REPUBLICA DEL PERU
REGISTRO NACIONAL DE IDENTIFICACION Y ESTADO CIVIL
CUI
DUPLICADO
DOCUMENTO NACIONAL DE IDENTIDAD
DNI 71289778 - 9

71289778

Primer Apellido
ZAPATA

Segundo Apellido
ACCHO

Pre Nombres
EDUARDO ALFONSO

Nacimiento: Fecha y Ubigeo
12 03 1998 140101

Sexo Estado Civil
M S

Fecha Inscripción
25 03 2008

Fecha Emisión
15 03 2016

Fecha Caducidad
26 05 2023

ZAPATA

I<PER71289778<7<<<<<<<<<<<<<<<
9803123M2305260PER<<<<<<<<<<<4
ZAPATA<<EDUARDO<ALFONSO<<<<<<<

CONSTANCIA DE SUFRAGIO	CONSTANCIA DE SUFRAGIO	CONSTANCIA DE SUFRAGIO	CONSTANCIA DE SUFRAGIO
CONSTANCIA DE SUFRAGIO	CONSTANCIA DE SUFRAGIO	CONSTANCIA DE SUFRAGIO	CONSTANCIA DE SUFRAGIO

Departamento
LIMA

Provincia
LIMA

Distrito
SAN MARTIN DE PORRES

Dirección
MZ E LT 23 PROVIV LAS MERCEDES ETP III

Observaciones
Donación de Organos NO

Grupo de Votación 265401

DR. JORGE LUIS TRIVIÑO LAZO
JEFE NACIONAL

Indice derecho 000371 000430 68733W 0355 85385352

PERÚ Ministerio de Salud

CERTIFICADO DE VACUNACIÓN / VACCINATION CERTIFICATE

Nombre / Name EDUARDO ALFONSO ZAPATA ACCHO	**Fecha de Nacimiento / Date of birth** 12/03/1998
Documento de Identidad / Identification document DNI: 71289778	**Nacionalidad / Nationality** PERU
Sexo / Sex M	**Vacuna / Vaccine** Vacuna contra Covid

Vacunado / Vaccinated

Fecha de Vacunación / Vaccination Date	Dosis / Dose	Fabricante y Lote de Vacuna / Product Name and Manufacturer Lot Number	Lugar de Vacunación / Vaccination Place
26/09/2021	1° DOSIS	SINOPHARM (B2021082362)	LIMA NORTE - Parque Zonal Mayta Capac - LIMA LIMA SAN MARTIN DE PORRES
17/10/2021	2° DOSIS	SINOPHARM (B2021082386)	LIMA NORTE - Parque Zonal Mayta Capac - LIMA LIMA SAN MARTIN DE PORRES
08/04/2022	3° DOSIS	PFIZER (FM9091)	LIMA NORTE - Parque Zonal Mayta Capac - LIMA LIMA SAN MARTIN DE PORRES

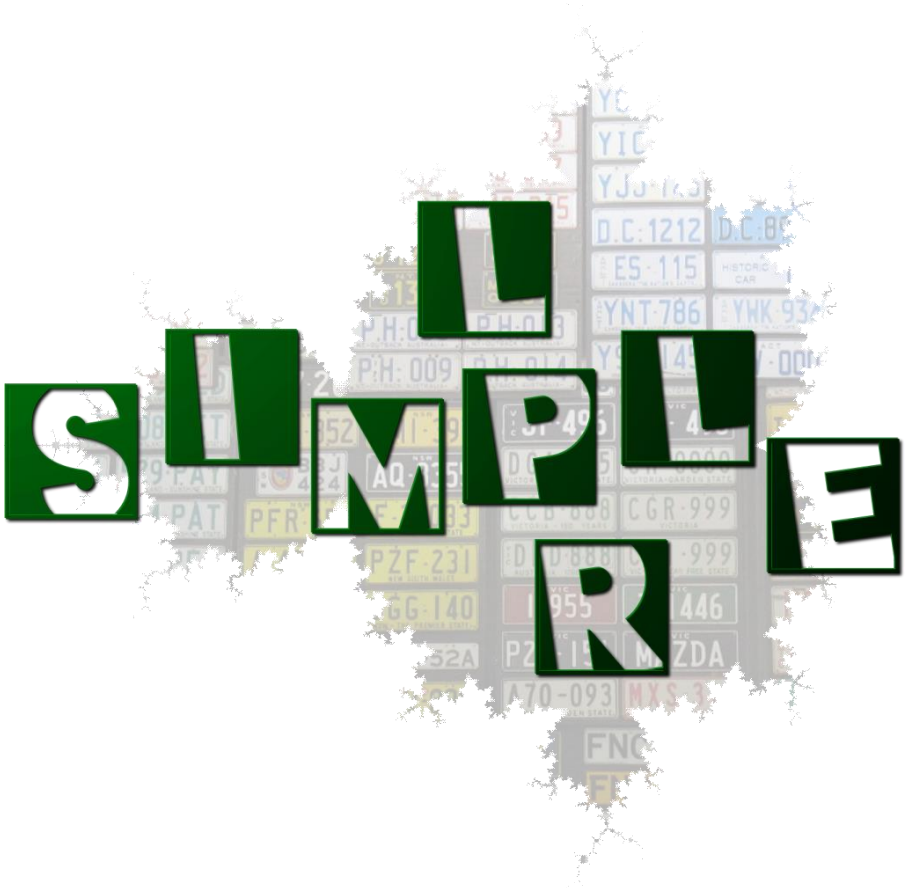
Certificado emitido por / Certificate issued by:
Ministerio de Salud del Perú

Fecha de Emisión / Date of Issue
07/07/2022, 4:03 pm

11 Once

SimpleLPR 3.4

License Plate Recognition Toolkit



12 Doce

13 Trece

Version History

Doc. version	Prod. version	Date	Description
1.0	2.0	28-Nov-09	First version of this document
1.1	2.1	22-May-11	Version 2.1of SimpleLPR
1.2	2.2	1-July-11	Version 2.2 of SimpleLPR
1.3	2.2.4.0	29-Oct-11	Version 2.2.4.0 of SimpleLPR
1.4	2.2.5.0	29-Nov-11	Version 2.2.5.0 of SimpleLPR
1.5	2.3.0.1	19-July-12	Version 2.3.0.1 of SimpleLPR
1.6	2.4.0.1	2-July-14	Version 2.4.0.1 of SimpleLPR
2.0	3.0.0.0	24-April-18	Version 3.0.0.0 of SimpleLPR
2.1	3.0.0.2	8-May-18	Version 3.0.0.2 of SimpleLPR
2.2	3.0.5.0	26-July-18	Version 3.0.5.0 of SimpleLPR
2.3	3.1.0.0	2-May-2019	Version 3.1.0.0 of SimpleLPR
2.4	3.2.0.0	27-June-2019	Version 3.2.0.0 of SimpleLPR
2.5	3.3.0.0	8-Oct-2020	Version 3.3.0.0 of SimpleLPR
2.6	3.4.0.0	28-March-2021	Version 3.4.0.0 of SimpleLPR

14 Catorce

Liability Disclaimer

Warelogic assumes no responsibility for any errors that may appear in this document nor does it make expressed or implied warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

To allow for design and specification improvements, the information in this document is subject to change at any time, without notice. Reproduction of this document or portions thereof without prior written approval of Warelogic is prohibited.

Warelogic shall not be liable for incidental or consequential damages in connection with, or arising out of the furnishing, performance, or use of this document and the program material that it describes.

Microsoft, MS, MSN, ActiveX, Windows, Windows NT, Visual Basic, Visual C++, and the Windows logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Microsoft products are licensed to OEMs by Microsoft Licensing, Inc., a wholly owned subsidiary of Microsoft Corporation.

All other product, brand, or trade names used in this publication are the trademarks or registered trademarks of their respective trademark owners.

15 Quince

Table of Contents

16 Dieciseis

17 Diecisiete

1 Introduction

At present, there are many license plate recognition (LPR) solutions in the market, designed to work in areas such as traffic control and monitoring, parking access, vehicle management, detection of security violations, and so forth. The best LPR engines support many countries, all license plate layouts, charsets, and some of them feature a 98% recognition rate or greater at recognition speeds of less than 10 ms per frame. Not surprisingly, this superior performance generally comes at a price; the best engines can cost thousands of dollars per runtime license.

On the other hand, there are consumer-oriented applications that can benefit from LPR but the cost of a high-end LPR engine is too high for their segment. This is where *SimpleLPR 3* comes in. *SimpleLPR 3* is a royalty-free low-end LPR engine directed to price sensitive applications that can cost an order of magnitude less than its high end counterparts, at the expense of a limited number of supported countries and lower accuracy; the typical recognition rate is about 90%, although as new releases appear this number is expected to improve. For applications that can trade some recognition accuracy for an affordable price *SimpleLPR 3* is an option that should be considered.

From a developer's perspective, *SimpleLPR 3* main design goal is integration and deployment simplicity. Hence, *SimpleLPR 3* runtime can be redistributed along with third-party applications by just x-copying a few *DLL* files that do not require COM registration. *SimpleLPR 3* provides a C/C++ interface for native applications, and it also exports a .NET object model that can be directly used from VB or C#. The current version supports more than 70 countries. It can read 24-bit RGB or 8-bit grayscale images from *JPEG*, *TIFF*, *BMP* or *PNG* files, as well as from a memory buffer.

This guide contains both the product description and documentation for developers. In the following sections, all required information on how to integrate *SimpleLPR 3* into a third-party application is provided.

18 Dieciocho

2 Supported Countries and Limitations

The current version can read license plates from the supported countries, provided that they are in good condition; namely, the license plates should be crisp, readable, without occlusion, bumps, or scratches. In addition, images supplied to the *SimpleLPR 3* engine should portray the license plate as viewed from a frontal angle, and the height of the license plate characters should be 20 pixels or taller.

Irrespective of complying with the above constraints, in general not all valid license plates from a specific country can be recognized. The level of support varies from country to country (supported countries are listed in http://warelogic.com). Readable license plates for which a country template does not exist are returned as unmatched text.

19 Diecinueve

3 Prerequisites, Installation and Deployment

3.1 Prerequisites

SimpleLPR 3 requires a computer running a *Windows Vista* or newer operating system, the *MSXML 6.0* XML parser, and the Microsoft Visual C++ Redistributable for Visual Studio 2017 package installed on the system.

Any 5 year old or newer computer kept regularly updated with *Microsoft*'s patches and service packs should already meet the above requirements. Nevertheless, if required the above software prerequisites can be downloaded from *Microsoft*.

1. *MSXML 6.0* can be installed from http://www.microsoft.com/downloads/details.aspx?FamilyID=D21C292C-368B-4CE1-9DAB-3E9827B70604&displaylang=en

2. The Visual Studio 2017 Redistributable can be downloaded from https://www.visualstudio.com/downloads/Installation

The *SimpleLPR 3 Toolkit* is shipped in a *MSI* installer. It creates the SDK file structure and automatically takes care of the dependencies: if needed it will install the *Visual Studio 2017 C++ runtime*. It will also verify that *MSXML 6.0* is installed and check for *AVX* extensions, prompting the user if any of them are not available.

Upon installation *SimpleLPR 3* operates in evaluation mode, which lasts for 60 days. Once the evaluation period terminates *SimpleLPR 3* will stop working, unless a valid product key is provided.

3.2 Deployment

SimpleLPR 3 can be deployed by simply copying the contents of the *bin* directory in the SDK to the same folder as the application executable. If the target application is using the .NET classes exposed by the *SimpleLPR3.dll* assembly it is **important to copy also the contents of the '*native*' *folder***. Two considerations to be borne in mind are

1. *SimpleLPR 3* cannot operate in evaluation mode when redistributed. The application has to supply a product key.
2. The installer of the third party application must ensure that all prerequisites are met.

4 License Plate Recognition with *SimpleLPR 3*

This section describes the major steps to integrate *SimpleLPR 3* into an application. In each stage both C++ and .NET flavors are examined individually. *.NET* sample code is presented in C#.

4.1 *SimpleLPR 3* Instantiation

This step includes the tasks of loading *SimpleLPR3* into the application process space and creating an instance of the *SimpleLPR* engine. In the case of a CUDA compatible GPU being available in the system, it can be selected as a parameter of the engine creation call and latter used for off-loading some computation-intensive tasks. When the GPU mode is selected, the user can independently enable the GPU for image processing tasks and/or text candidate classification. Moreover, since some of the image processing algorithms used in SimpleLPR are memory consuming, an additional parameter allows the user to specify the maximum number of concurrent image processing operations.

Native C++ Applications

For native C++ applications, a native DLL is provided with the name *SimpleLPR3_native.dll*, which exports a public function *Setup* that works as a factory method of *ISimpleLPR* objects. The prototype of the *Setup* function is defined in *include*/*SimpleLPR.h*

```
ISimpleLPR * _stdcall Setup(const SIMPLELPR_Engine_Setup_Parms &rcParms) throw ();
```

Although a *.lib* file is provided for *SimpleLPR3_native.dll,* it is advised to perform the binding programmatically using either the *LoadLibrary* or *LoadLibraryEx* calls. Once a pointer to the *Setup* function is acquired, it can be used to create an instance of the *SimpleLPR 3* engine, encapsulated in the *ISimpleLPR* interface. The following code snippet illustrates this.

```
// Load the SimpleLPR dll
HMODULE hLib = LoadLibraryEx( L"SimpleLPR3_native.dll", NULL, 0 );
if ( hLib == NULL )
       … \\ Handle error condition

// Get the entry point to the Setup function
SimpleLPRSetupFunc pfnSetup = (SimpleLPRSetupFunc)::GetProcAddress( hLib, "Setup" );
if ( pfnSetup == 0 )
       … \\ Handle error condition

SIMPLELPR_Engine_Setup_Parms setupParms;
setupParms.cudaDeviceId = -1; // Use CPU.
setupParms.enableImageProcessingWithGPU= 0;
setupParms.enableClassificationWithGPU= 0;
setupParms.maxConcurrentImageProcessingOps= 0; // Use default settings.
setupParms.guard = SIMPLELPR_SETUP_PARMS_GUARD; // Used to detect possible
                                                // version mismatches.
```

21 Veintiuno

```
// Create an instance of the SimpleLPR engine
ISimpleLPR *pLPR = (*pfnSetup)( setupParms );
if ( pLPR == NULL )
       ··· \\ Handle error condition

··· \\ Do your stuff

pLPR->release(); // Delete engine
```

.NET Applications

A *.NET* assembly with the name *SimpleLPR3.dll* is provided, which dispatches user calls to *SimpleLPR3_native.dll*. Once added to the project as an assembly reference, an instance of the *SimpleLPR 3* engine can be created using the *Setup* static factory method in the *SimpleLPR* class.

```
EngineSetupParms setupP;
setupP.cudaDeviceId = -1; // Select CPU
engP.enableImageProcessingWithGPU = false;
engP.enableClassificationWithGPU = false;
engP.maxConcurrentImageProcessingOps = 0; // Use default settings.
ISimpleLPR lpr = SimpleLPR.Setup(setupP);
```

4.2 Engine Configuration

Firstly, a valid product key must be provided if *SimpleLPR 3* is not to be used in evaluation mode. The product key can be supplied either from a file or from a memory buffer. As it contains personal data from the purchaser in readable form, the second method is preferred.

Then, the user must decide which countries are to be enabled as well as their relative weights. These weights are used to break ties in the event that two or more candidates from different countries are equally feasible. The *SimpleLPR 3* engine exports methods to assign each country a weight equal to or greater than 0. Setting a weight equal to 0 disables a country. Subsequently, those weights are later normalized to the [0...1] range. During license plate recognition, it is possible that the discovered groups of text and numbers match the syntax checking rules of more than one country. The 'goodness' index of each country's match is multiplied by the normalized country weight, and the result is used to return an ordered list of the possible country matches for each license plate candidate.

The samples below show how to set the product key and configure country weights in each development platform.

22 Veintidós

Native C++ Applications

```
// Set the product key
bool bOk = pLPR->productKey_set( L"productkey.xml" );
if ( ! bOk )
        … \\ Handle error condition
// Enable Germany, Spain and the United Kingdom
bOK = pLPR->countryWeight_set( L"Germany", 1.f )
if ( ! bOk )
        … \\ Handle error condition
bOK = pLPR->countryWeight_set( L"Spain", 1.f )
if ( ! bOk )
        … \\ Handle error condition
bOK = pLPR->countryWeight_set( L"UK-GreatBritain", 1.f )
if ( ! bOk )
        … \\ Handle error condition

// Apply changes
bOK = pLPR->realizeCountryWeights();
if ( ! bOk )
        … \\ Handle error condition
```

.NET Applications

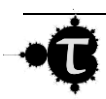
```
// Set the product key
lpr.set_productKey("productkey.xml");

// Enable Germany, Spain and the United Kingdom
lpr.set_countryWeight("Germany", 1.0f);
lpr.set_countryWeight("Spain", 1.0f);
lpr.set_countryWeight("UK-GreatBritain", 1.0f);

// Apply changes
lpr.realizeCountryWeights();
```

4.3 License Plate Recognition

Once the *SimpleLPR 3* engine has been properly configured, the remaining step is to create an instance of an *IProcessor* object to perform license plate recognition. Since *IProcessor* is not thread-safe, each working thread involved in LPR should manage its own instance of *IProcessor*.

An *IProcessor* can read and analyze images from a file in either *RGB* or *grayscale JPEG*, *TIFF*, *PNG,* and *BMP* formats or, alternatively, it can process images directly from a memory buffer. In the later case the image must be either *RGB* 24-bit or *grayscale* 8-bit. The output of the ANPR methods is a list of license plate candidate objects. Each plate candidate object, in turn, consists of a list of possible country match objects, being the last entry in the list the element corresponding to the raw text object. Country match objects hold the license plate text in Unicode string form, the candidate confidence score, and the bounding box and confidence value of all individual elements in the license plate.

The license plate text is formatted according to the rules of each specific country; thus spaces or hyphens can be added to separate groups of text and numbers. The returned



23 Veintitrés

glyph confidence score ranges from 0 to 1 and corresponds to the direct output of the OCR classifier. Since it can either underestimate or overestimate the actual probability of the result being correct, care should be taken when setting thresholds to discard weak candidates. The rule of thumb is that very low values correspond to unreliable detections, whereas values close to one imply a safer candidate. Lastly, the confidence score of the country match is computed as the product of the confidences of its glyphs weighted by a factor derived from the similarity between the expected inter-glyph separations and the actual ones present in the candidate.

Optionally, *IProcessor* can be configured to estimate the probability that the detected candidates actually correspond to license plates. When this mode is enabled, the confidence that the detected text belongs to a license plate, alongside the four vertices of the enclosing quadrilateral of the license plate, is returned in the *ICandidate* object.

Native C++ Applications

```
// Create processor
IProcessor *pProc = pLPR->createProcessor();
if (pProc == NULL )
{
   // Handle error condition
   ...
}

// Process source file
ICandidates *pCds = pProc->analyze(L"vehicle.jpg");
if (pCds == NULL )
{
   // Handle error condition
   ...
}

std::wcout << pCds->numCandidates_get() << L" license plate candidates found:" << std::endl;

// Iterate over all candidates
for ( _SIZE_T i = 0; nErr == 0 && i < pCds->numCandidates_get(); ++i )
{
         ICandidate *pCd = pCds->candidate_get( i );

         std::wcout << L"***********" << std::endl;
         std::wcout << L"Candidate " << i + 1 << std::endl;

         struct SIMPLELPR_Rect candBB;
         pCd->boundingBox_get( candBB );

         std::wcout << L"Light background " << pCd->brightBackground_get();
         std::wcout << L", left: " << candBB.left << L", top: " << candBB.top;
         std::wcout << L", width: " << candBB.width << L", height: " << candBB.height;
         std::wcout << std::endl;

         // Iterate over all country matches.
         for ( _SIZE_T j = 0; j < pCd->numCountryMatches_get(); ++j )
         {
                  ICountryMatch *pMatch = pCd->countryMatch_get( j );

                  std::wcout << L"-----------" << std::endl;
                  std::wcout << L"Match " << j + 1 << std::endl;
```

24 Veinticuatro

```
                std::wcout << L"Text: " << pMatch->text_get() << L", country: " << pMatch->country_get();
                std::wcout << L", ISO: " << pMatch->countryISO_get();
                std::wceout << L", confidence: " << pMatch->confidence_get();
                std::wcout << std::endl;

                std::wcout << L"Elements: " << std::endl;

                // Iterate over all elements
                for ( _SIZE_T k = 0; k < pMatch->numElements_get(); ++k )
                {
                        Element e;
                        pMatch->element_get( k, e );

                        std::wcout << L"Glyph: " << e.glyph << L", confidence: " << e.fConfidence;
                        std::wcout << L", left: " << e.boundingBox.left << L", top: " << e.boundingBox.top;
                        std::wcout << L", width: " << e.boundingBox.width;
                        std::wceout << L", height: " << e.boundingBox.height;
                        std::wcout << std::endl;
                }

                pMatch->release();
        }

        pCd->release();
}
// Cleanup
if ( pCds != NULL )
        pCds->release();

if ( pProc != NULL )
        pProc->release();
```

.NET Applications

```
// Create Processor
IProcessor proc = lpr.createProcessor();

// Process source file
List<Candidate> cds = proc.analyze( “vehicle.jpg” );
Console.WriteLine("{0} license plate candidates found:", cds.Count);

// Iterate over all candidates
foreach (Candidate cd in cds)
{
        Console.WriteLine("***********");
        Console.WriteLine("Light background: {0}, left: {1}, top: {2}, width: {3}, height: {4}",
                          cd.brightBackground, cd.bbox.Left, cd.bbox.Top, cd.bbox.Width, cd.bbox.Height);
        Console.WriteLine("Matches:");

        // Iterate over all country matches
        foreach (CountryMatch match in cd.matches)
        {
                Console.WriteLine("-----------");
                Console.WriteLine("Text: {0}, country: {1}, ISO: {2}, confidence: {3}",
                                  match.text, match.country, match.countryISO, match.confidence);
                Console.WriteLine("Elements:");

                foreach (Element e in match.elements)
                {
                        Console.WriteLine("   Glyph: {0}, confidence: {1}," +
```

25 Veinticinco

```
                                        "left: {2}, top: {3}, width: {4}, height: {5}",
                                    e.glyph, e.confidence, e.bbox.Left,
                                    e.bbox.Top, e.bbox.Width, e.bbox.Height);
                }
        }
}
```

4.4 Considerations on Error Handling and Resource Management

For the sake of simplicity, the subject of error handling has been deliberately omitted in the sample code above. Furthermore, the reference counting scheme implemented in the C++ interface deserves some discussion.

Native C++ Applications

Exceptions are a convenient mechanism for dealing with error conditions in C++. However, their implementation is compiler vendor dependant and thus they are not appropriate for a public interface exported from a *DLL*. For this reason, *SimpleLPR 3* relies on methods returning error codes combined with a per-thread error state. Every method in the C++ interface returns either a Boolean (true meaning success) or a NULL value in case that the method is expected to return a pointer.

Once an error condition has been determined, further error information can be queried by means of the *lastError_get()* method. Each thread keeps its own copy of the *last_error* variable. The following code snippet shows how the error state can be obtained.

```
SimpleLPR3_Native::IErrorInfo *pErr = pLPR->lastError_get();
if ( pErr != NULL )
{
  std::wcerr << L"Error occurred. Error code: " << pErr->errorCode;
  std::wcerr << L", description: " << pErr->description << std::endl;
  pErr->release(); // Free object
}
```

On the topic of resource management, the standard way of de-allocating objects in C++ is using the *delete* operator. On the other hand, the golden rule of any DLL returning C++ objects is that resources must be allocated and de-allocated inside the DLL. A way to accomplish it is by making objects delete themselves. The reference counting idiom does this; each object keeps track of how many references point to it. When this counter reaches 0, the object deletes itself. Thus, all objects exported in the C++ interface derive from *IReferenceCounted*. This interface has two methods, *addRef*, which increments objects reference counter by one, and *release*, which decrements it. All objects returned by the *SimpleLPR 3* factory methods have their reference counter set to 1, and calling their release method actually destroys them.

.NET Applications

As usual, *.NET* makes life easier for developers. Error conditions are dealt with the use exceptions, which is the standard way in *.NET*. Likewise, resource management is not an issue here as the garbage collector takes care of this subject.

26 Veintiséis

5 C++ Interface Reference

The file *include\SimpleLPR.h* contains all class declarations and function prototypes described in this section. All members are declared under the *SimpleLPR3_Native* namespace.

5.1 Functions

5.1.1 Setup

```
ISimpleLPR * Setup(const SIMPLELPR_Engine_Setup_Parms &rcParms) throw ();
```

Description: Factory method of *ISimpleLPR* objects.
Parameters
rcParms: A reference to a SIMPLELPR_Engine_Setup_Parms struct.
Returns: A pointer to a *ISimpleLPR* object. The returned *ISimpleLPR* object must be de-allocated by calling its *release* method.
Remarks: It is advised to load *SimpleLPR3_native.dll* dynamically.

5.2 Structures

5.2.1 SIMPLELPR_Engine_Setup_Parms

Parameters for engine creation.

```
struct SIMPLELPR_Engine_Setup_Parms
{
        __int32 cudaDeviceId;
        SIMPLELPR_BOOL enableImageProcessingWithGPU;
        SIMPLELPR_BOOL enableClassificationWithGPU;
        SIMPLELPR_SIZE_T maxConcurrentImageProcessingOps;
        unsigned __int32 guard;
};
```

Members
cudaDeviceId: Valid CUDA device identifier, or -1 for CPU only mode.
enableImageProcessingWithGPU: In case this value is different than 0 and a valid CUDA device identifier is provided then use the GPU for image processing tasks.
enableClassificationWithGPU: In case this value is different than 0 and a valid CUDA device identifier is provided then use the GPU for text candidate classification.
maxConcurrentImageProcessingOps: Maximum number of concurrent image processing operations. The recommended values are 1 or 2 when using the CPU for image processing tasks, and 4 or more when employing a GPU. Setting this parameter to 0 is equivalent to selecting the recommended values for each case.
guard: Used to detect version conflicts.
Must be set to SIMPLELPR_SETUP_PARMS_GUARD.

27 Veintisiete

5.2.2 Point

This structure encapsulates a point.

```
struct Rect
{
        __int32 x;
        __int32 y;
};
```

Members

x Horizontal coordinate.
y Vertical coordinate.

5.2.3 Rect

This structure represents a rectangle.

```
struct Rect
{
        __int32 left;
        __int32 top;
        __int32 width;
        __int32 height;
};
```

Members

left Leftmost coordinate of the rectangle.
top Topmost coordinate of the rectangle.
width Rectangle width.
height Rectangle height.

5.2.4 Quad

Represents the four vertices of a quadrilateral.

```
struct Quad
{
        Point vertices[ 4 ];
};
```

Members

vertices The four vertices of the quadrilateral in counterclockwise order.

5.2.5 Element

Holds the LPR results of an individual character in a license plate.

```
struct Element
{
        wchar_t glyph;
        _REAL_T fConfidence
        Rect boundingBox;
```

28 Veintiocho

```
};
```

Members

glyph Unicode representation of the character.

fConfidence 'Goodness' of the recognition. Its range is 0 to 1 and can be used to rank candidates although it should not be regarded as a probability. In general, a 2x goodness value is not twice as good as x. See the section on LPR for more detailed information on confidence values.

boundingBox Bounding rectangle of the character, in pixel coordinates.

5.2.6 VersionNumber

This structure holds the four numbers that comprise the SimpleLPR version number.

```
struct Rect
{
        unsigned __int32 A;
        unsigned __int32 B;
        unsigned __int32 C;
        unsigned __int32 D;
};
```

Members

A First element of the of SimpleLPR_Native3.dll product number.
B Second element of the of SimpleLPR_Native3.dll product number.
C Third element of the of SimpleLPR_Native3.dll product number.
D Fourth element of the of SimpleLPR_Native3.dll product number.

5.3 Interfaces

5.3.1 IReferenceCounted

This interface constitutes the base for all interfaces. It manages object life cycles by means of reference counting. All factory methods in this library return pointers to objects with their reference count set to 1 or more.

It is advisable to call the *addRef* method every time a pointer alias is created. Likewise, the *release* method should be called before an object pointer goes out of scope. When an object reference count reaches zero the object is destroyed.

```
struct IReferenceCounted
{
        virtual void addRef( void ) throw () = 0;
        virtual void release( void ) throw () = 0;
};
```

Methods

29 Veintinueve

```
virtual void addRef( void ) throw () = 0;
```

Description: Increments an object reference count by one.

```
virtual void release( void ) throw () = 0;
```

Description: Decrements an object reference count by one. When it reaches 0 the object is destroyed.

5.3.2 IErrorInfo

This class conveys an error code and a description. Every time an error occurs an *IErrorInfo* object is created and kept in thread local storage (TLS). To retrieve it use the *ISimpleLPR::lastError_get* method.

```
struct IErrorInfo : public IReferenceCounted
{
        virtual _HRESULT errorCode_get() const throw () = 0;
        virtual const wchar_t * description_get() const throw () = 0;
};
```

Methods

```
virtual _HRESULT errorCode_get() const throw () = 0;
```

Description: Returns a COM like *HRESULT* error code.

```
virtual const wchar_t * description_get() const throw () = 0;
```

Description: Returns a textual description of the error.

5.3.3 ICountryMatch

Encapsulates a country match for a license plate candidate.

```
struct ICountryMatch : public IReferenceCounted
{
        virtual const wchar_t *text_get() const throw () = 0;
        virtual const wchar_t *country_get() const throw () = 0;
        virtual const wchar_t *countryISO_get() const throw () = 0;
        virtual _REAL_T confidence_get() const throw () = 0;
        virtual _SIZE_T numElements_get() const throw () = 0;
        virtual bool element_get( _SIZE_T id, /*[out]*/Element &rElem ) const throw () = 0;
};
```

Methods

30 Treinta

```
virtual const wchar_t *text_get() const throw () = 0;
```

Description: Returns the Unicode representation of the license plate string. Separators are represented as white space.

```
virtual const wchar_t *country_get() const throw () = 0;
```

Description: Returns the country code in string form.

```
virtual const wchar_t *countryISO_get() const throw () = 0;
```

Description: Returns the country ISO Alpha-2 code.

```
virtual _REAL_T confidence_get() const throw () = 0;
```

Description: Returns the overall 'goodness' of the recognition.

```
virtual void boundingBox_get( /*[out]*/Rect &rBB ) const throw ( ) = 0;
```

Description: Returns the bounding box of the text in the country match.

```
virtual _SIZE_T numElements_get() const throw () = 0;
```

Description: Number of components in the license plate..

```
virtual bool element_get( _SIZE_T id, /*[out]*/Element &rElem ) const throw () = 0;
```

Description: Information about the individual chars that make up the license plate. They are listed in the same order as they appear in the text string. To know the physical layout of the license plates use the Element::bbox field.

5.3.4 ICandidate

Encapsulates a license plate candidate.

```
struct ICandidate : public IReferenceCounted
{
        virtual bool brightBackground_get() const throw () = 0;
        virtual void boundingBox_get( /*[out]*/ Rect &rBB ) const throw () = 0;
        virtual _REAL_T plateRegionDetectionConfidence_get() const throw () = 0;
        virtual void plateRegionVertices_get( /*[out]*/Quad &rQuad ) const throw () = 0;
        virtual _SIZE_T numCountryMatches_get () const throw () = 0;
        virtual ICountryMatch *countryMatch_get( _SIZE_T id ) const throw () = 0;
};
```

31 Treinta y uno

Methods

```
virtual bool brightBackground_get() const throw () = 0;
```

Description: *true* if the license plate features dark text on a light background. *false* if otherwise.

```
virtual void boundingBox_get( /*[out]*/ Rect &rBB ) const throw ( ) = 0;
```

Description: Returns the bounding box of the text in the license plate.

```
virtual _REAL_T plateRegionDetectionConfidence_get() const throw () = 0;
```

Description: Returns the estimated probability that the candidate corresponds to a license plate rather than incidental scene text.
Remarks: Returns -1 in case the plate region detection and extraction feature has not been previously enabled in *IProcessor*.

```
virtual void plateRegionVertices_get( /*[out]*/Quad &rQuad ) const throw () = 0;
```

Description: Returns the four vertices of the detected license plate region in case that the license plate region confidence is greater than zero, otherwise all point coordinates are set to -1.

```
virtual _SIZE_T numCountryMatches_get () const throw () = 0;
```

Description: Number of country matches for the candidate.

```
virtual ICountryMatch *countryMatch_get( _SIZE_T id ) const throw ( ) = 0;
```

Description: Returns a country match object given its index.
Remarks: The returned *ICountryMatch* object must be de-allocated by calling its *release* method.

5.3.5 ICandidates

Encapsulates a collection of license plate candidates.

```
struct ICandidates : public IReferenceCounted
{
        virtual _SIZE_T numCandidates_get() const throw () = 0;
        virtual ICandidate *candidate_get( _SIZE_T id ) const throw () = 0;
};
```

Methods

```
virtual _SIZE_T numCandidates_get() const throw () = 0;
```

32 Treinta y dos

Description: Returns the number of elements in the collection.

```
virtual ICandidate *candidate_get( _SIZE_T id ) const throw () = 0;
```

Description: Returns a candidate object given its index.
Returns: A pointer to the selected candidate object.
Remarks: The returned *ICandidate* object must be de-allocated by calling its *release* method.

5.3.6 IProcessor

Encapsulates the LPR functionality of *SimpleLPR 3*. This class is not multi-threaded and, therefore, each thread should use a dedicated *IProcessor* instance.

```
struct IProcessor : public IReferenceCounted
{
        virtual _REAL_T contrastSensitivityFactor_get() const throw () = 0;
        virtual bool contrastSensitivityFactor_set( _REAL_T fFactor ) throw () = 0;

        virtual bool plateRegionDetectionEnabled_get() const throw () = 0;
        virtual void plateRegionDetectionEnabled_set( bool bEnabled ) throw () = 0;

        virtual bool cropToPlateRegionEnabled_get() const throw () = 0;
        virtual void cropToPlateRegionEnabled_set( bool bEnabled ) throw () = 0;

        virtual ICandidates *analyze_C3( const void *pcvImgData,
                                         _SIZE_T widthStep,
                                         _SIZE_T width,
                                         _SIZE_T height,
                                         _REAL_T fWeight0,
                                         _REAL_T fWeight1,
                                         _REAL_T fWeight2 ) throw () = 0;
        virtual ICandidates *analyze( const void *pcvImgData,
                                      _SIZE_T widthStep,
                                      _SIZE_T width,
                                      _SIZE_T height) throw () = 0;
        virtual ICandidates *analyze( const wchar_t * pcwsImgPath ) throw () = 0;
};
```

Methods

```
virtual _REAL_T contrastSensitivityFactor_get() const throw () = 0;
```

Description: Returns the current contrast sensitivity factor (CSF) parameter for the IProcessor. The CSF must lie in the interval [0..1]. Low CSF values can improve the recognition accuracy when there is noise in the image or there are shadows on the license plates. Otherwise higher CSF values are recommended.

```
virtual bool contrastSensitivityFactor_set( _REAL_T fFactor ) throw () = 0;
```

Description: Sets the contrast sensitivity factor (CSF) parameter for the IProcessor. The CSF must lie in the interval [0..1]. Low CSF values can improve the recognition accuracy when

33 Treinta y tres

there is noise in the image or there are shadows on the license plates. Otherwise higher CSF values are recommended.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

```
virtual bool plateRegionDetectionEnabled_get() const throw () = 0;
```

Description: Returns true if the plate region detection and extraction functionality is enabled, and false otherwise.

```
virtual void plateRegionDetectionEnabled_set( bool bEnabled ) throw () = 0;
```

Description: Enables/disables the plate region detection and extraction functionality.

```
virtual bool cropToPlateRegionEnabled_get() const throw () = 0;
```

Description: Tells whether results are going to be cropped to the detected plate region or not. Returns true if the crop results to the plate region functionality is enabled, and false otherwise.

```
virtual void cropToPlateRegionEnabled_set( bool bEnabled ) throw () = 0;
```

Description: Enables/disables the crop results to the plate region functionality.

```
virtual ICandidates *analyze_C3( const void *pcvImgData,
                                 _SIZE_T widthStep,
                                 _SIZE_T width,
                                 _SIZE_T height,
                                 _REAL_T fWeight0,
                                 _REAL_T fWeight1,
                                 _REAL_T fWeight2 ) throw () = 0;
```

Description: Looks for license plate candidates in a memory buffer containing a 3-channel 8 bit/channel color image stored in pixel order. Every pixel in the source image is internally converted to gray scale according to the following formula:

*L = fWeight0 * C0 + fWeight1 * C1 + fWeight2 * C2*

For instance, to convert a RGB image to gray scale according the standard for the NTSC CRT the value of the weights should be:

Red: fWeight0 = 0.299
Green: fWeight1 = 0.587
Blue: fWeight2 = 0.114

34 Treinta y cuatro

Parameters
Input
pImgData: Pointer to the first image row. The image must be a 3-channel 8 bit/channel color image stored in pixel order and down. The top row of the image is the first row in memory, followed by next row down.
widthStep: Distance in bytes between starts of consecutive rows in the source image.
width: Image width in pixels.
height: Image height in pixels.
fWeight0: Weight of the first channel.
fWeight1: Weight of the second channel.
fWeight2: Weight of the third channel.
Returns: A pointer to an *ICandidates* collection containing all license plate candidates. If something goes wrong it returns a NULL pointer, use *ISimpleLPR::lastError_get* to get the error information.
Remarks: This method is not multi-threaded. The returned *ICandidates* object must be de-allocated by calling its *release* method.

```
virtual ICandidates *analyze( const void *pcvImgData,
                              _SIZE_T widthStep,
                              _SIZE_T width,
                              _SIZE_T height ) throw () = 0;
```

Description: Looks for license plate candidates in a memory buffer containing an 8 bit gray scale image.
Parameters
Input
pImgData: Pointer to the first image row. The image must be 8 bit gray scale and top down. The top row of the image is the first row in memory, followed by the next row down.
widthStep: Distance in bytes between starts of consecutive rows in the source image.
width: Image width in pixels.
height: Image height in pixels.
Returns: A pointer to an *ICandidates* collection containing all license plate candidates. If something goes wrong it returns a NULL pointer, use *ISimpleLPR::lastError_get* to get the error information.
Remarks: This method is not multi-threaded. The returned *ICandidates* object must be de-allocated by calling its *release* method.

```
virtual ICandidates *analyze( const wchar_t * pcwsImgPath ) throw () = 0;
```

Description: Looks for license plate candidates in an image in a *.jpg*, *.png*, *.tif* or .bmp file. The images can be either 24 bit RGB or 8 bit gray scale.
Parameters
Input
pcwsImgPath: Path to a file containing a 24 bit RGB or 8 bit gray scale image.
Returns: A pointer to a ICandidates collection containing all license plate candidates. If something goes wrong it returns a NULL pointer, use *ISimpleLPR::lastError_get* to get the error information.

35 Treinta y cinco

Remarks: This method is not multi-threaded. The returned *ICandidates* object must be de-allocated by calling its *release* method.

5.3.7 ISimpleLPR

Encapsulates the *SimpleLPR 3* engine.

```
struct ISimpleLPR : public IReferenceCounted
{
        virtual _SIZE_T numSupportedCountries_get() const throw () = 0;
        virtual bool countryCode_get( _SIZE_T id,
                                      /*[out]*/const wchar_t *&rpcwsCode ) const throw () = 0;
        virtual bool countryWeight_get( _SIZE_T id,
                                        /*[out]*/_REAL_T &rfWeight ) const throw () = 0;
        virtual bool countryWeight_get( const wchar_t *id,
                                        /*[out]*/_REAL_T &rfWeight ) const throw () = 0;
        virtual bool countryWeight_set( _SIZE_T id,
                                        _REAL_T fWeight ) throw () = 0;
        virtual bool countryWeight_set( const wchar_t *id,
                                        _REAL_T fWeight ) throw () = 0;
        virtual bool realizeCountryWeights() throw () = 0;
        virtual IErrorInfo *lastError_get( bool bClear = true ) throw () = 0;
        virtual IProcessor *createProcessor() throw () = 0;
        virtual bool productKey_set( const wchar_t *productKeyPath ) throw () = 0;
        virtual bool productKey_set( const void *key, _SIZE_T keySize ) throw () = 0;
        virtual bool versionNumber_get( /*[out]*/ VersionNumber &rVersion ) throw () = 0;
};
```

Methods

```
virtual _SIZE_T numSupportedCountries_get() const throw () = 0;
```

Description: Returns the number of supported countries.

```
virtual bool countryCode_get( _SIZE_T id, /*[out]*/const
                              wchar_t *&rpcwsCode ) const throw () = 0;
```

Description: Given a country index it returns its string identifier.
Parameters
  Input
    id: The country index. 0 <= *id* <= *numSupportedCountries_get()* - 1.
  Output:
    rpcwsCode:: The country Unicode string identifier..
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

36 Treinta y seis

```
virtual bool countryWeight_get( _SIZE_T id,
                                /*[out]*/_REAL_T &rfWeight ) const throw () = 0;
```

Description: Given a country index it returns the relative country weight. Weights are used to break ties when a candidate can belong to multiple countries.
Parameters
Input
Id: The country index. 0 <= *id* <= numSupportedCountries_get() - 1.
Output
rfWeight:: The relative weight of the country.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

```
virtual bool countryWeight_get( const wchar_t *id,
                                 /*[out]*/_REAL_T &rfWeight ) const throw () = 0;
```

Description Given a country string identifier it returns the relative country weight. Weights are used to break ties when a candidate can belong to multiple countries.
Parameters
Input
Id: The country string identifier. See *countryCode_get*.
Output
rfWeight:: The relative weight of the country.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

```
virtual bool countryWeight_set(_SIZE_T id,
                               _REAL_T fWeight) throw () = 0;
```

Description: Given a country index it sets the country relative weight. Weights are used to break ties when a candidate can belong to multiple countries.
Parameters
Input
id: The country index. 0 <= *id* <= numSupportedCountries_get() - 1.
fWeight:: The desired country weight. *fWeight* >= 0.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.
Remarks: This method is not multi-threaded. Weight must be >= 0. Use a zero weight to effectively disable a specific country.

```
virtual bool realizeCountryWeights() throw () = 0
```

Description: Rebuilds the internal country verification lookup tables based on which countries are enabled and their relative weights. Call it once you have finished configuring country weights.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

37 Treinta y siete

Remarks: This method is not multi-threaded. Depending on the countries selected this method can be time consuming. After this method execution all existing and new *IProcessor* instances will start using the new weights. Avoid calling this method when another thread is executing *IProcessor::analyze*.

```
virtual IErrorInfo *lastError_get( bool bClear = true ) throw () = 0;
```

Description Returns an *IErrorInfo* object that describes the latest error occurred.
Parameters
Input
bClear: If *true* the error state will be cleared after this call.
Returns: *An IErrorInfo* object that describes the latest error or NULL if no error has occurred since application startup or the last call to *lastError_get* with *bClear* set to *true*.
Remarks: This method is multi-threaded. In particular each thread maintains a TLS slot with error state so threads can be independent from each other.

```
virtual IProcessor *createProcessor() throw () = 0;
```

Description: Creates a new *IProcessor* object.
Returns: The newly created *IProcessor* instance, or NULL if the method failed. Use *ISimpleLPR::lastError_get* to obtain the error information.
Remarks: This method is multi-threaded. For this method to succeed, either the product must be within the evaluation period o a valid product key must be supplied using *productKey_set*. The returned *IProcessor* object must be de-allocated by calling its release method.

```
virtual bool productKey_set( const wchar_t *productKeyPath ) throw () = 0;
```

Description: Sets the product key from a license file.
Parameters
Input
productKeyPath: Path to the product key file.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

```
virtual bool productKey_set( const void *key, _SIZE_T keySize ) throw () = 0;
```

Description: Sets the product key from a memory buffer.
Parameters
Input
key: Pointer to the memory buffer.
keySize: keySize in bytes.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

38 Treinta y ocho
38 Treinta y ocho

```
virtual bool versionNumber_get( /*[out]*/VersionNumber &rVersion ) throw () = 0;
```

Description: Gets the version number of SimpleLPR3_native.dll.
Parameters
Output:
rVersion:: The 4 element version number of SimpleLPR3_native.dll.
Returns: *true* if the method succeeded, *false* otherwise. In the later case use *ISimpleLPR::lastError_get* to obtain the error information.

39 Treinta y nueve
39 Treinta y nueve

6 .NET Interface Reference

See the companion SimpleLPR.chm help file.