.1.3   Writing camera view parameters

Another part of the example concerns writing camera view parameters (image resolution, aspect ratio, …) to a file which can be then loaded and used to supply these parameters when loading modules. See the *LpmCameraViewParams* structure for more information about these parameters.

```
LpmCameraViewParams camera_view_params;
camera_view_params.view_type = LPM_VIEW_FRONTAL;
camera_view_params.camera_aspect = 1.f;
camera_view_params.min_horizontal_resolution = 135;
camera_view_params.max_horizontal_resolution = 260;

lpmWriteViewConfig(VIEW_CONFIG_FILENAME, camera_view_params);
```

5.1.4   Loading camera view parameters

Camera view parameters can be loaded from a config file in the following way:

```
// Load the camera view parameters from config file
LpmCameraViewParams camera_view_params;
if (lpmLoadViewConfig(VIEW_CONFIG_FILENAME, &camera_view_params) != 0)
{
    // Error handling
}
```

> **IMPORTANT:** camera view parameters do NOT apply when using the new GPU capable detection modules which work in a different way than the previous modules.

5.1.5   Getting LPM module index

To be able to load the appropriate module, the index of the module must be known. If the module id is known, then the easiest way to get this index is to use the function *lpmGetModuleIndex*(). If the version and subversion of the module is specified, then the function looks for the specified version; otherwise the function searches for all versions of the module with the specified ID and returns the latest version.

```
// Check if a module with a given MODULE_ID is available and if so then return its index
//     (handle).
int idx; // Module index (handle)
if ((idx = lpmGetModuleIndex(lpm_state, MODULE_ID, 0, 0)) == -1)
{
    // Error handling, module not found
}
```

5.1.6   Setting module configuration parameters

A module can be initialized with special configuration parameters.

For versions prior to 7.3 you can set the parameters to allow GPU computation or pass in a NULL pointer to use default values from the configuration file. We will set them to select computation on CPU:

```
LpmModuleConfig lpm_module_config;
lpm_module_config.compute_on_gpu = 0;
```

For version 7.3 and higher settings are provided by an extension structure:

```
LpmModuleConfig_extension1 lpm_module_config_extension1;
memset(&lpm_module_config_extension1, 0, sizeof(lpm_module_config_extension1));
lpm_module_config_extension1.lpm_config_filename = "config.ini";
lpm_module_config_extension1.ocr_compute_on_gpu = 1;
lpm_module_config_extension1.det_config_filename = "config-det.ini";
lpm_module_config_extension1.det_compute_on_gpu = 1;
lpm_module_config.extras = &lpm_module_config_extension1;
```

5.1.7   Loading LPM module

When the index of the desired module is known, the module can be loaded using the function *lpmLoadModule*(). The third parameter containing camera view parameters is optional, NULL value can be used to use default parameters from config file.

```
// Now load the module.
if (lpmLoadModule(lpm_state, idx, &camera_view_params, &lpm_module_config) != 0)
{
    // Error handling
}
```

One or more LPM modules can be loaded at one time simply by calling *lpmLoadModule*() multiple times. Module used by SDK functions is then specified by module_index parameter.

5.1.8   Input image loading

Before a LPM module can be used, image data must be loaded and decoded to a supported image format The example uses the Eyedea Recognition's custom image structure *ERImage* to manipulate the images. The image is loaded to a *ERImage* structure using the *erImageRead*() function. For more information about image structure and image manipulation functions used by LPM SDK see chapter *ERImage*.

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

5.1.9   License/ADR plates detection

Code bellow illustrate how to run a license/ADR plates detection procedure on multiple files. Image files are loaded into *ERImage* structures and a bounding box covering the whole image is supplied to the detection function by a *LpmBoundingBox* structure. At the end, the structure with the detection result is deleted using the *lpmFreeDetResult*() function.

```c
#define NUM_IMG 3
const char TestImageList[NUM_IMG][_MAX_PATH] = { "../images/1.jpg", "../images/2.jpg",
                                                 "../images/3.jpg"}; // images to process

LpmDetResult *det_result = NULL;    // A pointer to the detection result structure

for (int i = 0; i < NUM_IMG; i++) // Cycle through the images
{
    ERImage er_image;
    if (erImageRead(&er_image, TestImageList[i]) != 0)
    {
        // Error handling
    }

    // Specify an area of the input image where detection will be performed
    LpmBoundingBox bb;
    bb.top_left_col = 0;
    bb.top_left_row = 0;
    bb.bot_right_col = er_image.width - 1;
    bb.bot_right_row = er_image.height - 1;

    // Run LP detection
    if ((det_result = lpmRunDet(lpm_state, idx, er_image, &bb)) == NULL)
    {
        // Error handling
    }

    // Print information about the detection
#ifdef LPM_EXTENSIONS_v7_3
    if (det_result->extras != NULL)
    {
        LpmDetection_extension1 &detection_extension1 = det_result->extras->detections[j];
        printf(" - Detection %d, confidence %.2f, truncated %d, occlusion %.2f, cluster_id
   %d:\n",
            j + 1,
            detection.confidence,
            detection_extension1.truncated,
            detection_extension1.occlusion,
            detection_extension1.cluster_id);
    }
    else
    {
#endif
        printf(" - Detection %d, confidence %.2f:\n", j + 1, detection.confidence);
#ifdef LPM_EXTENSIONS_v7_3
    }
#endif
```

5.1.10 Printing detection information

This code illustrates how to print detection information. In version 7.3 and higher, additional information is stored in an extension structure, and printing of this information is enabled by a preprocessor directive and by checking for NULL.

```c
// ...
// Print information about the detection
#ifdef LPM_EXTENSIONS_v7_3
if (det_result->extras != NULL)
{
    LpmDetection_extension1 &detection_extension1 = det_result->extras->detections[j];
    printf(" - Detection %d, confidence %.2f, truncated %d, occlusion %.2f, cluster_id %d:\n",
            j + 1,
            detection.confidence,
            detection_extension1.truncated,
            detection_extension1.occlusion,
            detection_extension1.cluster_id);
}
else
{
#endif
    printf(" - Detection %d, confidence %.2f:\n", j + 1, detection.confidence);
#ifdef LPM_EXTENSIONS_v7_3
}
#endif

// Do OCR or other stuff with detections
// ...
```

5.1.11   License/ADR plate OCR

When license/ADR plates are detected, OCR is called on every plate detection. More hypotheses can be returned for each plate, together with their confidences. Bounding boxes of plate detections are supplied directly to *lpmRunOcr*() function along with the original image. In version 7.3 and higher, detection results contain an extension structure which holds additional information. At the end, the structure with the OCR result is deleted using the *lpmFreeOcrResult*() function.

```cpp
// Running OCR on each LP detection
LpmOcrResult *ocr_result = NULL;                          // A pointer to the OCR result structure
for (int j = 0; j < det_result->num_detections; j++)
{
    printf(" - Detection %d:\n", j + 1);
    if ((ocr_result = lpmRunOcr(lpm_state, idx, er_image,
        &(det_result->detections[j].position), det_result->detections[j].label)) != NULL)
    {
        // We take the first OCR hypothesis.
        LpmOcrHypothesis &hypothesis = ocr_result->hypotheses[0];
        printf("    - llpc: %s, confidence:
  %.2f\n",hypothesis.plate_type,hypothesis.confidence);
        printf("    - dimensions: w*h=%d*%d[mm], confidence: %.2f\n",
                hypothesis.lp_dimensions.physical_width,
                hypothesis.lp_dimensions.physical_height,
                hypothesis.lp_dimensions_confidence);

        // Print all the lines contained in the hypothesis
        for (unsigned int k = 0; k < hypothesis.num_lines; k++)
        {
            // Note that the prediction can contain non-ASCII characters
            printf("    - line %d, ASCII: '", k + 1);
            for (unsigned int l = 0; l < hypothesis.text_lines[k].length; l++)
            {
                printf("%c", hypothesis.text_lines[k].characters[l]);
            }
            printf("', Unicode: ");
            for (unsigned int l = 0; l < hypothesis.text_lines[k].length; l++)
            {
                printf("0x%X ", hypothesis.text_lines[k].characters[l]);
            }
            printf(", length %d, confidence %.2f\n", hypothesis.text_lines[k].length,
                    hypothesis.text_lines[k].line_confidence);
        }
        // Empty LP/ADR table can be recognized using predicted number of lines
        if (hypothesis.num_lines == 0)
        {
            printf("        - empty\n");
        }
        printf("\n");
    }
    lpmFreeOcrResult(lpm_state, ocr_result);
}
```

5.1.12   Cleaning up

At the end, when you are done working with the LPM SDK instance(for example at the end of the program), it must be deleted together with all the loaded modules and camera view parameters structures. To delete these, use the API functions *lpmFreeModule*() and *lpmFree*(), which are designed for this purpose.

```cpp
// Finish work with the current module}
lpmFreeModule(lpm_state, idx);

// Free the LPM state}
lpmFree(&lpm_state);
```

6 Modules configuration files

This chapter describes configuration files used by the SDK modules for some of the configuration. The SDK uses multiple configuration files, this chapter will cover two main configuration files.

Configuration files use an INI style key-value pair format, and the files are divided into multiple sections. Each section starts with a name enclosed in square brackets, and can contain multiple key-value pairs, each on a new line. Key-value pairs are in the **parameter_name=parameter_value** format. Values can be numbers, strings enclosed in quotation marks or **True** or **False** for Boolean values. Comments can be written after the **#** sign.

The first part of this chapter describes the General configuration file config.ini, and the second part describes the Detector configuration file config-det.ini. In some configurations, LPM may also use other configuration files, which are not described in this chapter.

Some settings can be set in multiple ways - either via a configuration file, or by passing a configuration structure to an SDK function. Values in the detector configuration file config-det.ini have the lowest priority and are overwritten by values in the general configuration file config.ini. Values passed to the module initialization function *lpmLoadModule*() in *LpmModuleConfig* and *LpmModuleConfig_extension1* structures have the highest priority and will overwrite the settings in configuration files.

6.1 General configuration file config.ini

This is the main configuration file, and values from this file will overwrite values from the detector config file config-det.ini. Some of these values can be overwritten by *LpmModuleConfig* and *LpmModuleConfig_extension1* structures passed to *lpmLoadModule*(). config-det.ini contains the following sections:

6.1.1 EYEDENTIFY PARAMETERS

These are the parameters used prior to version 7.3, which are now deprecated. In version 7.3 and higher, please use OCR PARAMETERS and DET PARAMETERS instead.

This section contains the following parameters:

- **edf_compute_on_gpu**
  Boolean (True/False) value whether to use GPU for detection and OCR.
- **edf_gpu_device_id**
  GPU id of the device to be used for computation if GPU computation is enabled.

Example:

```
[EYEDENTIFY PARAMETERS]
edf_compute_on_gpu  = False
edf_gpu_device_id   = 0
```

6.1.2 OCR PARAMETERS

Parameters for the OCR part of the SDK:

- **ocr_compute_on_gpu**
  Boolean (True/False) value whether to use GPU for OCR.
- **ocr_gpu_device_id**
  GPU id of device to be used for OCR computation if OCR computation on GPU is enabled.

- **ocr_num_threads**

  Number of threads for OCR computation. The value 0, used by default, corresponds to 90% of your logical processors. Be careful when setting high values, because setting this value to or above the number of your logical processors may block all other processes, including the system processes used by the SDK.

- **disable_ocr**

  Setting this to True will disable OCR computation. Trying to run the OCR function will return NULL and a warning will be printed to standard output.

Example:

```
[OCR PARAMETERS]
ocr_compute_on_gpu  = False
ocr_gpu_device_id   = 0
ocr_num_threads     = 0
disable_ocr         = False
```

6.1.3   OCR MODELS

This section deals with setting which OCR model is to be used, depending on the label of each detection of the detection output.

- **model[n]**

  Definition of labels for each model file. For each model, the definition should look like model[number]=model_dat_file,label1,label2,… where model_dat_file is the filename of the model dat file, and label1, label2,… are the detector label numbers for which this OCR model will be used. The model with no attached labels acts as the default model, and is used for detection labels that are not explicitly attached to any model declared in this section. The models numbering [n] in model[n] is not important. Labels are defined in structure *LpmDetectionLabel* in file **lpm_type.h**

Example:

```
[OCR MODELS]
model1 = CNN_ANPRTF2LITE_EU_GRAY_96x24_NONE_LIN_EXP07_enc.dat
model2 = CNN_ANPRTF2LITE_ML_GRAY_64x40_NONE_LIN_EXP05_enc.dat,1002,1102,2000,2001,2002,2100
```

6.1.4   DET PARAMETERS

This section contains parameters for the detector:

- **det_config_filename**

  Filename of the detector configuration file relative to the directory of the general configuration file. Default is config-det.ini.

- **det_compute_on_gpu**

  Boolean (True/False) value whether to run the detector on GPU.

- **det_gpu_device_id**

  GPU id of the device to be used for detection computation if detection computation on GPU is enabled.

- **det_num_threads**

  Number of threads for detection computation. The value 0, used by default, corresponds to 90% of your logical processors. Be careful when setting high values, because setting this value to or above the number of your logical processors may block all other processes, including the system processes used by the SDK.

- **disable_det**

Setting this to True will disable the detector. Trying to run the detector function *lpmRunDet*() will return NULL and a warning will be printed to standard output.

Example:

```
[DET PARAMETERS]
det_config_filename = "config-det.ini"
det_compute_on_gpu  = False
det_gpu_device_id   = 0
det_num_threads     = 0
disable_det         = False
```

6.1.5   LPIMAGE CROP PARAMETERS

This section contains parameters for generating image crops from detection outputs:

- **lp_crop_enabled**
  This Boolean value controls whether an image crop should be generated from detections. If set to True, an image crop will be generated and accessible via *LpmDetection*.image. Set to False for speed optimization if you do not need image crops for your own purposes.
- **lp_img_width**
  Width of the cropped image in pixels. Must be greater than zero.
- **lp_img_height**
  Height of cropped image in pixels. Set to 0 for automatic calculation of height according to crop width and detection aspect ratio.

Example:

```
[LPIMAGE CROP PARAMETERS]
lp_crop_enabled = True
lp_img_width    = 256
lp_img_height   = 0
```

6.2   Detector configuration file config-det.ini

This is the configuration file of the detector. Most of the values in this file are primarily for internal use and are not to be modified. Please be careful when changing the values in this file. The file contains the following sections:

6.2.1   MODULE

This section stores data about the module and some of its settings:

- **type**
  Type of the detector module. Can specify multiple types, in which case if the initialization of the first type fails, the SDK will try to initialize the next module. This setting can be used to try to initialize the GPU version, and if this fails (for example if some GPU libraries are missing), the CPU version is initialized instead.
- **name**
  Name of the module used internally.
- **group**
  Name of preprocessed image used internally.

- **num_threads**
  Number of threads for detection computation. The value 0, used by default, corresponds to 90% of your logical processors. Be careful when setting high values, because setting this value to or above the number of your logical processors may block all other processes, including the system processes used by the SDK.
- **gpu_device_id**
  GPU id of the device to use for detection computation if detection computation on GPU is enabled.
- **use_gpu**
  Boolean (True/False) value whether to use GPU for detection computation.

Example:

```
[MODULE]
type         = "Tf2Lite-GPU,Tf2Lite"
name         = "Tf2lite-CNN"
group        = "Tf2lite-lp-eu-rgb-608x416"
num_threads  = 0
gpu_device_id = 0
use_gpu      = 0
```

6.2.2   NMS PARAMETERS

These are parameters for non-maximum suppression which is used to filter out duplicated detections:

- **overlap**
  Minimal overlap to apply NMS.
- **nms**
  Number defining the type of non-maximum suppression to use:
  0 - no-nms, 1 - simple nms, 2 - nms+, 3 - nms+ suppress-nested
- **labels**
  Number defining how NMS deals with detections with different labels:
  0 – ignore labels, 1 – separate NMS for each label.
- **threshold**
  Final threshold on the score.

Example:

```
[NMS PARAMETERS]
overlap   = 0.5
nms       = 1
labels    = 0
threshold = 0.2
```

6.2.3   DETECTION MODELS

This section contains parameters defining model .dat files:

- **model_filename**
  Path of the model .dat file relative to the detector config.

Example:

```
[DETECTION MODELS]
model_filename="models/CNN_DETECTTF2LITE_LP_EU_BGR_608x416_NONE_NONE_EXP22_enc.dat"
```

6.2.4   ROI

This section contains parameters which can set the region of interest for the detector. The detector will only run on the region of interest rectangle defined by:

- **x**
  X position of the top left corner of the region of interest.
- **y**
  Y position of the top left corner of the region of interest.
- **width**
  Width of the region of interest in pixels.
- **height**
  Height of the region of interest in pixels.

Example:

```
[LPIMAGE CROP PARAMETERS]
x      =   200
y      =   200
width =  1200
height=   500
```

6.2.5   PADDING

This section contains parameters used to add padding to the input image. This can be used for some types of detectors to allow detection at the edges of images:

- **padding**
  Four numbers enclosed in square brackets specifying the padding size on each side of the image in the order of left, top, right, bottom.

Example:

```
[LPIMAGE CROP PARAMETERS]
padding = "[100, 100, 0, 0]"
```

7 ERImage Application Interface

This part describes ERImage library used for image data storage and manipulation by LPM SDK. The *Image Format* section describes how image data is stored in the memory from a theoretical point of view, and the remaining parts cover the application interface used for image manipulation using the data structure *ERImage*. Description of all available *Enumerators*, *Structures* and *Functions* is included.

7.1 Image Format

Digital image data can be persisted in many different forms. Since it is the main input of the processing, it is very important to understand the form used for image storage and manipulation. Currently five color models are supported in the *ERImage* image structure. The first is the *BGR* color model, the second is the *Gray* color model, the third is the *YCbCr I420* color model, the fourth is the *BGRA* color model, and the fifth is the *YCbCr NV12* color model.

7.1.1 BGR

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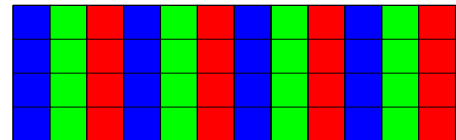
The model stores image using three values per pixel, where the first value is the blue component, the second value is the green component and the third is the red component. An image is saved row by row in a 1D array. The following formulas show how to access the pixel color components B, G and R in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

$$B(x,y) = data(3*(width*y+x)+0)$$ B component at (x, y) coordinates
$$G(x,y) = data(3*(width*y+x)+1)$$ G component at (x, y) coordinates
$$R(x,y) = data(3*(width*y+x)+2)$$ R component at (x, y) coordinates

7.1.2 Gray

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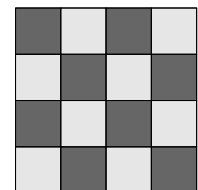
$$Y(x,y) = data(width*y+x)$$ Y component at (x, y) coordinates

7.1.3 YCbCr I420

The three-plane model *YCbCr I420* is used for storing color image, where the first plane contains luminance (Y component, image brightness), the second plane contains the blue-difference chroma component (Cb) and the third plane contains the red-difference chroma component (Cr). Cb and Cr planes have half the resolution of the Y image plane. Four neighboring Y values belongs to one Cb and one Cr value.

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

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

Y component at (x, y) coordinates

$$|Y| = width * height$$

Size of the Y image plane

$$Cb(x,y) = data(|Y| + \frac{y}{2} * \frac{width}{2} + \frac{x}{2})$$

Cb component at (x, y) coordinates

$$|Cb| = |Cr| = \frac{width * height}{4}$$

Size of the Cb and Cr image plane

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

Cr component at (x, y) coordinate

7.1.4 BGRA

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

$$B(x,y) = data(4 * (width * y + x) + 0)$$ B component at (x, y) coordinates
$$G(x,y) = data(4 * (width * y + x) + 1)$$ G component at (x, y) coordinates
$$R(x,y) = data(4 * (width * y + x) + 2)$$ R component at (x, y) coordinates
$$A(x,y) = data(4 * (width * y + x) + 3)$$ A component at (x, y) coordinates

7.1.5 YCbCr NV12

The two-plane model *YCbCr NV12* is used for storing color images, where the first plane contains luminance (Y component, image brightness) and the second plane contains interleaved blue-difference chroma components (Cb) and red-difference chroma components (Cr). The Cb and Cr planes have half the height and the same width as the Y image plane (because there are two components). Four neighboring Y values belongs to one Cb and one Cr value.

Y00	Y01	Y02	Y03	Y04	Y05
Y06	Y07	Y08	Y09	Y10	Y11
Y12	Y13	Y14	Y15	Y16	Y17
Y18	Y19	Y20	Y21	Y22	Y23
CB0	CR0	CB1	CR1	CB2	CR2
CB3	CR3	CB4	CR4	CB5	CR5

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

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

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

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

$$Cr(x,y) = data(|Y| + \frac{y}{2} * width + \frac{x}{2} + 1)$$ Cr component at (x, y) coordinate

$$|CbCr| = width * \frac{height}{2}$$ Size of the CbCr image plane

7.2 Application Interface

7.2.1 Enumerators

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

ERImageColorModel

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

- **ER_IMAGE_COLORMODEL_UNK = 0**
  Default value - Unknown color model.
- **ER_IMAGE_COLORMODEL_GRAY = 1**
  One-channel grayscale color model. Image luminance values are saved row by row.
- **ER_IMAGE_COLORMODEL_BGR = 2**
  Three-channel *BGR* color model. Three values per pixel, stored row by row.
- **ER_IMAGE_COLORMODEL_YCBCR420 = 3**
  Three-plane *YCbCr I420* color model. Luminance plane and two chroma planes are stored separately, each row by row.
- **ER_IMAGE_COLORMODEL_BGRA = 4**
  Four-channel *BGRA* color model. Four values per pixel, stored row by row.

- **ER_IMAGE_COLORMODEL_YCBCRNV12 = 5**
  Two-plane *YCbCr NV12* color model. Luminance plane and interleaved chroma plane are stored separately each row by row.

ERImageDataType

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

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

7.2.2   Structures

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

ERImage

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

ERImage represents the digital image data in a special structure designed to work with the LPM SDK. The structure contains the color model and the data type in the *ERImageColorModel*, and the *ERImage-DataType* enumerators, together with the parameters defining the size of the image and the underlying data. Image data is saved in the data field row by row as a contiguous 1D array. For more information see the section *Image Format*.

- **color_model**
  Image data color model represented by the enumerator *ERImageColorModel*.
- **data_type**
  Image date type represented by the enumerator *ERImageDataType*.
- **width**
  Width of the image in pixels.
- **height**
  Height of the image in pixels.
- **num_channels**
  Number of image channels. Zero for YCbCr color models.
- **depth**
  Size of one image pixel in bytes. Zero for YCbCr color models.
- **step**
  Number of bytes between each two beginnings of the row in the data array.

- **size**
  Size of the image in bytes.
- **data_size**
  Size of the allocated data in the structure.
- **data**
  Array containing the image data.
- **Row_data**
  Array containing pointers to the data array. Each element points to the beginning of the specific image row in the data array.
- **data_allocated**
  Value containing the flag whether the data field was allocated within the structure or on the user's side. (0 – allocated by user, 1 – allocated within the structure)

7.2.3   Functions

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

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

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

erImageAllocate

Allocates an *ERImage* structure.

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

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

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

Description:

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

Example:

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

erImageAllocateBlank

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

Specification:

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

Input:

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

Returns:

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

Description:

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

Example:

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

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

erImageAllocateAndWrap

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

Specification:

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

Input:

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

Returns:

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

Description:

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

Example:

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

erImageCopy

Performs a deep copy of the *ERImage* structure instance.

Specification:

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

Input:

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

Returns:

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

Description:

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

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

Example:

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

erImageGetDataTypeSize

Returns the size of the specific *ERImageDataType* in bytes.

Specification:

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

Input:

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

Returns:

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

Description:

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

Example:

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

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

erImageGetColorModelNumChannels

Returns the number of channels of the provided *ERImageColorModel* value.

Specification:

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

Input:

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

Returns:

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

Description:

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

IMPORTANT: For the ER_IMAGE_COLORMODEL_YCBCR* color model, zero is returned.

Example:

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

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

unsigned int numPlanesYCBCR420 = erImageGetColorModelNumChannels}(ER_IMAGE_COLORMODEL_YCBCR420);
// numPlanesYCBCR420  == 0
```

erImageGetPixelDepth

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

Specification:

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

Input:

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

Returns:

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

Description:

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

IMPORTANT: For the ER_IMAGE_COLORMODEL_YCBCR* color model, zero is returned.

Example:

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

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

erVersion

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

Specification:

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

Returns:

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

Description:

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

Example:

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

erImageRead

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

Specification:

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

Input:

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

Returns:

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

Description:

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

Supported image formats:

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

Example:

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

erImageWrite

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

Specification:

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

Input:

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

Returns:

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

Description:

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

Example:

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

erImageFree

Frees the whole *ERImage* structure instance.

Specification:

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

Input:

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

Description:

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

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

Example:

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

8   LPM SDK Licensing

LPM SDK 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 tral version, which uses software key instead.

8.1   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	

8.2   Licenses Overview

Several licenses are available for the LPM SDK. 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.

8.2.1   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.

8.2.2   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.

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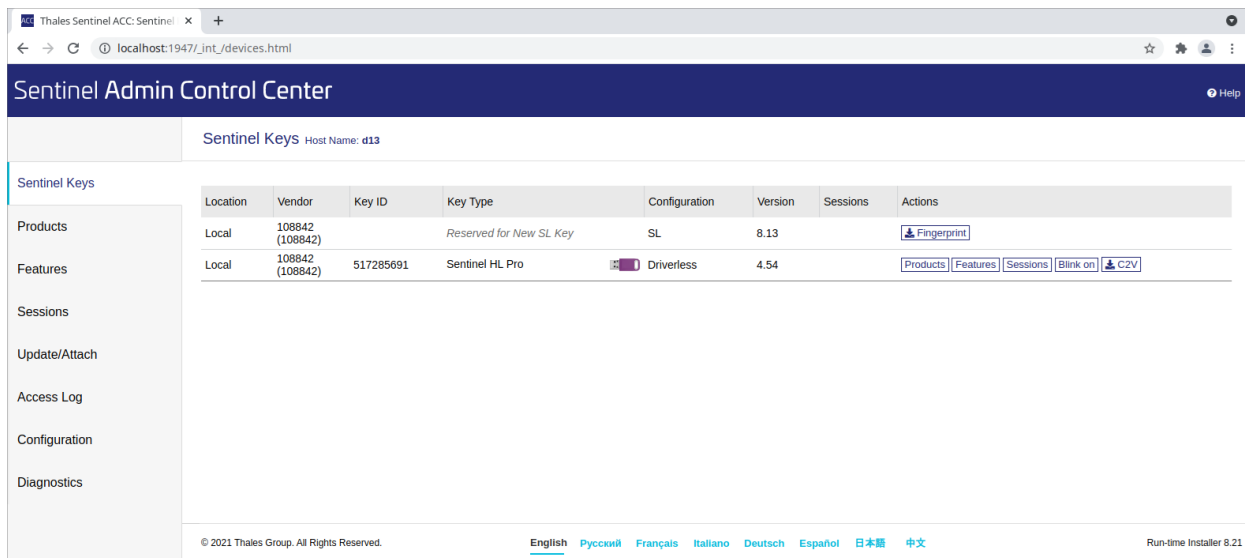
An execution counting license allows counting the number of times the license was logged in. The SDK is designed in such a way that it logs in the license every time a specified SDK function is called. It allows limiting the number of executions with the license. This type of license is used mainly in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

8.3   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.

8.3.1   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.
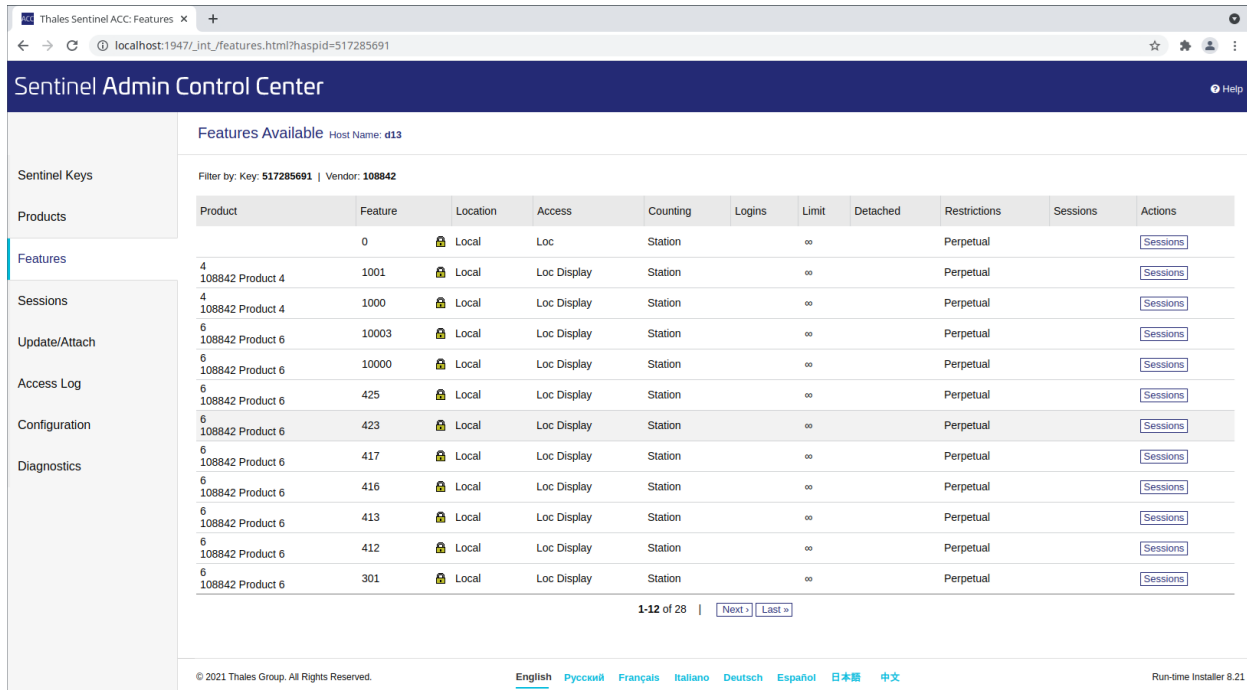
Web interface with list of plugged keys on http://localhost:1947/_int_/devices.html

8.3.2   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, …).

Web interface with key 517285691
details on http://localhost:1947/_int_/features.html?haspid=517285691
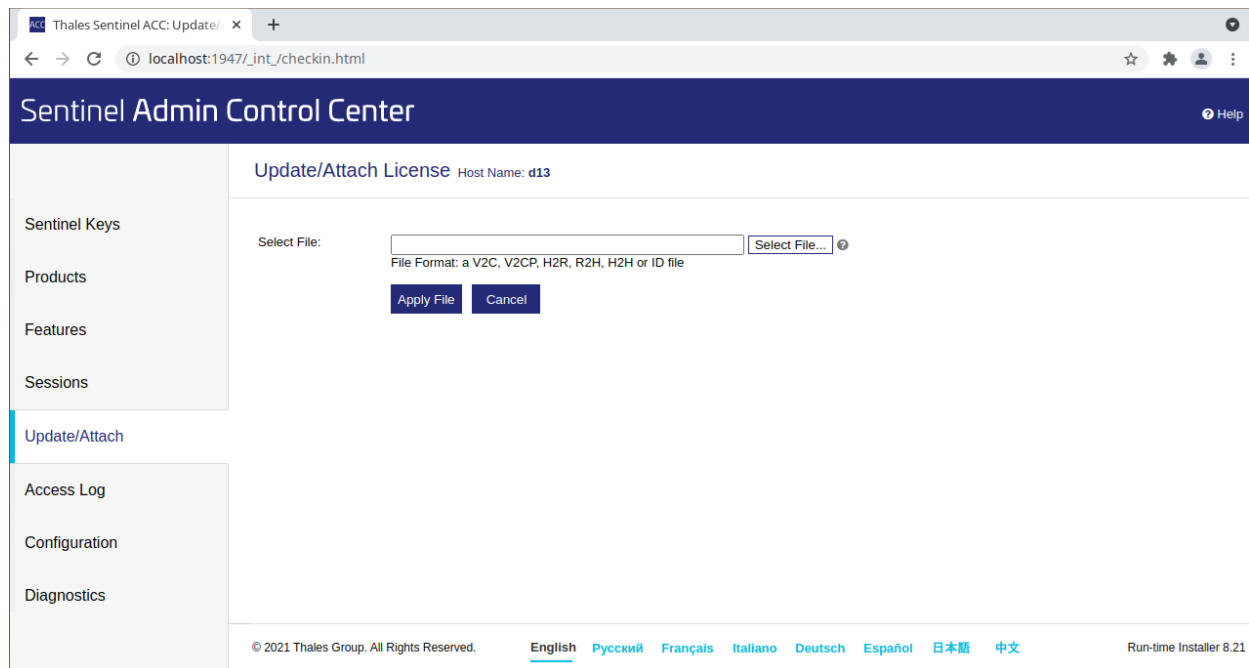
8.4   License Update

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.

8.4.1   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.

8.4.2   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.

9 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:

- Boost
- 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/)."