
# LPM v3.x SDK

## Developer's Guide

## Contact:

Eyedea Recognition, s.r.o.

Office address:
Vyšehradská 320/49
128 00, Prague 2
Czech Republic.

web: http://www.eyedea.cz
E-mail: info@eyedea.cz

# Table of Contents

# Product description

***LPM v3.x SDK*** is a library designed to provide a comfortable car license plate detection and character recognition (OCR) within the input images. It defines an interface between the client's software and ***Eyedea Recognition, Ltd.*** detection engines (modules). This special API allows simple module administrations and updates without any need for changes in the client's software.

Each client receives a **FTP** account automatically created at ***Eyedea Recognition's*** server. This FTP access serves as two-way communication between the client and ***Eyedea Recognition, Ltd.*** Clients have the possibility to regularly upload data samples (or problematic data) on the ftp server and consecutively receive the corresponding LPR engine updates. This systematic approach enables to adapt LPR engines continuously to client's specific data and to ensure the best possible performance. Later it is used for regular verification of result statistics.

## Technical details

| | |
|---|---|
| Platforms supported: | *Windows 7, Windows XP and Vista*<br>*(for linux/unix distribution contact Eyedea Recognition, Ltd.)* |
| Release version: | v3.x |
| Release date: | *2012, Sep 15th* |

# Distribution Contents

- **\root**
  - **\LPM-v3.x**
    *LPM library (version v3.x). This library is used as API between the client's solution and LPR engines (modules).*
    - **\docs**

      *Doxygen documentation of the LPM API*
    - **\example**

      *A simple example of MS Visual C++ project demonstrating a basic usage of the LPM-v3.x library*
    - **\include**

      *Header files to LPM-v3.x library.*
    - **\lib**

      *Release LPM-v3.x library binary files.*
  - **\modules-v3**

    *A directory containing plug-in modules. By a module it is meant an independent stand-alone application for object detection and recognition with standardized API. These modules are accessed through LPM API.*
  - **\LPM-v3.x_SDK_Develop_Guide.pdf**

    *The LPM v3.x SDK developer's guide*

# Installation

### *A) Install LPM-v3.x SDK*

Extract ***LPM-v3.x*** directory from the CD or a downloaded zip file to your custom location.

### *B) Install modules*

It is recommended to copy/extract module files to a directory containing the ***\LPM-v3.x*** to let the example project to work properly.

# LPM - Overview

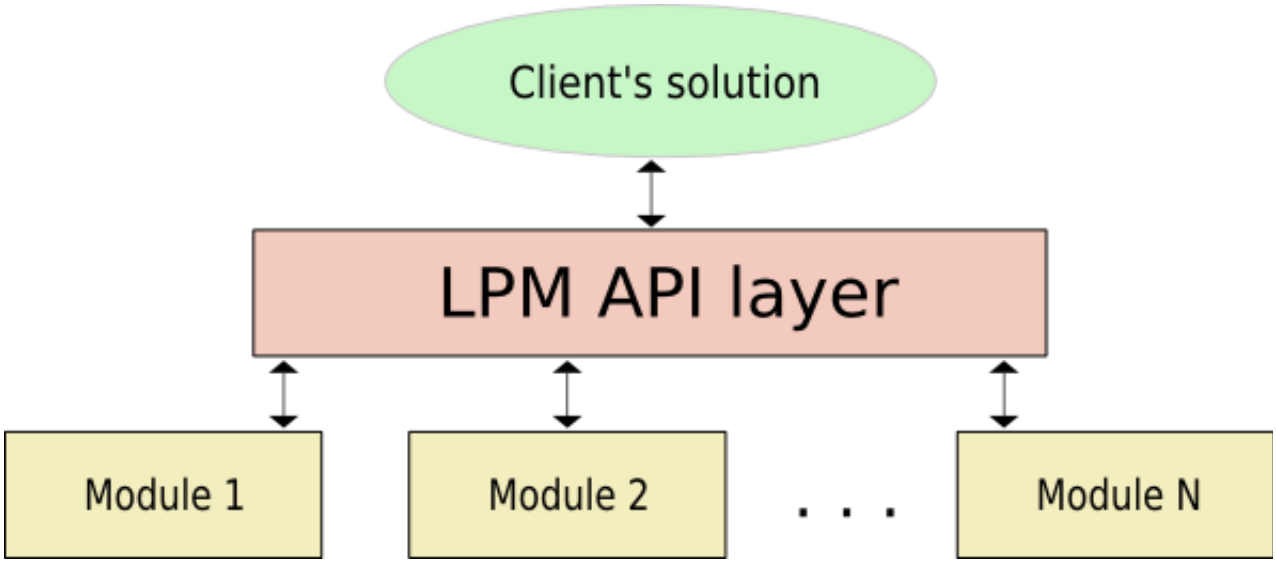


***Fig. 1 LPM API architecture***

LPM library is used as an interface between a client's solution and Eyedea Recognition LPR modules. By a module it is meant an independent stand-alone application (plug-in) for object detection and recognition with standardized API. LPM library allows to work with multiple modules (engines) simultaneously and makes module administration and updating easy.

### LPM library includes:

| | |
|---|---|
| `LPM-v3.x\include` | … header files defining LPM API |
| `LPM-v3.x\libs` | … both shared (lpm.dll) and static (lpm.lib) library files |
| `LPM-v3.x\docs` | … doxygen html documentation |
| `LPM-v3.x\example` | … MSVC example project |

# LPM - Tutorial

This tutorial gives an overview of using LPM functions in client's applications. For detailed function description see the html documentation (`\Cdroot\LPM-v3.x\docs\index.html`) or the included example project.

### 1. Initializing the LPM layer:

The LPM layer is initialized using the `lpmInit()` function which searches the given directory (e.g. "`LPM-v3.x\modules-v3\`") for installed modules while assigning them unique zero-based indices. Assigned indices range from a zero to the number of installed modules. A pointer to the lpm-state structure is returned on success.

Example:

```
void * pState;
pState = lpmInit("../../modules-v3/");
```

The number of available/installed modules is returned by `lpmGetNumAvlbModules()` function.

Example:

```
int numAvlbMod = lpmGetNumAvlbModules(pState);
printf("Number of available modules: %d\n",numAvlbMod);
```

### 2. Querying module properties:

The module properties structure is accessed by the `lpmGetModuleInfo()` function.

Example:

```
int idx;
for (idx = 0; idx < numAvlbMod; idx++)
{
   LpmModuleInfo *  pInfo = lpmGetModuleInfo(pState,idx);

   /* Each module has its own ID and this is a way
       how to get it. */
    printf("  Module ID     : %d\n", pInfo->Id);

    /* Another module properties... */
    printf("  Module name   : %s\n", pInfo->Name);
    printf("  Module path   : %s\n", pInfo->Path);
    printf("  Module date   : %s\n", pInfo->Date);
    printf("  Module version: %d.%d\n\n", pInfo->Version,
      pInfo->Subversion);
}
```

Furthermore, examine the result of the bitwise AND operation between `LpmModuleInfo::prop`

variable and the `LPM_XXX flags` defined in `lpm_type.h` to get more detailed information about a specific module.

### 3. Load/activate a module of a given index:

Use `lpmLoadModule()` function to load a module of a given index.

Example:

```
lpmLoadModule(pState,idx);
```

A module becomes active (i.e. module functions can be called) once the module is loaded. The LPM layer allows to activate and to work with multiple modules simultaneously.

The appropriate module index `idx` can be retrieved from the module ID and version using the `lpmGetModuleIdx()` function.

Example:

```
/* get module index from module ID, version and subversion */;
int iModuleID = 21, iModuleVersion = 2, iModuleSubversion = 1;

int idx = lpmGetModuleIndex(pState,iModuleID,
              iModuleVersion,iModuleSubversion);

/* or get idx of the latest available module of given ID */;

int iModuleID = 21;
int idx = lpmGetModuleIndex(pState,iModuleID,0,0);
```

Alternatively:

```
/* get index from module name if it is known in advance */;
int idx = lpmGetModuleIndexByName(pState,"some_module_name");
```

### 4. Reading image data from a file:

Image data can be loaded from a file using function `lpmReadImage_uc()`. Only 8bit jpegs, tiffs and pngs are supported. Image data are stored in `Lpm_Matrix_Uc` structure (see lpm_type.h, or html documentation for the definition) continuously in row-wise order (row by row), 8-bit per pixel.

Example:

```
/* read image from a file */
Lpm_Matrix_Uc* pImage = lpmReadImage_uc("images/frontal_A4K.jpg");
```

### 5. Copying image data from memory:

Alternatively, image data can be copied into the `Lpm_Matrix_Uc` structure directly from the memory via the following example:

```
/* allocate Lpm_Matrix_Uc structure */
Lpm_Matrix_Uc* pImage = lpmAllocMatrix_uc(height,width);

/* copy image data from pSrcData */
memcpy( pImage->data,pSrcData,sizeof(unsigned char)*height*width );
```

## 6. Running license plate detection:

Next, the license plate detection function `lpmRunDetModule()` can be called. This function scans the selected image area and searches for license plates. The detection result (i.e. positions of all detected license plates) is returned in `LpmDetResult` structure (see lpm_type.h, or the html documentation for the definition).

Example 1:

```
/* define a scanning area */
LpmBoundingBox BoundingBox;
BoundingBox.TopLeftCol=0;
BoundingBox.TopLeftRow=0;
BoundingBox.BotRightCol=pImage->no_cols-1;
BoundingBox.BotRightRow=pImage->no_rows-1;

/* run LP detection on a given image */
LpmDetResult *pDetRes;
pDetRes = lpmRunDetModule(pState,idx,pImage,&BoundingBox);

/* print the results */
printf("Number of detections: %d\n",pDetRes->NumDetections);
```

If the license plate rotation, skew or size is out of the allowed range (see `LpmModuleInfo`) then the detection routine `lpmRunDetModuleW()` should be used.

Example 2:

```
/* set license plate rotation, skew and size parameters         */
double dAlpha = 0.027;    /* license plate rotation angle [rad] */
double dBeta  = 0.007;    /* license plate skew angle     [rad] */
double dLpWidth  = 222.0; /* license plate width  [pxl]         */
double dLpHeight = 22.0;  /* license plate height [pxl]         */

/* run LP detection on a given image */
LpmDetResult *pDetRes;
pDetRes = lpmRunDetModuleW(pState,idx,pImage,&BoundingBox,
                           dAlpha, dBeta, dLpWidth, dLpHeight);

/* print the results */
printf("Number of detections: %d\n",pDetRes->NumDetections);
```

Note: `LpmDetResult` structure has to be later deallocated using function `lpmFreeDetResult()`.

## 7. Running license plate reading:

Call the license plate reading function `lpmRunOcrModule()` which analyzes a given LP detection and reads the appropriate license plate text. The OCR result will be returned in `LpmOcrResult` structure (see lpm_type.h or the html documentation for the definition). `LpmOcrResult` structure has to be later deallocated using the function `lpmFreeOcrResult()`.
Example:

```
/* Running OCR on each LP detection */
for (int i = 0; i<pDetRes->NumDetections;i++)
{
      LpmOcrResult* pOcrRes;

      /* license plate reading */
      pOcrRes = lpmRunOcrModule(pState,idx,
                  pDetRes->Detection[i].Image);

      /* Print LP text */
      int * asciicode = pOcrRes->Hypothesis[0].Line[0].AsciiCode;
      while (*asciicode)
         printf("%s", asciicode++);
      printf("\n");

      /* deallocate ocr result*/
      lpmFreeOcrResult(pState,idx,pOcrRes);
}
```

## 8. Running vehicle classification:

The `lpmRunVclModule()` function performs the vehicle classification if the module supports this feature (if not, the NULL is returned). The original input image with the detected license plate center coordinates are taken as input arguments.

Example:

```
/* Running VCL on each LP detection */
for (int i = 0; i<pDetRes->NumDetections; i++)
{
      LpmVclResult* pVclRes;

      /* LP Row center coordinate */
      int cr = (  pDetRes->Detection[i].Position.TopLeftRow+
                  pDetRes->Detection[i].Position.BotRightRow)/2;

      /* LP Col center coordinate */
      int cc = (  pDetRes->Detection[i].Position.TopLeftCol+
                  pDetRes->Detection[i].Position.BotRightCol)/2;

      pVclRes = lpmRunVclModule(pState,idx,pImage,cr,cc);

      /* Print the vehicle category */
      printf("Category: %s\n\n", ppVclRes[i_det]->Category);
      printf("Make    : %s\n\n", ppVclRes[i_det]->Make);
      printf("Model   : %s\n\n", ppVclRes[i_det]->Model);
```

```
        lpmFreeVclResult(pState,idx,pVclRes);
}
```

Two new features come with the LPM-v3.0 release: lpmGetVclCount() and lpmGetVclDescription() which can evaluate information about vehicle classes which are distinguishable within the current VCL engine.

Example:

```
char * catg, * make, * model;
/* Get a number of vehicle classes. */
int iCount = lpmGetVclCount(pState, idx);
for (int i = 0; i < iCount; i++)
{
      /* Get a given class description */
      lpmGetVclDescription(pState, idx, i, &catg, &make, &model);
      printf("Class ID: %d (%s %s %s)\n", i, catg, make, model);
}
```

### 9. Deallocate detection result and image data:

Do not forget to deallocate detection result and image data after finishing work with a given image.

Example:

```
/* deallocate detection result */
lpmFreeDetResult(pState,idx,pDetRes);
/* deallocate image data */
lpmFreeMatrix(pImage);
```

### 10. For the next image repeat steps 4 – 9.

### 11. Finishing work with the current module:

Deallocate module data using `lpmFreeModule()` function after finishing work with a given module.

Example:

```
lpmFreeModule(pState,idx);
```

### 12. Finishing work with LPM layer, free lpm-state:

At the end deallocate lpm-state.

Example:

```
lpmFree(pState);
```

# Systematic cyclic updating

Eyedea Recognition LPR engines are based on supervised learning algorithms, i.e. on algorithms which automatically adapt to training data sets. Eyedea Recognition provides a systematic adaptive process of all its LPR engines with respect to client's data samples. This updating process is recommended especially when client's specific cameras and lighting devices are used.

Each client receives a **FTP** account at ***Eyedea Recognition's*** server. A client has the possibility to regularly upload data samples (or problematic data) from their cameras and consecutively receive the corresponding update of LPR engines. This systematic approach of cyclic updates enables to adapt LPR engines to client's data and to receive the best possible performance. Later, the same procedure is used for the regular verification of acquired statistics.