MMRRESTServer 2.1.1

User Guide

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

Contact:

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

web: http://www.eyedea.cz

email: info@eyedea.cz

Table of Contents

1 Introduction

Eyedea Recognition's MMRRESTServer is a Java server application with REST interface running on Tomcat Docker container which allows to detect vehicle license plates in input images and recognize the text and type of detected license plates, as well as the view, category, make, model, generation, variation and color of the vehicle.

MMRRESTServer uses our state-of-the-art libraries, LPM and MMR SDK, with a possibility to easily switch to the latest models or modules for a different region. Both server REST interface and the simple web application built on it provide the detection and recognition used in various use cases, as well as server monitoring.

2 Hardware requirements

2.1 Minimal requirements

- Processor: Intel® Core™ i5, 2 cores (4 logical processors)
- RAM: 8 GB
- Hard disk: 256 GB (optional SSD)
- GPU (optional): NVIDIA Driver version >= 410.48 compatible
- Operating system: Ubuntu 18.04 and higher – x86_64 platform

2.2 Recommended requirements

- Processor: Intel® Core™ i7, 4 cores (8 logical processors)
- RAM: 16 GB
- Hard disk: 512 GB, SSD
- GPU (optional): NVIDIA® GeForce® GTX 1050 Ti, 4GB GDDR5
- Operating system: Ubuntu 18.04 and higher – x86_64 platform

3 Version history

Version 2.1.1

Released: 2022/08/18

- Used LPM module: LPM-v7.3.1-2022-08-16-Ubuntu-18.04-hasp
- Used MMR module: Eyedea-MMR-2.12.0-Ubuntu-18.04-x86_64-HASP
- Updated Sentinel license protection system: aksusbd-8.41.1
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 2.1.0

Released: 2022/07/28

- Fixed returning error message on SDK (usually license) error.
- Renamed Dockerfile and env.list variables.
- Limited number of LPM Detector CPU threads to 1 (optimized internally).
- Extended documentation.
- Updated to Java 12. Built with OpenJDK 12.0.1.
- Used LPM module: LPM-v7.3.0-2022-06-22-Ubuntu-18.04-hasp
- Used MMR module: Eyedea-MMR-2.12.0-Ubuntu-18.04-x86_64-HASP
- Updated Sentinel license protection system: aksusbd-8.41.1
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 2.0.4

Released: 2022/04/11

- Fixed LPM memory leak.
- Used LPM module: LPM-v7.2-2021-10-08-Ubuntu-16.04-hasp
- Used MMR module: Eyedea-MMR-2.11.0-Ubuntu-18.04-x86_64-HASP
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 2.0.3

Released: 2022/02/24

- Added support for GPU detection (currently available for LPM module 800).
- Reduced Sentinel license requirements.
- Fixed initialization of engines running on the CPU.
- LPM module selection moved to Dockerfile.
- Used LPM module: LPM-v7.2-2021-10-08-Ubuntu-16.04-hasp
- Used MMR module: Eyedea-MMR-2.11.0-Ubuntu-18.04-x86_64-HASP
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 2.0.2

Released: 2021/12/20

- Split MMR-VCMMGV and MMR-Color in settings, renamed Dockerfile and env.list variables.
- Used LPM module: LPM-v7.2-2021-10-08-Ubuntu-16.04-hasp
- Used MMR module: Eyedea-MMR-2.11.0-Ubuntu-18.04-x86_64-HASP
- Updated Sentinel license protection system: aksusbd-8.31.1
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 2.0.1

Released: 2021/07/16

- Added generation and variation attributes to MMR result.
- Used LPM module: LPM-v7.1-2020-04-16-Ubuntu-16.04-hasp
- Used MMR module: Eyedea-MMR-2.10.0-Ubuntu-18.04-x86_64-HASP
- Updated Sentinel license protection system: aksusbd-8.21.1
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 2.0.0

Released: 2021/02/02

- Added more detailed SDK information to serverSystemInfo response.
- Removed null / NaN fields from response.
- Used LPM module: LPM-v7.1-2020-04-16-Ubuntu-16.04-hasp
- Used MMR module: Eyedentify-VCL-2.9.0-Ubuntu-16.04-x86_64-HASP
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 2.0.0-BETA

Released: 2020/12/11

- Used LPM SDK for detection and OCR.
- Added option to specify a bounding box to reduce the input image area scanned by the detector.
- Simplified response for OCR / MMR module disabled (returning null).
- Removed countryID from anprResult response element.
- Used LPM module: LPM-v7.1-2020-04-16-Ubuntu-16.04-hasp
- Used MMR module: Eyedentify-VCL-2.9.0-Ubuntu-16.04-x86_64-HASP
- Embedded modules with GPU support are using CUDA 10.0.
  (CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 1.2.1

Released: 2020/04/14

- Used EyeScan module: EyeScanSDK-v3.8.3-DummyPack-Ubuntu-16.04-x86_64-HASP
- Used ANPR module: Eyedentify-ANPR-2.7.0-Ubuntu-16.04-x86_64-hasp
- Used MMR module: Eyedentify-VCL-2.7.2-Ubuntu-16.04-x86_64-hasp
- Embedded modules with GPU support are using CUDA 10.0.

(CUDA 10.0 requires the Linux x86_64 Driver version >= 410.48)

Version 1.2.0

Released: 2019/10/08

- Added support for bypassing the internal detector by specifying license plates positions in JSON format.
- HTML documentation updated.
- Used EyeScan module: EyeScanSDK-v3.7.0-DummyPack-Ubuntu-16.04-x86_64-HASP
- Used ANPR module: Eyedentify-ANPR-2.5.1-Ubuntu-16.04-x86_64-hasp
- Used MMR module: Eyedentify-VCL-2.6.0-Ubuntu-16.04-x86_64-hasp
- Embedded modules with GPU support are using CUDA 9.1.85.
  (CUDA 9.1.85 requires the Linux x86_64 Driver version >= 390.46)

Version 1.1.1

Released: 2019/07/01

- Added option to run the server with ANPR / MMR modules disabled.
- HTML documentation updated.
- Embedded modules with GPU support are using CUDA 9.1.85.
  (CUDA 9.1.85 requires the Linux x86_64 Driver version >= 390.46)

Version 1.1.0

Released: 2019/04/17

- Web interface added: System Info page and Recognition page.
- Updated to the Java 11. Built with OpenJDK 11.0.2.
- HTML documentation added.
- Used EyeScan module: EyeScanSDK-v3.4.1-DummyPack-Ubuntu-16.04-x86_64-HASP
- Used ANPR module: Eyedentify-ANPR-2.5.1-Ubuntu-16.04-x86_64-hasp
- Used MMR module: Eyedentify-VCL-2.5.1-Ubuntu-16.04-x86_64-hasp
- Embedded modules with GPU support are using CUDA 9.1.85.
  (CUDA 9.1.85 requires the Linux x86_64 Driver version >= 390.46)

Version 1.1.0-BETA

Released: 2019/01/11

- Added support for descriptor batch computation.
- Added support for GPU computation (ANPR, MMR).
- Embedded modules with GPU support are using CUDA 9.1.85.
  (CUDA 9.1.85 requires the Linux x86_64 Driver version >= 390.46)

Version 1.0.0

Released: 2018/06/13

- First release version.
- MMR and ANPR recognition REST API server application with basic functionality for still photos.
- Docker image creation scripts included.

4 Docker setup

4.1 GPU Support Prerequisites

If you do not intend to use the GPU, you can skip this chapter.

1. **Install GPU**

   Install GPU supporting NVIDIA CUDA® platform into your Docker host system. The Docker host is the system running the Docker daemon.

2. **Linux x86_64**

   Using the GPU for computation in the Docker environment requires the Docker to be installed and running on the Linux x86_64 system. Use the distribution which is supported by both the Docker and the NVIDIA Driver.

3. **Install NVIDIA Driver**

   Install NVIDIA Driver into the Docker host system. See the Version history for the minimal NVIDIA Driver version required by the CUDA® used in the current build. Write down the version of the NVIDIA Driver you are installing, the same driver version must be installed in the Docker image (see the chapter Building the Docker image).

4. **Install nvidia-container-runtime library**

   Install nvidia-container-runtime library into the Docker host system. As a root, run the following script:

   ```
   ./installNvidiaContainerRuntime
   ```

4.2 Sentinel license protection system

Installation

The SDK engines used by MMRRESTServer are protected with a standard third-party software licensing solution, *Sentinel LDK by Gemalto*.

1. **Install the 32-bit compatibility binaries**

   As a root, execute the following command to install the 32-bit compatibility binaries:

   ```
   apt-get install libc6:i386
   ```

   (Without compatibility binaries error “No such file or directory.” might appear.)

2. **Uncompress the package containing the Run-time Environment installer**

   Uncompress the aksusbd-*.tar.gz file.

3. **Uninstall prior Sentinel LDK Run-time Environment version**

   If you have installed a prior version of the Sentinel LDK Run-time Environment, as a root, run the following script from the uncompressed directory to uninstall it:

   ```
   ./dunst
   ```

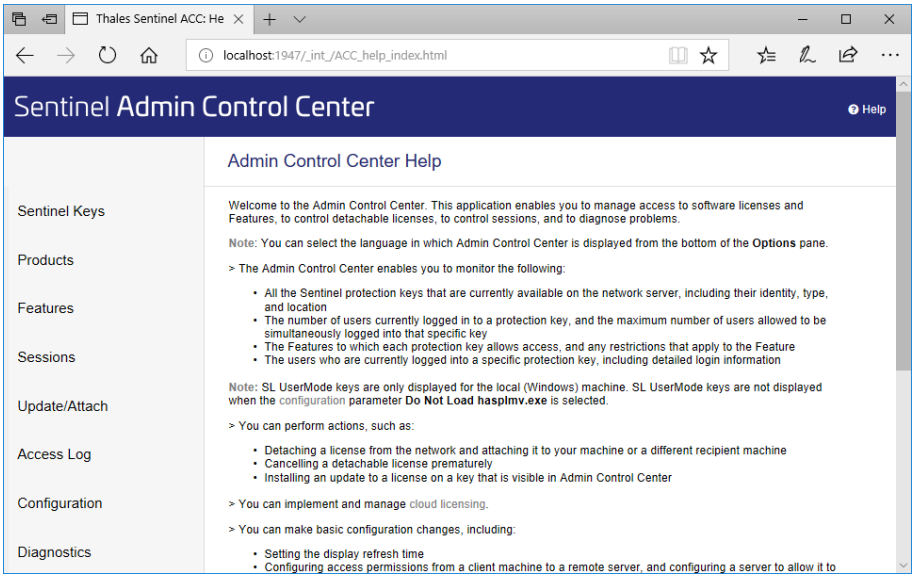
4. Install the Sentinel LDK Run-time Environment

As a root, run the following script from the uncompressed directory to install the Sentinel LDK Run-time Environment:

```
./dinst
```

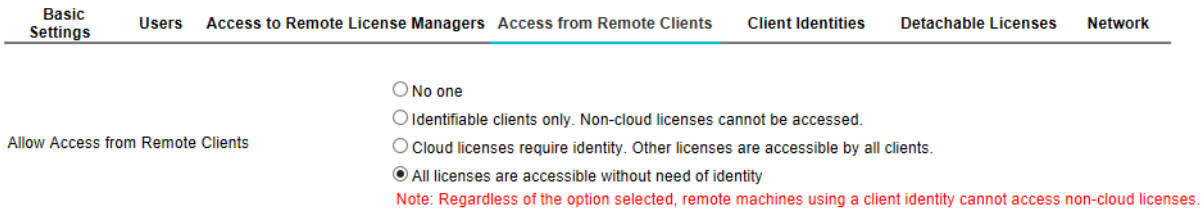
Verification

After the successful Sentinel License Server installation, open the address http://localhost:1947 in a web browser and check whether the license server is running. If the Sentinel Admin Control Center web application is displayed, the license server is running. License server can be configured to allow to connect clients to use available licenses (configured as a server) or it can connect to another license server (configured as a client).



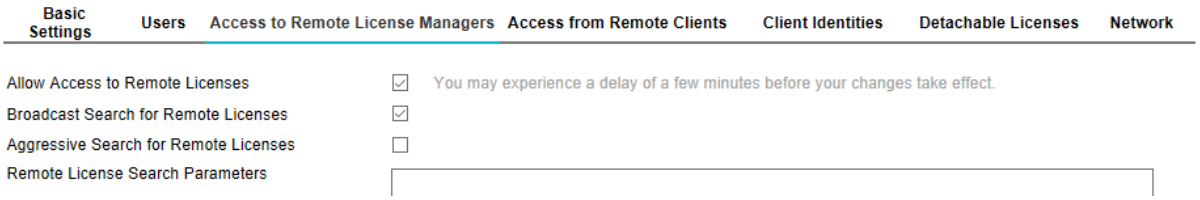
Server configuration

To set the license server as a license provider, please open the address in your web browser http://localhost:1947/_int_/config_from.html and choose the appropriate option from Allow Access from Remote Clients to allow other clients to connect to the network license key plugged in the server.



Client configuration

To set the license server as a client, open the address http://localhost:1947/_int_/config_to.html in a web browser and choose the Allow Access to Remote Licenses option. If the license provider is in another network than the computer, put the server's IP address into the field Remote License Search Parameters.



4.3 Building the Docker image

0. Prerequisites

Docker (https://www.docker.com) must be installed and Docker daemon must be running before creating the Docker image.

1. Set the Docker image variables (Dockerfile)

Instructions for building the Docker image are listed in Dockerfile which is located in the [PACKAGE]/ directory. Set the NVIDIA_DRIVER_VERSION variable (line 14) to install the proper version of the NVIDIA Driver (needed for GPU computation). The same version of the Driver must be installed on the Docker host system.

> The NVIDIA Driver version may be found out by running the following command:
>
> ```
> nvidia-smi
> ```

Set the NVIDIA_DRIVER_VERSION variable to empty to disable the NVIDIA Driver installation. In that case only CPU computation will be supported.

> Example of line 14 in Dockerfile which enables GPU computation:
>
> ```
> ENV NVIDIA_DRIVER_VERSION 410.104
> ```
>
> Example of line 14 in Dockerfile for CPU computation only:
>
> ```
> ENV NVIDIA_DRIVER_VERSION
> ```

To update LPM or MMR SDK, change the LPM_VERSION (line 17) or MMR_VERSION (line 22) variables, respectively. The appropriate *.tar.gz archive must be in the [PACKAGE]/ directory. The LPM module ID is specified on line 19, the MMR models are defined on lines 26 and 28; change them to the actual files in the SDK archive you want to use (the default MMR Vcmmgv model is MMR_VCMM_* which supports only view, category, make and model recognition; use MMR_VCMMGV_* to also support generation and variation recognition).

2. Run the build script (buildDocker)

Use the buildDocker script located in the [PACKAGE]/ directory to build the Docker image:

```
./buildDocker
```

> The Docker image is built using the following command:
>
> ```
> docker build -t mmrrest .
> ```
>
> The -t option specifies the name of the new Docker image.

4.4 Running the Docker image

1. Set the application variables (env-GPU.list or env-CPU.list)

The application settings (license server address, number of threads, ...) are loaded from the system environment variables by the application. The supported variables are defined in the env-GPU.list and env-CPU.list files (the first one is intended to use the GPU for computation, the other uses the CPU). One of these files is passed to the Docker during the image initialization and its variables are available in the Docker container as system environment variables.

In the following overview, <ENGINE> stands for: LPM_DETECTOR, LPM_OCR, MMR_VCMMGV and MMR_COLOR (if not specified otherwise).

HASP_REMOTE_SERVERADDR	The address of the Sentinel License Manager server with the valid license. **The initial value should be changed.**
<ENGINE>_NUMTHREADS	The number of engine's threads to initialize. Set the variable to 0 to disable the given engine. Set -1 to select the number of threads automatically (1 for CPU, or 1 for each <ENGINE>_GPU_ID in case of GPU computation mode). **Note:** When running on CPU, the number of LPM Detector threads is limited to 1 because it internally uses the optimal number of CPU threads.
<ENGINE>_COMPUTATION_MODE	Specifies whether the given engine runs on CPU, or GPU. (Certain older LPM Detectors support only CPU.)
<ENGINE>_GPU_ID	Relevant only for GPU computation. Specifies the 0-based index or indexes of GPU devices to compute on. If more GPUs are available, you can assign a comma separated list of indexes to a single engine (e.g.: LPM_OCR_GPU_ID=0,1), or distribute them among multiple engines (e.g.: MMR_VCMMGV_GPU_ID=0 MMR_COLOR_GPU_ID=1).
LPM_DETECTOR_MIN_HORIZONTAL_RESOLUTION	Specify the camera minimal and maximal horizontal resolution in number of pixels per meter, respectively. See the example below the table. Relevant only for certain modules.
LPM_DETECTOR_MAX_HORIZONTAL_RESOLUTION	

Check the SDK Engine Overview chapter to select the appropriate SDK engines for your use case.

Example:

Let's have the env-GPU.list file with the following configuration (only the relevant variables listed for brevity):

```
HASP_REMOTE_SERVERADDR=127.0.0.1
LPM_DETECTOR_NUMTHREADS=1
LPM_DETECTOR_COMPUTATION_MODE=CPU
LPM_DETECTOR_MIN_HORIZONTAL_RESOLUTION=240
LPM_DETECTOR_MAX_HORIZONTAL_RESOLUTION=4000
LPM_OCR_NUMTHREADS=1
LPM_OCR_COMPUTATION_MODE=GPU
LPM_OCR_GPU_ID=0
MMR_VCMMGV_NUMTHREADS=0
MMR_COLOR_NUMTHREADS=0
```

The license key with the valid licenses must be plugged in the current machine as the Sentinel License Manager address is the localhost.

There will be two running SDK engines: LPM Detector in one CPU thread and LPM OCR in one thread running on the GPU. The MMR engines will be disabled.

The European 1-line license plate detector will detect license plates (520 mm wide) if their width in the image is in the range of approximately 125 – 2080 pixels (because 240 × 0.52 = 125 and 4000 × 0.52 = 2080).

2. Run the image (runDockerGPU or runDockerCPU)

Use the runDockerGPU or runDockerCPU script located in the [PACKAGE]/ directory to run the built Docker image with / without the support for GPU computation. Edit these scripts if needed.

runDockerGPU – support for GPU computation:

```
docker run --gpus device=0 --env-file env-GPU.list -p 8080:8080 mmrrest
```

runDockerCPU – CPU computation only:

```
docker run --env-file env-CPU.list -p 8080:8080 mmrrest
```

Notes:

The --env-file option specifies the file with the environment variables.

The -p option publishes port 8080 in the container and maps it to the host's port 8080.

The --gpus option specifies GPU device(s) used by the container. Possibilities:

- Use a GPU device using its index: --gpus device=0
- Use multiple GPU devices using their indexes: --gpus '"device=2,3"'
- Use all available GPU devices: --gpus all

The server tries to initialize all engine instances specified in the env-*.list file. Check the error output for details if there are problems starting any engine. After launching the Docker container, the application is accessible at:

```
http://[MACHINE_IP]:8080/MMRRESTServer/
```

You can check the status of the engines on the application home page or by requesting the serverSystemInfo. "DISABLED" means that no thread was requested for the given engine, "FAILED TO START" means that all requested threads of the given engine could not be started; in both cases, the server handles requests and returns data obtained by other running engines.

The engine status never changes when the container is started. If an engine encounters a license problem after the successful initialization, its status will still be "RUNNING", but the response to a request for that engine will have a status code of 500 with a message indicating the problematic engine (e.g.: "Error during OCR processing. Computation error. Please check your licenses.").

5 REST API Documentation

5.1 SDK Engine Overview

Which SDK engines do you need? Check the following table.

Use Case	SDK Engines
License plate detection	LPM Detector
License plate detection + OCR	LPM Detector, LPM OCR
License plate detection + OCR + Vehicle recognition	LPM Detector, LPM OCR, MMR Vcmmgv, MMR Color
Vehicle recognition (without detection; *detections must be provided in the request*)	MMR Vcmmgv, MMR Color

Note: License plate detection + OCR = ANPR

LPM Detector detects license plates.

LPM OCR recognizes the text and type of license plates detected by LPM Detector (both LPM engines are needed).

MMR Vcmmgv can recognize the view, category, make, model, generation and variation of vehicles specified by their license plate. Either LPM Detector and LPM OCR are needed, or the license plate detections must be provided in the request. **Note**: By default, only the view, category, make and model are recognized; to support also generation and variation recognition, change the MMR Vcmmgv binary model in Dockerfile to MMR_VCMMGV_*.

MMR Color recognizes the color of vehicles specified by their license plate. Either LPM Detector and LPM OCR are needed, or the detections must be provided in the request.

5.2 Response Status Codes

The following table lists all response status codes MMRRESTServer returns.

Status Code	Meaning
200	Success.
400	Bad Request. The request was invalid. The error message provides the details.
500	Internal Server Error. Most of these errors are caused by a licensing issue. The error message provides the details.

5.3 System Info

System info contains information about system resources and SDK engines.

Main web page with real time server monitoring

Endpoint:	/
HTTP Method:	GET

Consumes Media Type:	-
Produces Media Type:	text/html
URL Parameters:	-
Data Parameters:	-

CURL Command Example:

```
curl -X GET -H "Accept: text/html" http://localhost:8080/MMRRESTServer/
```

Response Example:

Text output with server monitoring information

Endpoint:	/
HTTP Method:	GET
Consumes Media Type:	-
Produces Media Type:	text/plain
URL Parameters:	-
Data Parameters:	-

CURL Command Example:

```
curl -X GET -H "Accept: text/plain" http://localhost:8080/MMRRESTServer/
```

Response Example:

```
System:
 - Average load: 16.37%
 - Used memory: 46.82% (14.67 GB)

GPU 0 (Graphics Device):
 - Utilization: 3%
 - Free memory: 4822 / 5911 MB

Detection:
 - Engine: LPM Detector
 - Version: v7.2-2021-10-08-Ubuntu-16.04-hasp
 - Module ID: 800
 - Module version: 800-frontal.lp-eu-v7.14
 - Computation mode: GPU
 - Status: RUNNING
 - Processing batch size: 1
 - Number of running threads: 1
 - Number of waiting tasks: 0
 - Number of processed tasks: 6

Classification:
 - Engine: LPM OCR
 - Version: v7.2-2021-10-08-Ubuntu-16.04-hasp
 - Module ID: 800
 - Module version: 800-frontal.lp-eu-v7.14
 - Computation mode: GPU
 - Status: RUNNING
 - Processing batch size: 1
 - Number of running threads: 1
 - Number of waiting tasks: 0
 - Number of processed tasks: 10

Classification:
 - Engine: MMR Vcmmgv
 - Version: 2.11.0-Ubuntu-18.04-x86_64-HASP
 - Module: MMR_VCMMGV_PREC_2021Q4.dat
 - Module version: 20211101
 - Computation mode: GPU
 - Status: RUNNING
 - Processing batch size: 1
 - Number of running threads: 1
 - Number of waiting tasks: 0
 - Number of processed tasks: 10

Classification:
 - Engine: MMR Color
 - Version: 2.11.0-Ubuntu-18.04-x86_64-HASP
 - Module: MMR_COLOR_PREC_2021Q4.dat
 - Module version: 20211105
 - Computation mode: GPU
 - Status: RUNNING
 - Processing batch size: 1
 - Number of running threads: 1
 - Number of waiting tasks: 0
 - Number of processed tasks: 10
```

Get detailed server system info

Endpoint:	/
HTTP Method:	GET

Consumes Media Type:	-
Produces Media Type:	application/json
URL Parameters:	-
Data Parameters:	-

CURL Command Example:

```
curl -X GET http://localhost:8080/MMRRESTServer/serverSystemInfo
```

Response Example:


```
{
  "timestamp": 1654694188241,
  "systemLoadAverage": 18.36734693877551,
  "systemUsedMemory": 46.81931794132052,
  "systemUsedMemoryBytes": 15748575232,
  "processingServersInfosList": [
    {
      "batchSize": 1,
      "computationMode": "GPU",
      "engineName": "LPM",
      "engineType": "Detector",
      "engineVersion": "v7.2-2021-10-08-Ubuntu-16.04-hasp",
      "moduleId": 800,
      "moduleVersion": "800-frontal.lp-eu-v7.14",
      "numComputingThreads": 1,
      "numProcessedTasks": 6,
      "numWaitingTasks": 0,
      "status": "RUNNING"
    },
    {
      "batchSize": 1,
      "computationMode": "GPU",
      "engineName": "LPM",
      "engineType": "OCR",
      "engineVersion": "v7.2-2021-10-08-Ubuntu-16.04-hasp",
      "moduleId": 800,
      "moduleVersion": "800-frontal.lp-eu-v7.14",
      "numComputingThreads": 1,
      "numWaitingTasks": 0,
      "numProcessedTasks": 10,
      "status": "RUNNING"
    },
    {
      "batchSize": 1,
      "computationMode": "GPU",
      "engineName": "MMR",
      "engineType": "Vcmmgv",
      "engineVersion": "2.11.0-Ubuntu-18.04-x86_64-HASP",
      "moduleName": "MMR_VCMMGV_PREC_2021Q4.dat",
      "moduleVersion": "20211101",
      "numComputingThreads": 1,
      "numWaitingTasks": 0,
      "numProcessedTasks": 10,
      "status": "RUNNING"
    },
    {
      "batchSize": 1,
      "computationMode": "GPU",
      "engineName": "MMR",
      "engineType": "Color",
      "engineVersion": "2.11.0-Ubuntu-18.04-x86_64-HASP",
```

```
      "moduleName": "MMR_COLOR_PREC_2021Q4.dat",
      "moduleVersion": "20211105",
      "numComputingThreads": 1,
      "numWaitingTasks": 0,
      "numProcessedTasks": 10,
      "status": "RUNNING"
    }
  ],
  "gpuInfoList": [
    {
      "id": 0,
      "name": "Graphics Device",
      "totalMemoryMB": 5911,
      "freeMemoryMB": 4822,
      "gpuUtilizationPerc": 3
    }
  ]
}
```


Response Definitions:

The following table describes each item in the response.

Response item	Description	Data type
timestamp	Current system time in milliseconds.	Integer
systemLoadAverage	Recent CPU usage for the whole system in percent.	Decimal number
systemUsedMemory	Amount of physical memory used by any process in percent.	Decimal number
systemUsedMemoryBytes	Amount of physical memory used by any process in bytes.	Integer
processingServersInfosList	Information about the SDK engines.	Array of objects
processingServersInfosList/ batchSize	Size of the computation batch.	Integer
processingServersInfosList/ computationMode	Type of processing unit used for computation. Either "CPU", or "GPU".	String
processingServersInfosList/ engineName	SDK engine name.	String
processingServersInfosList/ engineType	SDK engine type.	String
processingServersInfosList/ engineVersion	SDK engine version.	String
processingServersInfosList/ moduleId	DK engine module ID.	Integer
processingServersInfosList/ moduleName	SDK engine module name.	String
processingServersInfosList/ moduleVersion	SDK engine module version.	String

processingServersInfosList/numComputingThreads	Number of initialized engine's threads.	Integer
processingServersInfosList/numWaitingTasks	Number of queued computation tasks to be processed.	Integer
processingServersInfosList/numProcessedTasks	Number of processed computation tasks.	Integer
processingServersInfosList/status	SDK engine status. Either "RUNNING" (at least 1 thread successfully initialized), "DISABLED" (no thread requested), or "FAILED TO START" (all requested threads could not be started).	String
gpuInfoList	Information about available GPU devices.	Array of objects
gpuInfoList/id	GPU device ID.	Integer
gpuInfoList/name	GPU device name.	String
gpuInfoList/totalMemoryMB	Total amount of physical GPU device memory in megabytes.	Integer
gpuInfoList/freeMemoryMB	Amount of free physical GPU device memory in megabytes.	Integer
gpuInfoList/gpuUtilizationPerc	Utilization of the GPU device in percent.	Integer

5.4 Recognition

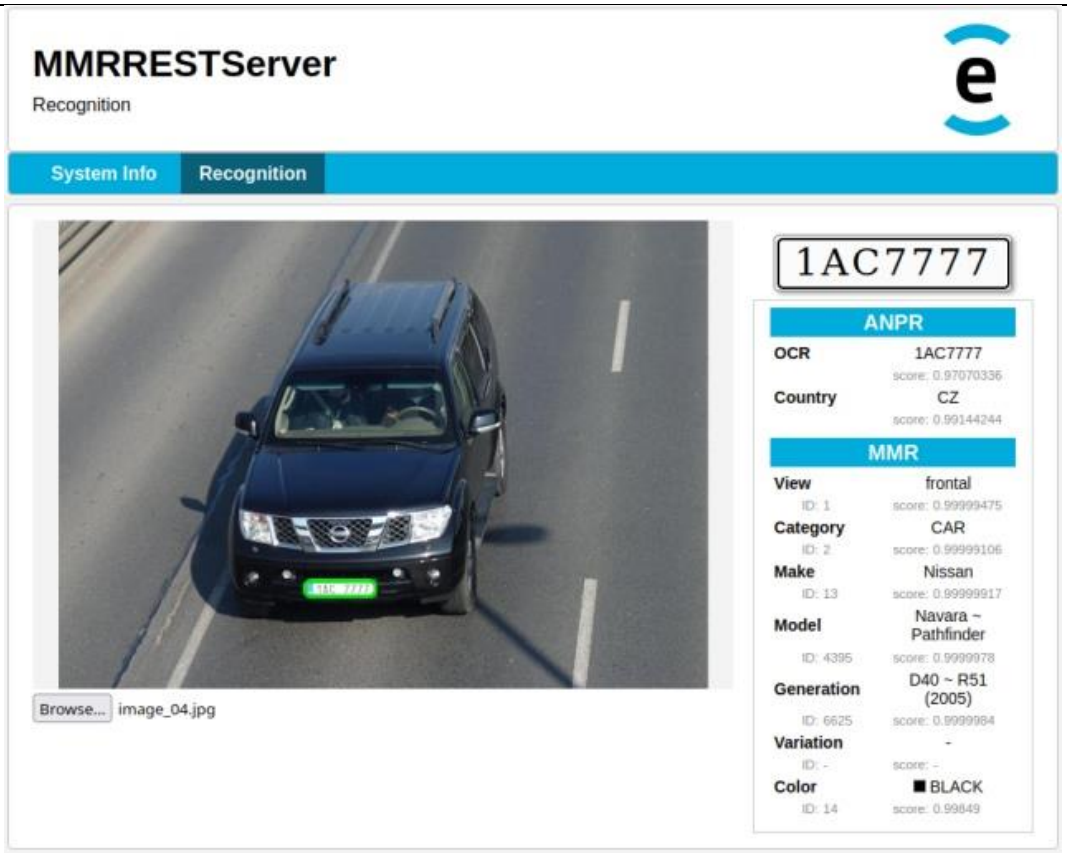
Recognition provides information about recognized license plates and corresponding vehicles in the input image.

Web page with image upload and recognition functionality using the license plate detector

Endpoint:	/recognition
HTTP Method:	GET
Consumes Media Type:	-
Produces Media Type:	text/html
URL Parameters:	-
Data Parameters:	-
CURL Command Example:	

```
curl -X GET http://localhost:8080/MMRRESTServer/recognition
```

Response Example:



Notes:

License plate detector is available for all LPM modules. Both ANPR and MMR results are displayed (for LPM Detector, LPM OCR, MMR Vcmmgv and MMR Color engines running).

MMR Vcmmgv engine uses a binary model also supporting generation and variation (MMR_VCMMGV_*).

Detect and read license plates in the input image and recognize the corresponding vehicles

Endpoint:	/recognition
HTTP Method:	POST
Consumes Media Type:	multipart/form-data
Produces Media Type:	application/json
URL Parameters:	-
Data Parameters:	file – form data parameter containing input image boundingBox – *(optional)* form data parameter containing a JSON object that defines the area of the input image to be scanned by the detector

The bounding box parameter is defined by its corners coordinates: topLeftX, topLeftY, topRightX, topRightY, bottomLeftX, bottomLeftY, bottomRightX and bottomRightY (decimal numbers).

Example:

```
{
  "topLeftX": 176,
  "topLeftY": 362,
  "topRightX": 1234,
  "topRightY": 362,
  "bottomLeftX": 176,
  "bottomLeftY": 1000,
  "bottomRightX": 1234,
  "bottomRightY": 1000
}
```

For the sides of a bounding box parallel to the sides of the image, a simplified form of bounding box defined by its sides can be used instead: leftX, topY, rightX and bottomY (decimal numbers).

Example:

```
{
  "leftX": 176,
  "topY": 362,
  "rightX": 1234,
  "bottomY": 1000
}
```

lpDetection – *(optional)* form data parameter containing a JSON object with an array of license plate positions (for MMR computation only)

The license plate detection parameter is defined as an array of "center, angle, scalePPM" structures which define the position of a license plate in the image for MMR computation. If this parameter is used, ANPR (automatic detection and OCR) will not run during the input image processing.

center – 2D point (structure of x and y decimal numbers) defining the center of the license plate. The *center* point must be inside the input image.

angle – Clockwise correction of the source image in degrees (decimal number).

scalePPM – The real-world scale in pixels per meter of the input image in the *center* position (decimal number). The *scalePPM* defines the real size of the license plate in the image and does not consider the margin added by the detector.

Example:

```
[
  {
    "center": {
      "x": 176.5,
      "y": 362.0
    },
    "angle": -5.1,
    "scalePPM": 214.3
  },
  ...
]
```

CURL Command Examples:

Use automatic detection:

```
curl -X POST
     -H "Content-Type: multipart/form-data"
     -F "file=@image.jpg"
```

```
     http://localhost:8080/MMRRESTServer/recognition
```

Use automatic detection with a bounding box in JSON file:

```
curl -X POST
     -H "Content-Type: multipart/form-data"
     -F "file=@image.jpg"
     -F "boundingBox=@box.json"
     http://localhost:8080/MMRRESTServer/recognition
```

Use automatic detection with a bounding box in JSON string:

```
curl -X POST
     -H "Content-Type: multipart/form-data"
     -F "file=@image.jpg"
     -F "boundingBox={\"topLeftX\":176,\"topLeftY\":362,\"topRightX\":1234,
\"topRightY\":362,\"bottomLeftX\":176,\"bottomLeftY\":1000,
\"bottomRightX\":1234,\"bottomRightY\":1000}"
     http://localhost:8080/MMRRESTServer/recognition
```

Use automatic detection with a simplified form of bounding box in JSON string:

```
curl -X POST
     -H "Content-Type: multipart/form-data"
     -F "file=@image.jpg"
     -F "boundingBox={\"leftX\":176,\"topY\":362,\"rightX\":1234,
\"bottomY\":1000}"
     http://localhost:8080/MMRRESTServer/recognition
```

Detections in JSON file:

```
curl -X POST
     -H "Content-Type: multipart/form-data"
     -F "file=@image.jpg"
     -F "lpDetection=@detections.json"
     http://localhost:8080/MMRRESTServer/recognition
```

Detections in JSON string:

```
curl -X POST
     -H "Content-Type: multipart/form-data"
     -F "file=@image.jpg"
     -F "lpDetection=[{\"center\":{\"x\":176.5,\"y\":362.0},\"angle\":-5.1,
\"scalePPM\":214.3}]"
     http://localhost:8080/MMRRESTServer/recognition
```

Notes:

Content type of the request is **multipart/form-data**, where the inputs are stored in the form data fields.

Input image is referenced with filepath (starting with @) in the form field **file**.

Optional parameters **boundingBox** and **lpDetection** can be referenced with filepath (starting with @, referencing a text file containing JSON string) or can contain the whole JSON as a string (special characters must be escaped).

Response Examples:

Automatic license plate detection, OCR and vehicle recognition:

```
[
  {
    "lpDetection":{
      "corners": [
        {
          "x": 1234.56,
          "y": 234.56
        },
        {
```

```
          "x": 1289.10,
          "y": 234.56
        },
        {
          "x": 1289.10,
          "y": 244.67
        },
        {
          "x": 1234.56,
          "y": 244.67
        }
      ],
      "width": 54.54,
      "height": 10.11,
      "angles": [ 5.0, 0.0, 0.0 ],
      "score": 0.7891011,
      "position": {
        "center": {
          "x": 1261.83,
          "y": 239.615
        },
        "angle": -5.0,
        "scalePPM": 93.65
      }
    },
    "anprResult":{
      "country": "CZ",
      "countryScore": 0.987654321,
      "ocrText": "1A23456",
      "ocrTextScore": 0.953135953
    },
    "mmrResult":{
      "category": "CAR",
      "categoryID": 2,
      "categoryScore": 0.9999992,
      "color": "GRAY",
      "colorID": 7,
      "colorScore": 0.9012345,
      "generation": "Mk VI (2008)",
      "generationID": 3664,
      "generationScore": 0.9999988,
      "make": "VW",
      "makeID": 43,
      "makeScore": 0.9999914,
      "model": "Golf",
      "modelID": 3544,
      "modelScore": 0.9234567,
      "view": "frontal",
      "viewID": 1,
      "viewScore": 0.9999999
    }
  }
]
```


License plate detection provided in the request:


```
[
  {
    "lpDetection":{
      "angles": [ 5.0, 0.0, 0.0 ],
      "position": {
        "center": {
          "x": 1261.83,
          "y": 239.615
        },
        "angle": -5.0,
        "scalePPM": 93.65
      }
```

```
    },
    "mmrResult":{
      "category": "CAR",
      "categoryID": 2,
      "categoryScore": 0.9999992,
      "color": "GRAY",
      "colorID": 7,
      "colorScore": 0.9012345,
      "generation": "Mk VI (2008)",
      "generationID": 3664,
      "generationScore": 0.9999988,
      "make": "VW",
      "makeID": 43,
      "makeScore": 0.9999914,
      "model": "Golf",
      "modelID": 3544,
      "modelScore": 0.9234567,
      "view": "frontal",
      "viewID": 1,
      "viewScore": 0.9999999
    }
  }
]
```


Notes:

When the **lpDetection** form field is provided in the request, the response does **not** contain the **anprResult** element and **lpDetection** element contains only the fields from the request.

If the **LPM Detector** engine is not running and the **lpDetection** form field is not provided in the request, an empty array is returned in the response.

The **anprResult** element is returned in the response only if the **LPM Detector** and **LPM OCR** engines are running and the **lpDetection** form field is **not** provided in the request.

To obtain the **mmrResult** element in the response, running the **MMR Vcmmgv** and/or **MMR Color** engine is required. Additionally, either the **LPM Detector** and **LPM OCR** engines must be running, or the **lpDetection** form field must be provided in the request.

Response Definitions:

The following table describes each item in the response.

Response item	Description	Data type
lpDetection	Information about the detected license plate location in the input image.	Object
lpDetection/corners	An array of top-left, top-right, bottom-right and bottom-left corners of the detected license plate.	Array of objects
lpDetection/corners/x	The horizontal coordinate of the license plate corner within the image.	Decimal number
lpDetection/corners/y	The vertical coordinate of the license plate corner within the image.	Decimal number
lpDetection/width	The width of the detected license plate in pixels.	Decimal number
lpDetection/height	The height of the detected license plate in pixels.	Decimal number

lpDetection/**angles**	The license plate orientation - roll, pitch, yaw angles in degrees. Currently, only the first array item, roll, is evaluated.	Array of decimal numbers
lpDetection/**score**	The license plate detection confidence factor. Range 0 – 1.	Decimal number
lpDetection/**position**	The "center, angle, scalePPM" structure defining the license plate position in the image for MMR computation.	Object
lpDetection/position/**center**	The coordinates of the license plate center within the image.	Object
lpDetection/position/center/**x**	The horizontal coordinate of the license plate center within the image.	Decimal number
lpDetection/position/center/**y**	The vertical coordinate of the license plate center within the image.	Decimal number
lpDetection/position/**angle**	Clockwise correction of the source image in degrees.	Decimal number
lpDetection/position/**scalePPM**	The real-world scale in pixels per meter of the image in the center position. The scalePPM defines the real size of the license plate in the image and does not consider the margin added by the detector. The value is calculated by the LPM OCR engine.	Decimal number
anprResult	Information about the recognized license plate.	Object
anprResult/**country**	The international license plate country code. If the value is "UNK", then it was recognized as a false positive detection.	String
anprResult/**countryScore**	The confidence factor for "country" prediction. Range 0 – 1.	Decimal number
anprResult/**ocrText**	The license plate text recognized by the OCR.	String
anprResult/**ocrTextScore**	The confidence factor for the OCR result. Range 0 – 1.	Decimal number
mmrResult	Information about the recognized vehicle attributes.	Object
mmrResult/**view**	The recognized vehicle view, either "FRONTAL", or "REAR".	String
mmrResult/**viewID**	ID of the recognized vehicle view.	Integer
mmrResult/**viewScore**	The confidence factor for the view result. Range 0 – 1.	Decimal number
mmrResult/**category**	The recognized vehicle category, e.g., "BUS", "CAR", "HVT", ... For the full list of possible categories and their definition, check the Eyedea MMR SDK documentation.	String
mmrResult/**categoryID**	ID of the recognized vehicle category.	Integer

mmrResult/categoryScore	The confidence factor for the category result. Range 0 – 1.	Decimal number
mmrResult/make	The recognized vehicle manufacturer, e.g., "VW", "Ford", "Fiat", ... For the full list of possible categories, check the Eyedea MMR SDK documentation.	String
mmrResult/makeID	ID of the recognized vehicle manufacturer.	Integer
mmrResult/makeScore	The confidence factor for the make result. Range 0 – 1.	Decimal number
mmrResult/model	The recognized vehicle model (vehicle instance defined by a bodywork), e.g., "Golf", "Mondeo", "500", ...	String
mmrResult/modelID	ID of the recognized vehicle model.	Integer
mmrResult/modelScore	The confidence factor for the model result. Range 0 – 1.	Decimal number
mmrResult/generation	The recognized vehicle generation (vehicle mark and first model year), e.g., "Mk VI (2019)", "Mk I (2020)", ...	String
mmrResult/generationID	ID of the recognized vehicle generation.	Integer
mmrResult/generationScore	The confidence factor for the generation result. Range 0 – 1.	Decimal number
mmrResult/variation	The recognized vehicle variation (vehicle trim level and/or body type), e.g., "AMG", "AMG-Line SUV", "Coupe", ...	String
mmrResult/variationID	ID of the recognized vehicle variation.	Integer
mmrResult/variationScore	The confidence factor for the variation result. Range 0 – 1.	Decimal number
mmrResult/color	The recognized vehicle color, e.g., "BLUE", "GRAY", "RED", ...	String
mmrResult/colorID	ID of the recognized vehicle color.	Integer
mmrResult/colorScore	The confidence factor for the color result. Range 0 – 1.	Decimal number