



# JIDOSHA

## OCR/LPR

## JIDOSHALIGHT

LIBRARY FOR CHARACTER RECOGNITION WITH HIGH ASSERTIVENESS INDEX

# | Integration

Pumatronix Equipamentos Eletrônicos Ltda.

Rua Bartolomeu Lourenço de Gusmão, 1970. Curitiba, Brazil

Copyright 2020 Pumatronix Equipamentos Eletrônicos Ltda.

All rights reserved.

Visit our website <https://www.pumatronix.com/>

Send feedback about this document in the [suporte@pumatronix.com](mailto:suporte@pumatronix.com)

Information contained in this document is subject to change without notice.

Pumatronix reserves the right to modify or improve this material without obligation to notify you of the changes or improvements.

Pumatronix permits the downloading and printing of this document, provided that the electronic or hard copy of this document contains the entire text. Any change to this content is strictly prohibited.

## Change History

---

Date	Revision	Updated content
12/22/2022	1.0	Initial Version
06/17/2024	1.0.1	Update for firmware versions 3.22.0 to 3.26.0

## Overview

---

This document aims to guide the developer in the application of the *JidoshaLight* software library responsible for the recognition and automatic license plate reading (LPR) from image analysis and applicable in software compatible with the library. This document details the configuration options of the software development kit (SDK) and the APIs available.



**According to the library version applied to the software, some vehicle license plates formats may not be supported and some functions may be available only in the most current versions.**

---

# Summary

---

1.	JidoshaLight SDK Structure .....	6
	Supported APIs.....	6
2.	JidoshaLight Linux.....	7
	JidoshaLight Linux software architecture .....	7
	JidoshaLight Linux Restrictions .....	9
	JidoshaLight Linux Installation.....	9
	Hardkey Permissions Configuration .....	9
	Setting Environment Variables.....	9
	Preferences Archive Configuration .....	11
	Example Applications.....	12
3.	JidoshaLight Windows .....	13
	JidoshaLight Windows Installation .....	13
	Setting Environment Variables.....	14
	Preferences Archive Configuration .....	14
	Example Applications.....	14
4.	JidoshaLight Linux/FPGA.....	15
	JidoshaLight Linux/FPGA software architecture .....	15
	JidoshaLight Linux/FPGA Restrictions .....	16
	JidoshaLight Linux/FPGA installation .....	16
	License Setting .....	16
	Setting Environment Variables.....	17
	Setting Linux Kernel.....	17
	Example Applications.....	18
5.	JidoshaLight Android™ .....	18
	JidoshaLight Android™ Software Architecture .....	18
	JidoshaLight Android™ Restrictions.....	19
	Installing JidoshaLight Android™ .....	20
	Licensing.....	20
	Permissions.....	20
	Example Application .....	21

Package <code>br.gaussian.io</code> .....	21
Package <code>br.gaussian.jidoshalight</code> .....	21
Package <code>br.gaussian.jidoshalight.camera</code> .....	21
Package <code>br.gaussian.jidoshalight.sample</code> .....	21
<b>6. User APIs</b> .....	<b>25</b>
Known Limitations.....	25
JidoshaLight C/C++ API.....	25
JidoshaLight C/C++ API (Local).....	25
JidoshaLight C/C++ API (Synchronous Remote).....	42
JidoshaLight C/C++ API (Asynchronous Remote).....	43
JidoshaLight C/C++ API (Server).....	48
Java JidoshaLight API.....	50
Java JidoshaLight API (Local).....	50
JidoshaLight Java API (Asynchronous Remote).....	61
Java JidoshaLight API (Server).....	65
Java JidoshaLight API (IO/Mjpeg).....	67
Migration Guide - API 1 C/C++ JIDOSHA.....	69
<b>7. JIDOSHA User APIs</b> .....	<b>70</b>
jidoshaCore.h.....	71
API1 JIDOSHA C/C++.....	74
Types.....	74
Methods.....	74
API2 JIDOSHA C/C++.....	76
Types.....	76
Methods.....	76
API 2 - Configuration.....	80
API 2 - Image Perspective Configuration.....	84
API JIDOSHA C# / VB.NET.....	86
API 1.....	86
JIDOSHA C# / VB.NET API Examples.....	88
API JIDOSHA Delphi.....	90
API 1.....	90

---

Example JIDOSHA Delphi API.....	90
API JIDOSHA Java.....	91
API 1 .....	91
JIDOSHA Java API Example.....	91
Legacy API Special Builds .....	92

# 1. JidoshaLight SDK Structure

All paths used in this manual are relative to the root directory of the *JidoshaLight\_TARGET\_x.y.z* SDK. The SDK is JidoshaLight's software development kit consisting of:

- libjidoshaLight.so, libjidoshaLightRemote.so, libjidoshaLightJava.so plate recognition libraries;
- respective APIs of the libraries;
- by wrappers (bindings) for other languages;
- by pre-compiled example applications;
- by the source code of these applications;
- by a basic compilation script;
- by the Integration Manual;
- by a plate image for testing.

The SDK data packet structure contains:

Folder	Content
<i>res</i>	folder containing the manuals and a photo to use as an example in the software execution
<i>lib</i>	folder containing the libraries (.so or .dll)
<i>includes</i>	folders with the headers to be included in C applications
<i>sample</i>	folder containing the sample codes for C *Note: In the sample/bin folder are mini programs to test the operation.
<i>wrappers</i>	folder containing bindings for other language types
<i>jidoshapc</i>	folder containing bindings to the old <i>Jidosha</i> interface

## Supported APIs

Jidosha's primary Application Programming Interface (API) is the *JidoshaLight C/C++* API and can be found within the *include* and *lib* folders. Wrappers (bindings) for other languages are provided along with the SDK and are inside the *wrappers* folder:

- 1) JidoshaLight C/C++;
- 2) JidoshaLight Java (1.7+);
- 3) JidoshaLight Android;
- 4) JidoshaLight Python (2.7 and 3.x);
- 5) JidoshaLight C#.

For integration with other languages not yet supported, please contact Technical Support.

The SDK also provides a set of legacy APIs within the legacy folder. These APIs do not receive new functionality and exist only to ensure support for legacy applications developed from *Jidosha* (version 1.7.0 or lower). Internally this API uses the standard *JidoshaLight C/C++* API and therefore generates the same recognition results.

### Attention to APIs that are NOT recommended for new designs:



- 1) **jidoshapc C/C+**
- 2) **jidoshapc Java (1.6+)**
- 3) **jidoshapc Python (2.7)**
- 4) **jidoshapc Delphi (Windows only)**

5) `jidoshapc C#`

## 2. JidoshaLight Linux

---

The *JidoshaLight Linux* software library was created to work in conjunction with the hardkey (security key) that came with the library. That is, for the correct functioning of the library, the referred hardkey must be connected to the USB of the environment in which the library will be used. There are two hardkey versions, one for general use and the demo version, with expiration date. When its expiration date expires, the library automatically returns empty plates. If your demo hardkey expires and you wish to purchase a license or extend the demo period, please contact Pumatronix.

Check the installation prerequisites explained in the Product Manual.

### JidoshaLight Linux software architecture

---

Calls to the library API can be made locally or remotely over an IP network.

Local calls are executed on the same thread where the call was made. For licenses with more than 1 thread enabled or for cases where the main thread cannot be locked while the image is being processed, new threads must be created for processing.

Remote calls can be synchronous or asynchronous. In both cases the calls are made locally, and the images are processed remotely on a server. The license to use is required only on the server running the algorithm and is not required for remote library use.

Synchronous calls are blocking and return the processing result at the end of the call.

In the case of the asynchronous interface, the call returns immediately and the processing result is returned through a user *callback*.

The following figure presents a diagram with the architecture suggested for a Linux application that uses the JidoshaLight library with local calls, whether single thread or multithread. For the application to function correctly, the library *'libjidoshaLight.so'* must be linked to the application and the *hardkey* must be plugged into the machine. Then, for the single thread case, just call the API functions. As for the multithread case, the application must create the necessary processing threads and, from them, make the calls to the API functions of the *JidoshaLight* library.



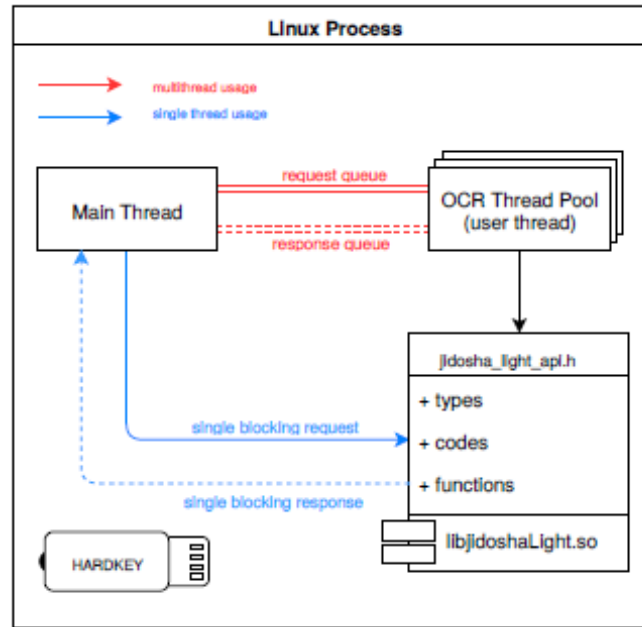


Figure 1 – Diagram with the suggested architecture for a Linux application

The following figure shows the suggested architecture for using the library with remote calls. For the application to function correctly, the 'libjidoshaLightRemote.so' library must be linked to the client application. The library 'libjidoshaLight.so' must be linked to the server application and the hardkey must be plugged into the machine. Client and server applications must be interconnected through a real or virtual TCP/IP network (loopback, for example). Although not illustrated in the figure, in the same way that for local calls the client application may have several threads, and the server may limit the number of active concurrent sessions depending on the license.

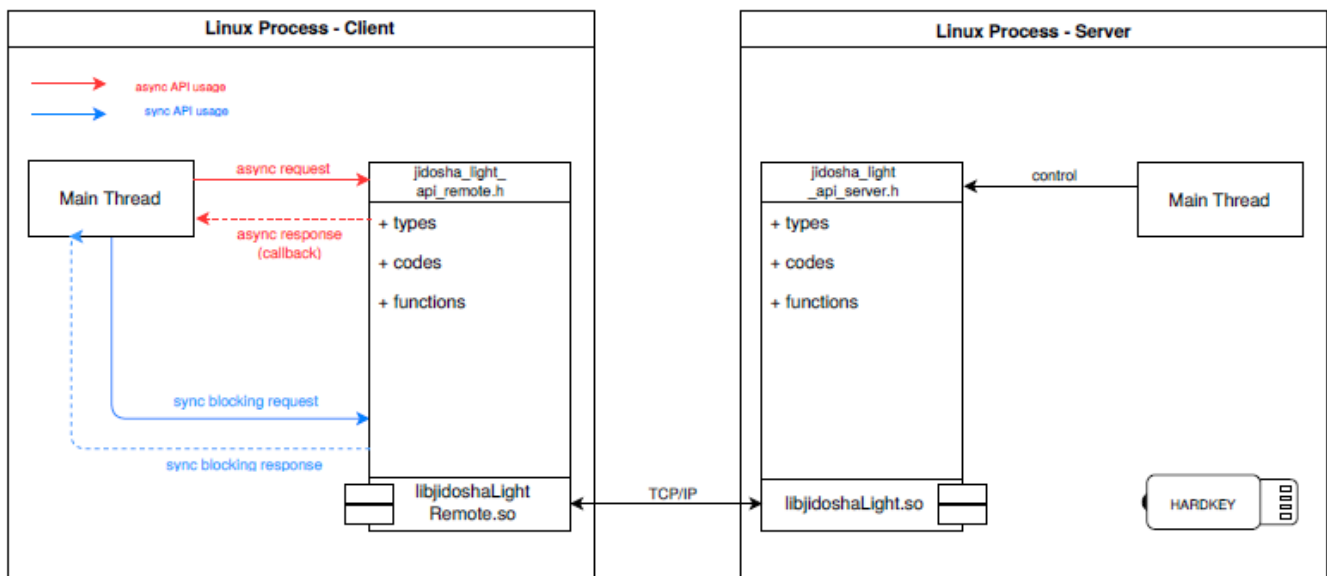


Figure 2 – Diagram with the suggested architecture for using the library with remote calls

## JidoshaLight Linux Restrictions

The library supports multithread and multiprocess applications, with the maximum number of threads of all processes limited by the license purchased.

The library does not support process *fork*.

## JidoshaLight Linux Installation

### Hardkey Permissions Configuration

For the correct operation of the USB hardkey, the access permissions of the **udev** must be changed. For its ease is included in the Jidosha SDK the script '`res/scripts/install_udev.sh`' running it with the argument '`-i`' will be installed the permissions automatically, it is also possible to run the script without argument to view the options. If you prefer to install manually it is necessary to follow the following procedures:

Add the following line:

```
ATTRS{idVendor}=="0403", ATTRS{idProduct}=="c580", MODE="0666"
```

at the end of the file corresponding to your Linux distribution:

```
Centos 5.2/5.4:      /etc/udev/rules.d/50-udev.rules
Centos 6.0 onwards: /lib/udev/rules.d/50-udev-default.rules
Ubuntu 7.10:       /etc/udev/rules.d/40-permissions.rules
Ubuntu 8.04/8.10:  /etc/udev/rules.d/40-basic-permissions.rules
Ubuntu 9.04 onwards: /lib/udev/rules.d/50-udev-default.rules
openSUSE 11.2 onwards: /lib/udev/rules.d/50-udev-default.rules
```

For Debian, add the lines:

```
SUBSYSTEM=="usb_device", MODE="0666"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", MODE="0666"
```

And at the end of the file:

```
Debian 6.0 onwards: /lib/udev/rules.d/91-permissions.rules
```



**For instructions on how to enable hardkey in other Linux distributions, contact Pumatronix Equipamentos Eletrônicos.**

## Setting Environment Variables

Before running the test applications supplied with the SDK, or any other application that uses the JidoshaLight Linux library, it is necessary to configure some environment variables for the correct functioning of the library.

Initially it is necessary to add the directory that contains the libraries to the system search path, as below:

```
$ export LD_LIBRARY_PATH=./lib:$LD_LIBRARY_PATH
```

## Logging and auditing system



**Attention: as of version 3.3.0 the library log system is DISABLED by default.**

The JidoshaLight SDK has a log system that can be used to audit the behavior of the library in the field. To enable some preconfigured debug messages simply export the environment variable `JL_LOGCFG` with the value "default".

```
$ export JL_LOGCFG=default
```

The log system also allows you to enable other debug messages and redirect the contents of these messages to one or more files. This functionality is configured through a configuration file whose structure is specified below.



**Reading the configuration file occurs only 1 time during the library load and has the following search order:**

- 1. absolute path indicated by the environment variable `JL_LOGCFG` (if set)**
- 2. `jlog.conf` file in current directory `./`**

Structure of the log system configuration file:

```
# JLog Configuration File
# This is a comment line in a JLog configuration file
# Entry format:
# TOPIC; LEVEL; TAG_FMT, FILES {comma separated}; SIZES {comma separated}
#
# Special Files
# [STDOUT] - prints to the screen (size always 0)
STDERR ; CRITICAL ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
MSGSERVER ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
MSGSERVER ; DEBUG ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
MSGSERVER ; WARN ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
MSGSERVER ; NOTICE ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
MSGSERVER ; CRITICAL ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
ANPRMSG ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
ANPRMSG ; DEBUG ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
ANPRMSG ; WARN ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
ANPRMSG ; NOTICE ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
ANPRMSG ; CRITICAL ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
LOGGER ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
LOGGER ; CRITICAL ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
LICENSE ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
LICENSE ; DEBUG ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
LICENSE ; WARN ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
LICENSE ; NOTICE ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
LICENSE ; CRITICAL ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
HARDWARE ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
HARDWARE ; CRITICAL ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
JLIB ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
JLIB ; CRITICAL ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
MSGANPR ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
MSGANPR ; DEBUG ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
VLOOP ; INFO ; SIMPLE_TS ; [STDOUT], log.txt ; 0, 20MB
```

The above log configuration file will cause the library to generate all enabled messages, both for the file with relative path `'log.txt'` and for the default output (stdout). If you want to inhibit one or more types of messages, just comment the line with `'#'` or remove it.

To inhibit messages to stdout and write only to the `log.txt` file, follow the example below for each type of message you want:

```
MSGANPR ; INFO ; SIMPLE_TS ; log.txt ; 20MB
```

## Preferences Archive Configuration

The library allows optional use of a preferences file. Through this file it is possible to configure the fields of the `'struct JidoshaLightConfig'`, overwriting the fields of this `'struct'` passed by the API. The format is json, as follows:

```
{
  "jidosha-light" : {
    "config" : {
      "vehicleType" : 3,
      "processingMode" : 4,
      "timeout" : 0,
      "countryCode" : 76,
      "minProbPerChar" : 0.85,
      "maxLowProbabilityChars" : 0,
      "lowProbabilityChar" : "?",
      "avgPlateAngle" : 0.0,
      "avgPlateSlant" : 0.0,
      "maxCharHeight" : 0,
      "minCharHeight" : 0,
      "maxCharWidth" : 0,
      "minCharWidth" : 0,
      "avgCharHeight" : 0,
      "avgCharWidth" : 0,
      "xRoi" : [0,0,0,0],
      "yRoi" : [0,0,0,0],
      "ENABLE_CONFIG_OVERRIDE" : true
    }
  }
}
```

By default, the library looks for the `jl_anpr_preferences.json` file in the working directory. If the file exists, it will be loaded; otherwise, it will be searched in the path indicated by the environment variable `JL_ANPR_PREFS`. If the variable does not exist, no preferences file is loaded. If it exists and the path indicated is a valid file, it is uploaded.

If a preferences file has been loaded, the preferences present in it will only be applied if the `'ENABLE_config_OVERRIDE'` field is `'true'`. Missing `'struct JidoshaLightConfig'` fields from the preferences file will receive the default value as defined by the library.

Because the preferences file is loaded at library startup (usually at the beginning of the process that uses it), modifications to the file will only take effect when the library is reloaded. This behavior may be changed in future versions.

## Example Applications

The SDK includes some sample applications with included source code:

- *JidoshaLightSample*: Example of local processing
- *JidoshaLightSampleClient*: Example of client application with asynchronous remote processing
- *JidoshaLightSampleServer*: Example of server application
- *JidoshaLightSampleAsync*: Example of asynchronous local processing with multiple threads
- *JidoshaLightSampleMulti*: Example of recognizing multiple plates in the same image

If you want to recompile the examples, use the `make_samples.sh` script:

```
$ cd sample/src && CXX=arm-none-linux-gnueabi-g++ && source make_samples.sh
```

After configuring the hardkey and environment variables and connecting the hardkey it will be possible to run the examples.

To run *JidoshaLightSample*, run the sample program with the reference plate image from the terminal:

```
$ ./sample/bin/JidoshaLightSample ./res/640x480.bmp
```

The application should inform the version of the library as well as the result of the image recognition.

```
-- JidoshaLight Sample Application --  
Library Info  
Version: x.y.z  
SHA1: abcdefghijklmnopqrstuvwxyz  
  
-> Processing: ./res/640x480.bmp  
PLATE: AJK7722 - PROB: 0.9944 - POSITION: (258,338,142,27) - TIME: 419.03 ms
```

To run the *JidoshaLightSampleServer* and *JidoshaLightSampleClient* examples, run the server program from a terminal:

```
$ ./sample/bin/JidoshaLightSampleServer
```

The application should inform that it was initialized using the TCP 51000 port and the hardkey was found.

```
[2016:08:30 15:08:16.620245 : LOGGER : 0x0001 : INFO] -> Logger session started  
Starting server with 1 thread(s), queue size: 10, queueTimeout: 0 ms, 1 connection(s),  
port: 51000  
[2016:08:30 15:08:16.621808 : MSGSERVER : 0x0001 : INFO] -> Started server at port 51000  
[2016:08:30 15:08:16.631327 : HARDWARE : 0x0007 : INFO] -> Hard key attached  
[2016:08:30 15:08:17.169088 : HARDWARE : 0x0008 : INFO] -> Found valid hard key
```

Then, in another terminal, run the client program. The client must inform the library version as well as the result/statistic of the image recognition:

```
$ ./sample/bin/JidoshaLightSampleClient resources/images/640x480.bmp  
=====  
Remote API: 127.0.0.1@51000  
Threads: 1  
Thread queue size: 5  
Compilation_Date: Aug 30 2016 - 15:08:10  
Images: 1  
=====  
PLATE: AJK7722 - PROB:0.9944 - ELAPSED: 14.35 ms - returncode: 0  
-- Library --  
Version: 2.1.0  
Build SHA1: d86e07e560206cb418fdc47b1c5108d7ac76657b
```

```
Build FLAGS: I686;Linux_32;DEBUG_LEVEL=DEBUG_LV_LOG;JDONGLE_VENDOR_MODE;...

-- Total --
TotalTime: 14.35 ms (CPU: 14.35 ms)
Plates: 1
NonEmpty: 1 - 100.00 %
AverageTime: 14.35 ms

-- Load/Decode --
ElapsedTime: 0.83 ms
AverageTime: 0.83 ms (5.77 %)

-- Localization --
ElapsedTime: 7.01 ms
AverageTime: 7.01 ms (48.83 %)

-- Segmentation --
ElapsedTime: 0.69 ms
AverageTime: 0.69 ms (4.81 %)

-- Classification --
ElapsedTime: 5.72 ms
AverageTime: 5.72 ms (39.88 %)
```

Returning to the server terminal, check the additional log messages for license data and connection events:

```
[2016:08:30 15:11:24.710395 : LICENSE : 0x0006 : INFO] -> Software license to GAUSSIAN,
max.
threads 16, max. connections 16
[2016:08:30 15:11:24.710421 : LICENSE : 0x0001 : INFO] -> Valid license found 0x2137069056
[2016:08:30 15:11:24.857709 : MSGSERVER : 0x0005 : INFO] -> Accepted connection:
127.0.0.1@51000
[2016:08:30 15:11:25.563783 : MSGSERVER : 0x0007 : NOTICE] -> Dropped connection:
127.0.0.1@51000
```

### 3. JidoshaLight Windows

The *JidoshaLight Linux* software library was created to work in conjunction with the hardkey (security key) that came with the library. That is, for the correct functioning of the library, the referred hardkey must be connected to the USB of the environment in which the library will be used. There are two hardkey versions, one for general use and the demo version, with expiration date. When its expiration date expires, the library automatically returns empty plates. If the demo hardkey expires, it is possible to purchase a license or extend the demo period by contacting Pumatronix.

Check the installation prerequisites explained in the Product Manual.

#### JidoshaLight Windows Installation

For installation there is only the need to plug the hardkey into a windows machine that will run the software, then windows should install a driver automatically the first time. To test the installation occurred correctly you can run the example applications, detailed in [Preferences Archive Configuration](#).

## Setting Environment Variables

Before running the test applications supplied with the SDK, or any other application that uses the JidoshaLight Linux library, it is necessary to configure some environment variables for the correct functioning of the library.

1. Initially, it is necessary to add the directory containing the libraries to the system search path, to access the SDK folder and enter the command:

```
$ set PATH=./lib;%PATH%
```



**NOTE: The previous command will only change the PATH for the open terminal session, if you need to configure for the environment, it is necessary to access the control panel > system > change settings > system properties > advanced > environment variables and there change the value of the system variable, or user variable, called Path.**

Logging and auditing system

See [Logging and auditing system](#) used in JidoshaLight Linux.

## Preferences Archive Configuration

See in [Preferences Archive Configuration](#) used in JidoshaLight Linux.

## Example Applications

The SDK includes some sample applications with included source code:

- *JidoshaLightSample*: Example of local processing
- *JidoshaLightSampleClient*: Example of client application with asynchronous remote processing
- *JidoshaLightSampleServer*: Example of server application
- *JidoshaLightSampleAsync*: Example of asynchronous local processing with multiple threads
- *JidoshaLightSampleMulti*: Example of recognizing multiple plates in the same image
- *JidoshaLightSampleServerService*: Example of server as a Windows service

To run *JidoshaLightSample*, from the SDK folder run the command below and you must get a similar output:

```
>sample\bin\JidoshaLightSample.exe .\res\640x480.bmp
[year:month:day time : LOGGER : 0x0001 : INFO] -> JLib log session started
[year:month:day time : JLIB : 0x0004 : INFO] -> JLib singleton created

-- JidoshaLight LPR Sample Application - 64-bit --
Compilation Date: month day year hourly
Library Info
Version: x.y.z
SHA1: sha1
[year:month:day time : HARDWARE : 0x0008 : INFO] -> Hardkey attached
[year:month:day time : HARDWARE : 0x000A : INFO] -> Hardkey access valid
[year:month:day time : LICENSE : 0x0002 : INFO] -> Licensed to company, product LPR,
threads 4, connections 1, serial 150089957 (0x8f230e5), TTL: -1
-- LicenseInfo --
>> Serial: 0x8f230e5
>> Customer: company
>> State: 0
>> TTL: -1 hours
>> MaxThreads: 4
```

```
>> MaxConections: 1
FILE: ..\..\res\640x480.bmp - PLATE: AJK7722 - COUNTRY: 76 - PROB: 0.9912 - POSITION:
(258,339,142,25) - TIME: 12.41 ms

Exiting
[year:month:day time : JLIB      : 0x0002 : INFO] -> JLib network module stopped
[year:month:day time : JLIB      : 0x0005 : INFO] -> JLib singleton destroyed
[year:month:day time : LOGGER    : 0x0002 : INFO] -> JLib log session stopped
```

## 4. JidoshaLight Linux/FPGA

The *JidoshaLight Linux* software library with FPGA acceleration is licensed from a hardware-linked license file, without the need to use hardkey (security key). This library supports hardware acceleration based on Xilinx FPGAs from the Zynq-7000 family. By default, it has support for the *XC7Z020-CLG400* device, and can be adapted for larger capacity devices.

Check the installation prerequisites explained in the Product Manual.

### JidoshaLight Linux/FPGA software architecture

The software architecture is like that of the non-accelerated Linux version, described in [JidoshaLight Linux](#).

The main differences are in the additional device programming interfaces and in the shared memory reserved area. The configuration and installation details are specific and are described in Installation.

The following figures illustrate the architecture suggested for both local and remote APIs.

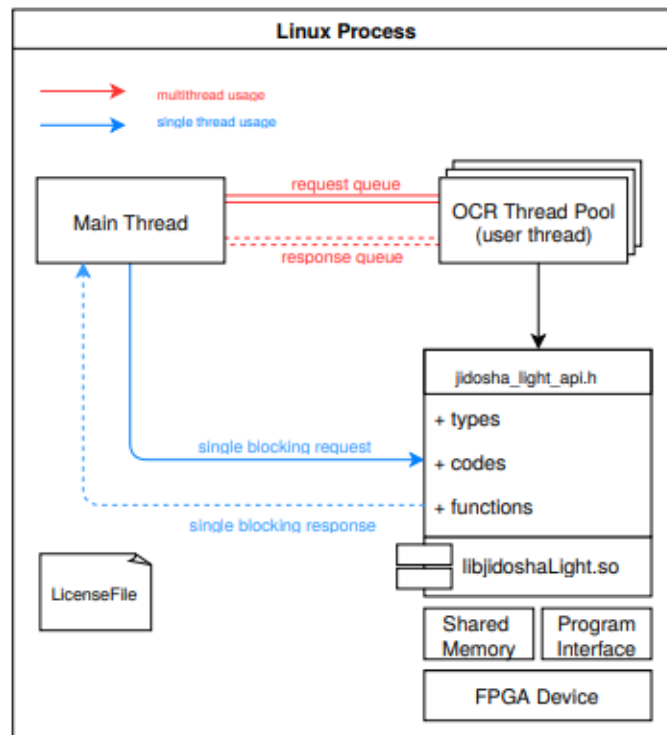


Figure 3 – Diagram with local API use cases





It is expected that in the first license plate reading call JidoshaLight will take longer than in subsequent calls, as the first call loads important information used by Jidosha into the computer's memory.

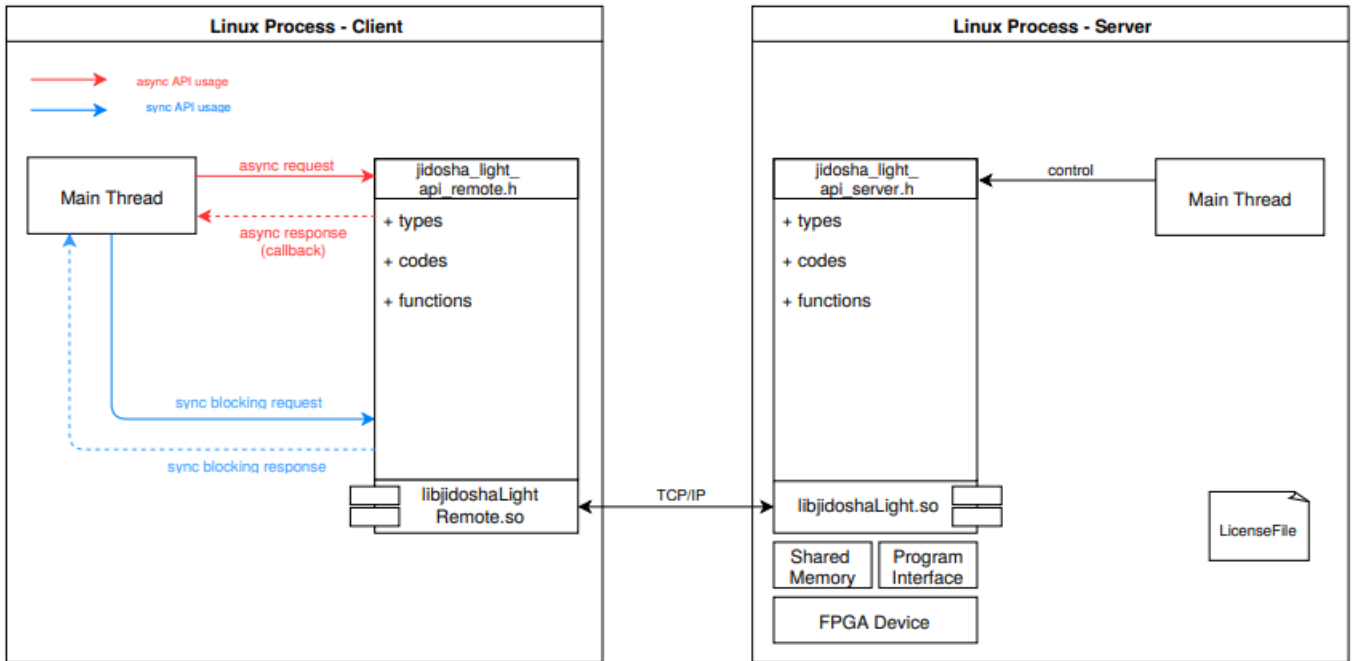


Figure 4 – Diagram with remote API use cases

## JidoshaLight Linux/FPGA Restrictions

The FPGA-accelerated library supports multithreaded applications, with the maximum number of threads limited by the license purchased. Multiprocess applications are not supported.

## JidoshaLight Linux/FPGA installation

### License Setting

The library is licensed through a file linked to the device used.

To obtain the identifier of your hardware it is necessary to run the *JidoshaLightDna* application from the terminal of the device properly configured as described in [Setting Environment Variables](#).

```
$ ./tools/JidoshaLightDna
0xFEDCBA9876543210
```



**IMPORTANT: Concurrent use of this application with any other using the library is not allowed and may cause crashes.**

## Setting Environment Variables

Before running the test applications supplied with the SDK, or any other application that uses the JidoshaLight Linux library, it is necessary to configure some environment variables for the correct functioning of the library.

When using the version with FPGA acceleration, in addition to the variables described in [Setting environmental variables](#), the settings described below are necessary:

- `JL_MPOOL_BASE`: Memory base address for communication between library and FPGA. If not set, the default value is `0x3A000000`. E.g.:

```
$ export JL_MPOOL_BASE=0x3A000000
```

- `JL_MPOOL_BUFFERNUM`: Number of memory buffers required to run the library. If not defined, the default value is 32 buffers, with the minimum required value being 12 \*buffers\* per thread that uses the library concurrently. E.g.:

```
$ export JL_MPOOL_BUFFERNUM=32
```

- `JL_MPOOL_BUFFERSIZE`: Size of each memory buffer, requiring a multiple of 4096 bytes. If not set, the default value is 2097152 \*bytes\* (2MB). This is the value required to process images up to 800x600 pixels. This value needs to be greater than 4 times the image resolution. E.g.:

```
$ export JL_MPOOL_BUFFERSIZE=2097152
```

- `JL_LICENSE_FILE`: Path to the license file. The license file is linked to the device used. For the identifier, see License Setting. E.g.:

```
$ export JL_LICENSE_FILE=./license.bin
```

## Setting Linux Kernel

For communication between the library and the FPGA device, two interfaces are required, one dedicated for configuration of the FPGA and a shared memory for data exchange.

The configuration interface is provided by Xilinx via a *char device* (`/dev/xdevcfg`) and is not currently part of the standard Linux *kernel*. Source code and instructions for installation can be found on the [Xilinx Wiki page](#).

Shared memory must be visible in `/dev/mem` and reserved for library use only, and cannot be used by the Linux \*kernel\*.

To do so, it is necessary to limit the amount of memory used by the *kernel* when booting it.

Below is an example for reserving the last 96MB of memory from a device with 1GB of RAM. In u-boot, configure:

```
set bootargs 'root=/dev/ram mem=928M rw'
```

To make the `/dev/mem` device available, use `CONFIG_DEVMEM=y` in `kconfig` in the kernel build process.

Also add the following settings to the Linux device tree (DTS):

```
memory {
    device_type = "memory";
    reg = <0x3A000000 0x6000000>;
};
```

```
reserved-memory {
    #address-cells = <1>;
    #size-cells = <1>;
    ranges;

    linux,cma {
        compatible = "shared-dma-pool";
        reusable;
        size = 0x6000000;
        alignment = 0x1000;
        linux,cma-default;
    };
};
```

## Example Applications

See [Example Applications](#) used for JidoshaLight Linux.

## 5. JidoshaLight Android™

The *JidoshaLight Android™* software library is designed to work in conjunction with the license file that must be generated after the application is installed by the user. The license file is generated per installation and is linked to the device hardware, requiring a new license in case of reinstallation of the application or modification of the device hardware, including the SIM card of the device. Replacing the battery does not require a new license. For temporary licenses, released for a limited time, the date and time of the equipment must be synchronized with the cellular network.

The library supports multithreaded applications, with the maximum number of threads and the minimum processing time limited by the license purchased. For the use of the server API, the maximum number of simultaneous connections it accepts is also limited by the license.

*JidoshaLight Android* library features are accessed via the Java API. This version is compatible with arm™ processors (armv7-a) with Android™ 4.4 or higher operating system for library use (shared libraries and basic Java classes) and Android™ 8 or higher for demo application installation.

### JidoshaLight Android™ Software Architecture

The most recommended way to work with the JidoshaLight library on Android platform is through the *asynchronous client and server* topology. This topology allows optimizing the flow of the plate recognition process, since all memory processing and allocation takes place in native code. This topology also makes it possible to process the images externally without changes in the application. The demo application that accompanies the SDK implements this topology.

Usually, a better user experience is obtained in *freeflow* mode. In *freeflow* mode, as opposed to "*point and shoot*", the plate recognition process is done on all images sent by the camera, without the need for intervention (shot) by the user. Thus, as soon as the camera is triggered, processing begins and callbacks with recognition results are generated asynchronously. The **asynchronous client and server** topology can work in *freeflow* mode without any significant change in application implementation.

Asynchronous freeflow customer **positives** points:

1. Simplification of application code, facilitating library integration
2. Improved user experience (faster recognitions)
3. Automatic resource management (queues, threads, network)
4. Greater decoupling between image acquisition (camera), processing (LPR) and output (UI and DB)
5. Local or remote processing capability without source code modification

Asynchronous freeflow client **negative** points:

1. Callbacks occur in a separate thread to the UI thread, requiring synchronization (runOnUiThread)
2. Callbacks are issued sequentially and cannot block (callback code must be light and fast)
3. Asynchronous error handling is usually more complex

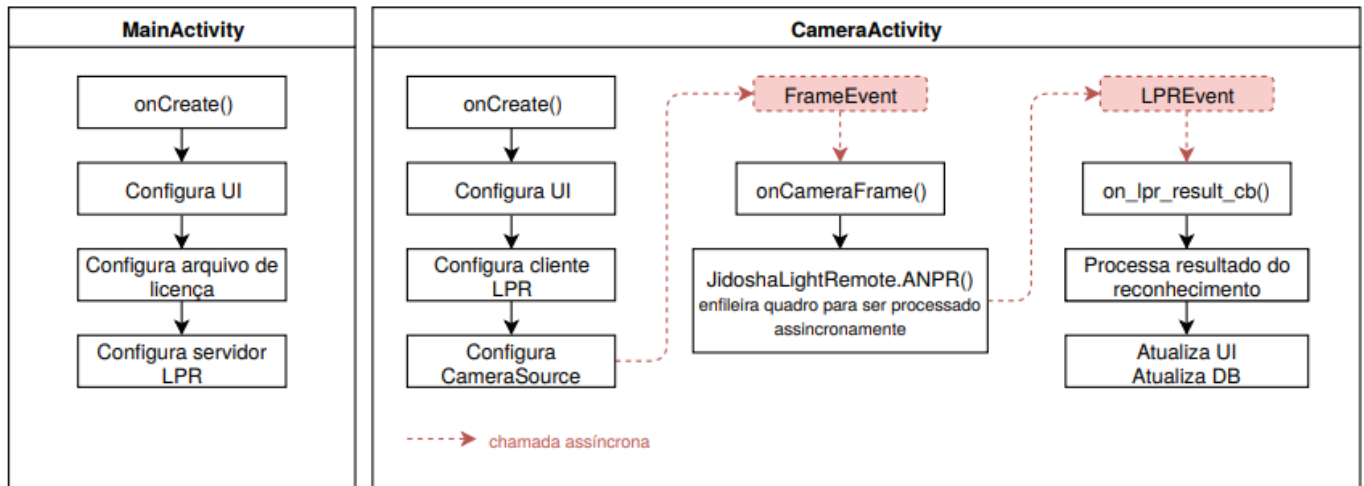


Figure 5 - Example of flow for a freeflow asynchronous client application: MainActivity configures the library license and starts the processing server, CameraActivity configures the plate reading client, the camera (CameraSource) and handles the events

## JidoshaLight Android™ Restrictions



**ATTENTION: The memory restrictions presented below do not apply to memory allocated natively (within the shared library). For high-performance applications, the use of the asynchronous client API is recommended.**

The Android™ operating system has strict restrictions on the use of RAM by applications. The maximum amount of memory an application can allocate in the *heap* varies between devices, but is around 24MB to 36MB. If an application tries to allocate more memory than it is allowed to, an *'OutOfMemoryError'* exception occurs and the application is terminated by the operating system.

Since the resolution of the cameras of smartphones and tablets is increasing, the developer must pay attention to the size of the images that he/she intends to recognize in order to mitigate the possibility of an exception of the type *'OutOfMemoryError'*. For example, an 8MP image in Bitmap format ARGB8888 occupies 24MB, which would be enough to overcome the memory limit on multiple devices.

Because JidoshaLight needs the **plate characters to be no more than 30 pixels tall**, an image with a resolution of **1280x720** is sufficient for plate recognition purposes. If you want to display a high-resolution image to the user, you can acquire and store the image in high-resolution and for processing use the resolution-reducing decoding methods supported by the Android™ class *'android.graphics.Bitmap'*. In this

case, one must take into account the reduction of the characters in the process of reducing the image size, ensuring the size of 15 to 30 pixels in the reduced image.

Another important specificity of the Android™ operating system is related to the locking of the graphical interface thread. By default, the graphical interface thread is the only one created by the application and all processing is performed on it. For the graphical interface to remain responsive to user actions, it should not block for more than a few milliseconds. If this occurs, the operating system will issue an alert to the user reporting that the application has stopped responding or will simply stop the application. For more information about memory management on Android:

- <https://developer.android.com/training/articles/memory.html>
- <https://developer.android.com/training/displaying-bitmaps/index.html?hl=en>

## Installing JidoshaLight Android™



**ATTENTION: All Java classes that have methods marked as native cannot have their package changed. All other classes can be moved freely.**

The reading library development SDK of *JidoshaLight* vehicle license plates for Android accompanies the API and the shared native C-language libraries (shared, which can be accessed by any JNI code), the *wrappers* and libraries for Java interface and a demo application described in [Example Application](#).

## Licensing

A valid license file is required for the operation of the *JidoshaLight* library on Android™ systems. The licensing is done by device and by time, requiring a new licensing if the equipment has its hardware characteristics changed or the term of the license has expired.

The `JidoshaLight.setLicenseFromData ()` API must be used to pass the contents of the license file to the library and must be made **before** any other call to other API functions. The `JidoshaLightAndroidHelper.java` class also brings some utility functions that help in the license loading process.

The license request procedure is fully automated by the `JidoshaLight.getLicenseFromServer` function, requiring only the user to register the Device ID with Pumatronix. The `LicenseManagerFragment.java` class of the sample application shows how to request the *Device ID* and how to request a server license.

For more information about licensing devices, please contact Technical Support.

## Permissions

The following permissions are required for the library to function and must be included in the application's *AndroidManifest.xml*:

```
<!-- Required permissions to use the library (required) -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<!-- Permissions required to use the device camera (optional) -->
<uses-feature android:name="android.hardware.camera" android:required="true" />
<uses-permission android:name="android.permission.CAMERA" />
<!-- Permissions required to use the MJPEG camera (optional) -->
<uses-permission android:name="android.permission.INTERNET"/>
```

## Example Application

---

The sample app that comes with the SDK is designed for use with Android™ Studio 4 or higher. It shows how to use the library to recognize the plates of a video stream coming from the back camera of your phone or an external MJPEG camera. It also brings an example of implementation to the configuration screen of the library parameters, Camera activity with grid and zoom support, recognition list, recognition details, licensing system and database integration to search for information related to the detected license plate.

### Package `br.gaussian.io`

- *Mjpeg.java*: Class wrapper over the native API of the MJPEG decoder. See the *MjpegCamera.java* class for a higher-level implementation.

### Package `br.gaussian.jidoshalight`

- *JidoshaLight.java*: Class containing local API functions (license plate recognition and licensing) and function return codes
- *JidoshaLightRemote.java*: Class containing asynchronous remote API functions
- *JidoshaLightServer.java*: Class containing server API roles

### Package `br.gaussian.jidoshalight.camera`

- *BaseCameraSource.java*: Base class for all camera deployments
- *BackCamera.java*: Implementation for smartphone rear camera
- *MjpegCamera.java*: Implementation for an external MJPEG camera
- *CameraFrame.java*: Class that stores a frame of a camera
- *CameraView.java*: View capable of displaying a stream of images from a *BaseCameraSource*; provides grid support, plate overlay, pinch zoom, and ROI selection

### Package `br.gaussian.jidoshalight.sample`

#### Common

- *common/JidoshaLightAndroidHelper.java*: Auxiliary class containing support methods for the process of reading and writing the license file, in addition to other utility methods.
- *common/JidoshaLightServerHelper.java*: Auxiliary class containing support methods for local LPR server startup.

#### Activities

- *MainActivity*: Main application activity, shows you how to configure the license file and initialize the local plate-reading server.

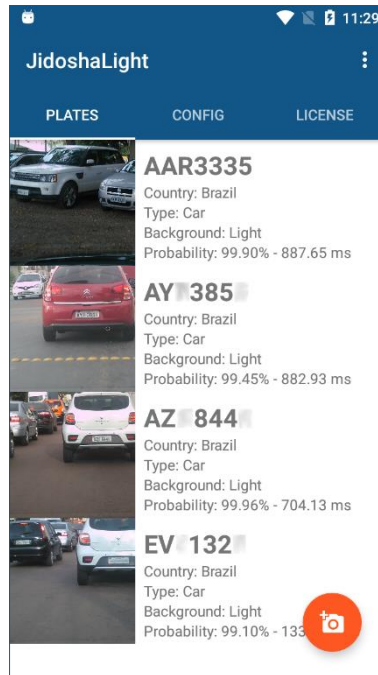


Figure 6 – MainActivity

- *DetailActivity*: Activity triggered when selecting an item from the recognition list. Expands the information of a given recognition.

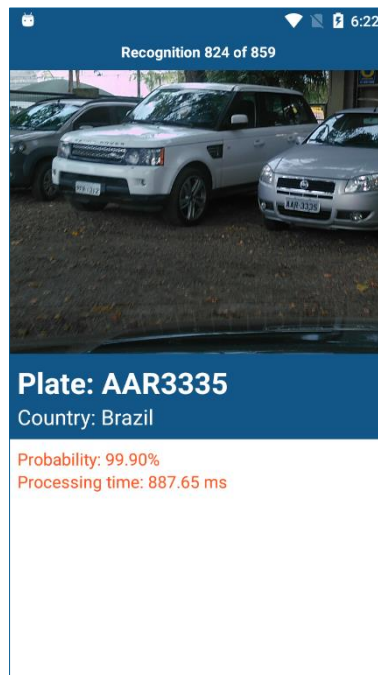


Figure 7 - DetailActivity

- *CameraActivity*: Exemplifies the process of configuring and capturing camera images, allowing:
  - 1) Instantiate a camera from the settings;
  - 2) Set up a plate processing client;
  - 3) Configure the optical zoom through the pinch movement;
  - 4) Select the region of interest (ROI) through touch;
  - 5) Enable/disable processing.

For better recognition performance, camera focus and zoom should allow you to capture images with good sharpness and size. The height of the plate should be between 30 and 50 pixels. The guides are intended to assist in framing the plate, ensuring the correct size and orientation of the plate at the time of capture. The plate should be approximately the size of a grid rectangle.

Since that the camera of the smartphone or tablet is constantly moving, it is ideal, when existing, to activate the auto-focus and video stabilization features of the device. Depending on the application, it is recommended to export manual focus and exposure adjustments for a better result.

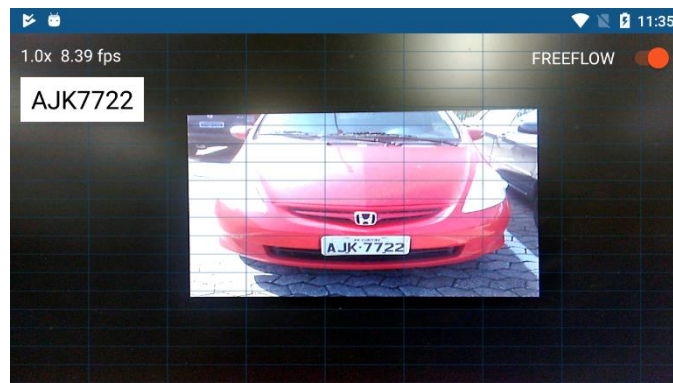


Figure 8 – CameraActivity: The ideal plate height for recognition must be the same as a grid rectangle (the plate does not need to be aligned with the grid)

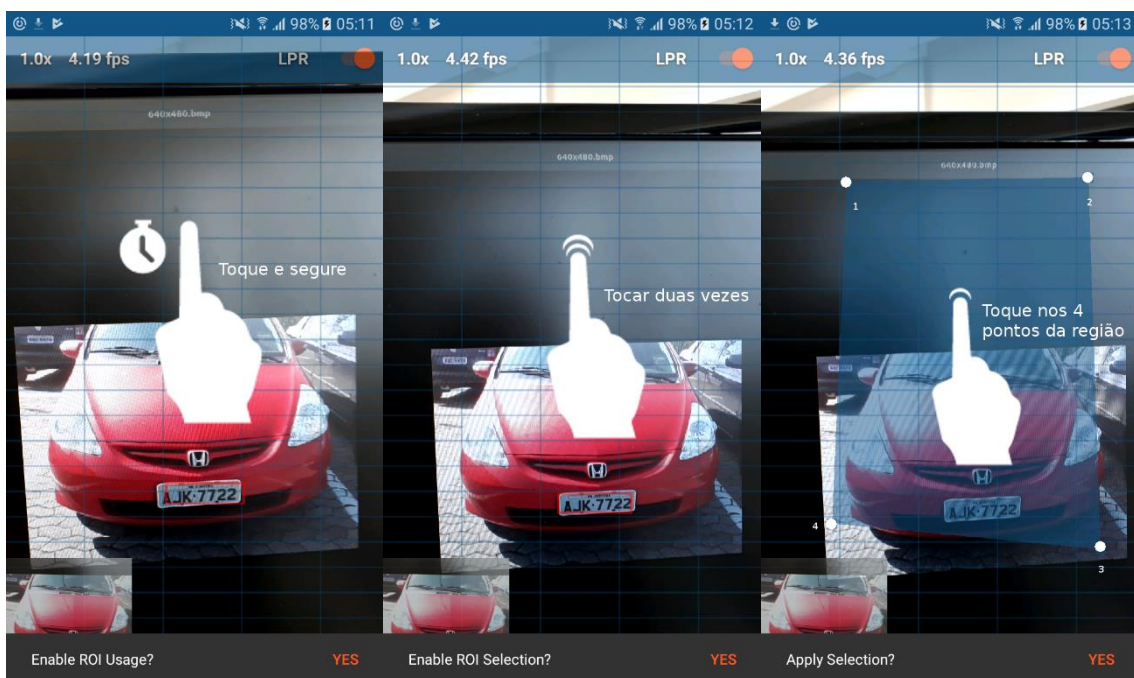
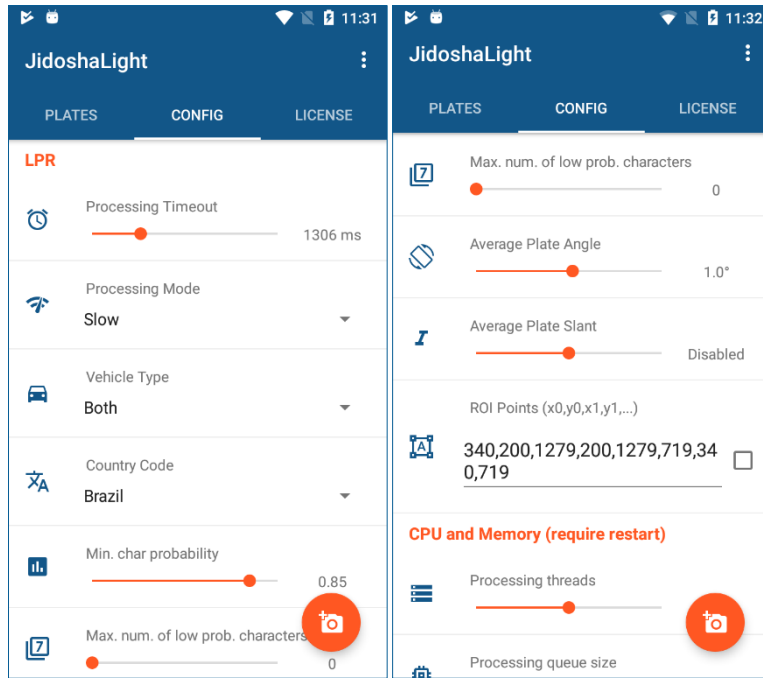


Figure 9 - ROI selection: 1) Tap and hold on the screen to enable the use of the ROI, 2) Double tap to start the selection of ROI points, 3) Tap on the four points that delimit the region (clockwise)

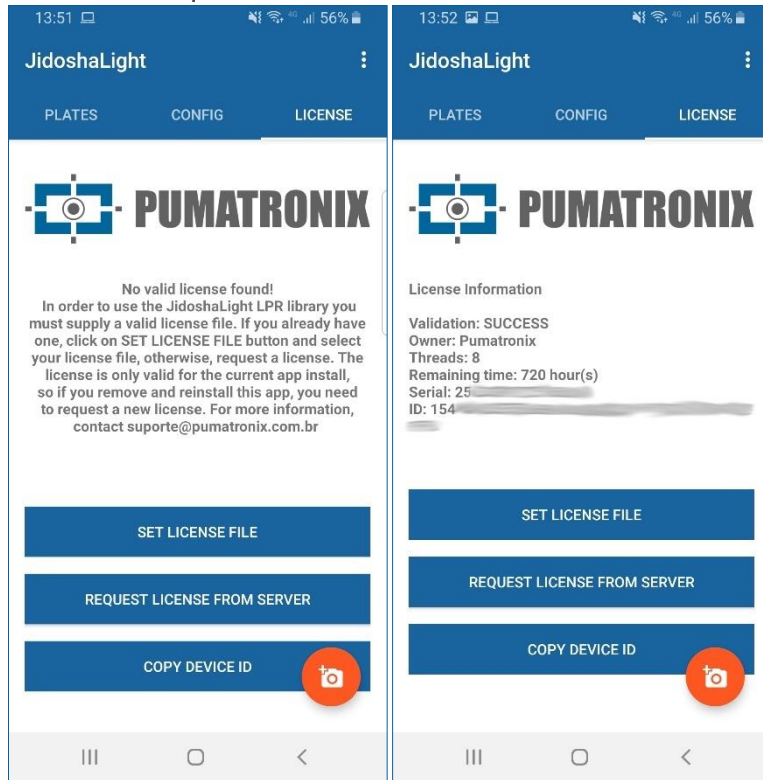
### Fragments

- **ConfigurationFragment:** Fragment used to display and configure JidoshaLight library parameters. Configurable parameters are the same as available in the Java/C API





- *LicenseManagerFragment*: Fragment used to implement the licensing system. Shows how to read the **Device ID** and how to request a server license file



**Changing some settings and the license file requires the application to be restarted.**

## 6. User APIs

The JidoshaLight library exports 4 distinct APIs for automatic plate recognition:

- *Local*
- *Synchronous Remote*
- *Asynchronous Remote*
- *Server*.

The Synchronous Remote API will be discontinued in the future and should not be used in new projects. In addition to the recognition APIs, the library provides some utility functions, such as a video receiver in MJPEG format and a license reader. These supplemental APIs are partially documented in this manual. For more information on use, see the headers in the `include/gaussian/common` folder.

By default, the API-supported languages that accompany the SDK are *C/C++* and *Java*. Wrappers in *Python*, *C#* and *Delphi* can be supplied on demand.

If you have any questions or support for other languages, please contact Pumatronix's Technical Support.

### Known Limitations

The following are the known limitations of the *JidoshaLight* library:

- 1) When the library is loaded dynamically (*LoadLibrary* on Windows, *dlopen* on Linux), it cannot be downloaded (*FreeLibrary* and *dlclose*, respectively).
- 2) On Linux, the process where the library runs cannot be *forked*.
- 3) In the case of concurrent use and in the same process as *JidoshaLight* with the *Jidosha Container* or *Jidosha Rail* library, *JidoshaLight* must be loaded last. This limitation will be removed in a future version of the libraries.

### JidoshaLight C/C++ API

The library's native Application Programming Interface (API) is written in C language, which makes it easy to create bindings for use in other languages. The entire API C is available through a set of headers within the *include* folder of the SDK.

#### JidoshaLight C/C++ API (Local)

The Local API contains the basic types, settings, and functions for local image processing. Since release 2.4.4 its contents are divided between *jidosha\_light\_api\_common.h* and *jidosha\_light\_api.h* files.



**To maintain compatibility with previous versions, header '*jidosha\_light\_api\_common.h*' is included with '*jidosha\_light\_api.h*'.**

```

jidosha_light_api_common.h
//=====
//  CODES
//=====
enum JidoshaLightVehicleType {
    JIDOSHA_LIGHT_VEHICLE_TYPE_CAR           = 1,
    JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO         = 2,

```

```

    JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH = 3
};

enum JidoshaLightMode {
    JIDOSHA_LIGHT_MODE_DISABLE = 0,
    JIDOSHA_LIGHT_MODE_FAST = 1,
    JIDOSHA_LIGHT_MODE_NORMAL = 2,
    JIDOSHA_LIGHT_MODE_SLOW = 3,
    JIDOSHA_LIGHT_MODE_ULTRA_SLOW = 4,

    /* the following values can be added to one of the above modes */
    JIDOSHA_LIGHT_LOCALIZATION_MODE_0 = 0 << 8,
    JIDOSHA_LIGHT_LOCALIZATION_MODE_1 = 1 << 8,
    JIDOSHA_LIGHT_LOCALIZATION_MODE_2 = 2 << 8
};

/* ISO 3166-1 */
enum JidoshaLightCountryCode {
    JIDOSHA_LIGHT_COUNTRY_CODE_CONESUL = 0,
    JIDOSHA_LIGHT_COUNTRY_CODE_SAFETYPLATES = 3,
    JIDOSHA_LIGHT_COUNTRY_CODE_ARGENTINA = 32,
    JIDOSHA_LIGHT_COUNTRY_CODE_BOLIVIA = 68,
    JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL = 76,
    JIDOSHA_LIGHT_COUNTRY_CODE_CHILE = 152,
    JIDOSHA_LIGHT_COUNTRY_CODE_COLOMBIA = 170,
    JIDOSHA_LIGHT_COUNTRY_CODE_COSTA_RICA = 188,
    JIDOSHA_LIGHT_COUNTRY_CODE_ECUADOR = 218,
    JIDOSHA_LIGHT_COUNTRY_CODE_FRANCE = 250,
    JIDOSHA_LIGHT_COUNTRY_CODE_ITALY = 380,
    JIDOSHA_LIGHT_COUNTRY_CODE_MEXICO = 484,
    JIDOSHA_LIGHT_COUNTRY_CODE_NETHERLANDS = 528,
    JIDOSHA_LIGHT_COUNTRY_CODE_PANAMA = 591,
    JIDOSHA_LIGHT_COUNTRY_CODE_PARAGUAY = 600,
    JIDOSHA_LIGHT_COUNTRY_CODE_PERU = 604,
    JIDOSHA_LIGHT_COUNTRY_CODE_EGYPT = 818,
    JIDOSHA_LIGHT_COUNTRY_CODE_USA = 840,
    JIDOSHA_LIGHT_COUNTRY_CODE_URUGUAY = 250
};

enum JidoshaLightReturnCode {
    /* success */
    JIDOSHA_LIGHT_SUCCESS = 0,
    /* basic errors */
    JIDOSHA_LIGHT_ERROR_FILE_NOT_FOUND = 1,
    JIDOSHA_LIGHT_ERROR_INVALID_IMAGE = 2,
    JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_TYPE = 3,
    JIDOSHA_LIGHT_ERROR_INVALID_PROPERTY = 4,
    JIDOSHA_LIGHT_ERROR_COUNTRY_NOT_SUPPORTED = 5,
    JIDOSHA_LIGHT_ERROR_API_CALL_NOT_SUPPORTED = 6,
    JIDOSHA_LIGHT_ERROR_INVALID_ROI = 7,
    JIDOSHA_LIGHT_ERROR_INVALID_HANDLE = 8,
    JIDOSHA_LIGHT_ERROR_API_CALL_HAS_NO_EFFECT = 9,
    JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_SIZE = 10,
    /* license errors */
    JIDOSHA_LIGHT_ERROR_LICENSE_INVALID = 16,
    JIDOSHA_LIGHT_ERROR_LICENSE_EXPIRED = 17,
};

```

```

JIDOSHA_LIGHT_ERROR_LICENSE_MAX_THREADS_EXCEEDED = 18,
JIDOSHA_LIGHT_ERROR_LICENSE_UNTRUSTED_RTC       = 19,
JIDOSHA_LIGHT_ERROR_LICENSE_MAX_CONNS_EXCEEDED  = 20,
JIDOSHA_LIGHT_ERROR_LICENSE_UNAUTHORIZED_PRODUCT = 21,
/* others */
JIDOSHA_LIGHT_ERROR_OTHER                       = 999
};

enum JidoshaLightReturnCodeNetwork {
    /* network errors */
    JIDOSHA_LIGHT_ERROR_SERVER_CONNECT_FAILED    = 100,
    JIDOSHA_LIGHT_ERROR_SERVER_DISCONNECTED      = 101,
    JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_TIMEOUT    = 102,
    JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_FULL       = 103,
    JIDOSHA_LIGHT_ERROR_SOCKET_IO_ERROR         = 104,
    JIDOSHA_LIGHT_ERROR_SOCKET_WRITE_FAILED     = 105,
    JIDOSHA_LIGHT_ERROR_SOCKET_READ_TIMEOUT    = 106,
    JIDOSHA_LIGHT_ERROR_SOCKET_INVALID_RESPONSE = 107,
    JIDOSHA_LIGHT_ERROR_HANDLE_QUEUE_FULL      = 108,
    JIDOSHA_LIGHT_ERROR_SERVER_CONN_LIMIT_REACHED = 213,
    JIDOSHA_LIGHT_ERROR_SERVER_VERSION_NOT_SUPPORTED = 214,
    JIDOSHA_LIGHT_ERROR_SERVER_NOT_READY       = 215
};

    /* Raw image pixel format */
enum JidoshaLightRawImgFmt {
    JIDOSHA_LIGHT_IMG_FMT_XRGB_8888 = 0,
    JIDOSHA_LIGHT_IMG_FMT_RGB_888   = 1,
    JIDOSHA_LIGHT_IMG_FMT_LUMA      = 2,
    JIDOSHA_LIGHT_IMG_FMT_YUV420    = 3
};

//=====
// TYPES
//=====
// JidoshaLightConfig
//=====
typedef struct JidoshaLightConfig
{
    int configId;                // Unique Configuration ID
    int vehicleType;            // Vehicle type
    int processingMode;         // Processing Mode
    int timeout;                // Processing timeout in milliseconds
    int countryCode;           // Plate Syntax Country

    float minProbPerChar;       // Range [0,1] - Minimal probability to accept a
                                // given character recognition
    int maxLowProbabilityChars; // Max number of characters whose propability is
lower
                                // than minProbPerChar to accept a recognition
    char lowProbabilityChar;    // ASCII encoded character that will replace
characters
                                // with probability lower than minProbPerChar

    float avgPlateAngle;        // Average plate angle
    float avgPlateSlant;        // Average plate slant
};

```

```
    int    maxCharHeight;           // Max acceptable char height in pixels (0 ==
default value)
    int    minCharHeight;          // Min acceptable char height in pixels (0 ==
default value)
    int    maxCharWidth;           // Max acceptable char width in pixels (0 ==
default value)
    int    minCharWidth;          // Min acceptable char width in pixels (0 ==
default value)
    int    avgCharHeight;          // Average char height in pixels (0 == default
value)
    int    avgCharWidth;           // Average char width in pixels (0 == default
value)

    int    xRoi[4];                // ROI points - x coords
    int    yRoi[4];                // ROI points - y coords

} JidoshaLightConfig;

//=====
// JidoshaLightRecognition
//=====
typedef struct JidoshaLightRecognitionInfo
{
    double  totalTime;
    double  localizationTime;
    double  segmentationTime;
    double  classificationTime;
    double  loadDecodeTime;
    int     libVersion[3];
    char    libSHA1[41];
} JidoshaLightRecognitionInfo;

typedef struct JidoshaLightRecognition
{
    int frameId;                    // Unique Recognition ID
    char plate[8];                  // Plate text + byte 0 (null-terminated string)
    float probabilities[7];         // Range [0,1] - Recognition probability of each
character

    int xText;                      // Plate up-left corner X coord
    int yText;                      // Plate up-left corner Y coord
    int widthText;                  // Plate Width
    int heightText;                 // Plate Height

    int xChar[7];                  // Individual character up-left corner X coord
    int yChar[7];                  // Individual character up-left corner Y coord
    int widthChar[7];              // Individual character width
    int heightChar[7];             // Individual character height

    int textColor;                 // 0: dark text over bright background,
// 1: bright text over dark background

    int isMotorcycle;              // 0: false, 1: true
    int countryCode;               // ISO 3166-1
```

```
JidoshaLightRecognitionInfo info; // Overall recognition benchmark information
} JidoshaLightRecognition;

//=====
// JidoshaLightLicenseInfo
//=====
typedef struct JidoshaLightLicenseInfo
{
    uint64_t serial;
    char customer[64];
    int maxThreads;
    int maxConnections;
    int state;
    int ttl;
} JidoshaLightLicenseInfo;

//=====
// JidoshaLightRecognitionList
//=====
typedef struct JidoshaLightRecognitionList JidoshaLightRecognitionList;
JL_API JidoshaLightRecognitionList* jidoshaLight_ANPR_createList();
JL_API JidoshaLightRecognitionList*
jidoshaLight_ANPR_duplicateList(JidoshaLightRecognitionList* list);
JL_API int jidoshaLight_ANPR_destroyList(JidoshaLightRecognitionList* list);
JL_API int jidoshaLight_ANPR_getListSize(JidoshaLightRecognitionList* list);
JL_API const JidoshaLightRecognition*
jidoshaLight_ANPR_getListElement(JidoshaLightRecognitionList* list, int pos);

//=====
// JidoshaLightImage
//=====
typedef struct JidoshaLightImage JidoshaLightImage;
JL_API JidoshaLightImage* jidoshaLight_ANPR_createImage();
JL_API JidoshaLightImage* jidoshaLight_ANPR_duplicateImage(JidoshaLightImage* img);
JL_API int jidoshaLight_ANPR_destroyImage(JidoshaLightImage* img);
JL_API int jidoshaLight_ANPR_setImageLazyDecode(JidoshaLightImage* img, int enable);
JL_API int jidoshaLight_ANPR_loadImageFromFile(
    JidoshaLightImage* img,
    const char* filename
);
JL_API int jidoshaLight_ANPR_loadImageFromMemory(
    JidoshaLightImage* img,
    const uint8_t* buffer,
    int bufferSize
);
JL_API int jidoshaLight_ANPR_loadImageFromRawImgFmt(
    JidoshaLightImage* img,
    const uint8_t* buffer,
    int width,
    int height,
    int stride,
    JidoshaLightRawImgFmt fmt
);

//=====
```

```
// Library Information
//=====
JL_API int jidoshaLight_getVersion(int* major, int* minor, int* release);
JL_API const char* jidoshaLight_getBuildSHA1();          // ASCII encoded SHA1
JL_API const char* jidoshaLight_getBuildFlags();        // ASCII encoded Build Flags
JL_API int jidoshaLight_getLicenseInfo(JidoshaLightLicenseInfo* info);
JL_API int jidoshaLight_isRemoteApi();

//=====
// Utilities
//=====
JL_API const char* jidoshaLight_getReturnCodeString(int rc);
```

### jidosha\_light\_api.h

```
//=====
// PROCESSING
//=====
JL_API int jidoshaLight_ANPR_fromFile (
    const char* filename,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromMemory (
    const unsigned char* buffer,
    int bufferSize,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromLuma (
    unsigned char* luma,
    int width,
    int height,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromRawImgFmt (
    const unsigned char* buffer,
    int width,
    int height,
    int stride,
    JidoshaLightRawImgFmt fmt,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_fromImage(
    JidoshaLightImage* img,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);

JL_API int jidoshaLight_ANPR_multi_fromImage (
    JidoshaLightImage* img,
```

```
JidoshaLightConfig* config,
int maxPlates,
JidoshaLightRecognitionList* list
);
```

### Types

<b>enum JidoshaLightVehicleType</b>	
<b>Description</b>	Defines the types of plate that the OCR should look for in the image.
<b>Members</b>	<b>JIDOSHA_LIGHT_VEHICLE_TYPE_CAR</b> : car plates only <b>JIDOSHA_LIGHT_VEHICLE_TYPE_MOTORCYCLE</b> : motorcycle plates only <b>JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH</b> : both license plate types

<b>enum JidoshaLightMode</b>	
<b>Description</b>	Defines the processing strategies that can be used by the plate-reading algorithm. <i>FAST</i> uses the least computational effort possible for reading while <i>ULTRA_SLOW</i> uses the largest. The higher the level of effort, the greater the probability of reading the plate.
<b>Members</b>	<b>JIDOSHA_LIGHT_VEHICLE_TYPE_CAR</b> : car plates only <b>JIDOSHA_LIGHT_MODE_DISABLE</b> : value reserved for future use. Currently has the same effect as <i>ULTRA_SLOW</i> OPTION <b>JIDOSHA_LIGHT_MODE_FAST</b> : faster processing strategy, its use is advised for cases where processing time is critical <b>JIDOSHA_LIGHT_MODE_NORMAL</b> : Moderate processing strategy, has higher processing time and recognition index than <i>FAST</i> method <b>JIDOSHA_LIGHT_MODE_SLOW</b> : slow processing strategy, has higher processing time and recognition index than <i>NORMAL</i> method <b>JIDOSHA_LIGHT_MODE_ULTRA_SLOW</b> : super slow processing strategy, has the longest processing time and the highest recognition index One of the above modes must necessarily be used. In addition, one can optionally select the localization strategy used by the plate-reading algorithm. <i>LOCALIZATION_MODE_0</i> is the default. The other options currently only affect plate processing in Brazil. <b>JIDOSHA_LIGHT_LOCALIZATION_MODE_0</b> : localization strategy with longer processing time and higher recognition index <b>JIDOSHA_LIGHT_LOCALIZATION_MODE_1</b> : slightly faster localization strategy with slightly lower recognition index <b>JIDOSHA_LIGHT_LOCALIZATION_MODE_2</b> : fast localization strategy, with lower recognition index, applicable only to license plates, not motorcycles

<b>enum JidoshaLightCountryCode</b>	
<b>Description</b>	Sets the code of the countries supported by the recognition library in ISO 3166-1 format. The use of a particular country is limited by the license.
<b>Members</b>	<b>JIDOSHA_LIGHT_COUNTRY_CODE_CONESUL</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_SAFETYPLATE</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_ARGENTINA</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_BOLIVIA</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_CHILE</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_COLOMBIA</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_COSTA_RICA</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_ECUADOR</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_FRANCE</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_ITALY</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_MEXICO</b> <b>JIDOSHA_LIGHT_COUNTRY_CODE_NETHERLANDS</b>



	JIDOSHA_LIGHT_COUNTRY_CODE_PANAMA JIDOSHA_LIGHT_COUNTRY_CODE_PARAGUAY JIDOSHA_LIGHT_COUNTRY_CODE_PERU JIDOSHA_LIGHT_COUNTRY_CODE_EGYPT JIDOSHA_LIGHT_COUNTRY_CODE_USA JIDOSHA_LIGHT_COUNTRY_CODE_URUGUAY
--	---

enum JidoshaLightRawImgFmt	
<b>Description</b>	Sets the raw format types supported by the library.
<b>Members</b>	<b>JIDOSHA_LIGHT_IMG_FMT_XRGB_8888</b> : 32-bit XRGB format, with the least significant byte used for the blue channel (*Blue*). The most significant byte is ignored <b>JIDOSHA_LIGHT_IMG_FMT_RGB_888</b> : 24-bit RGB format, with the least significant byte used for the blue channel (*Blue*) <b>JIDOSHA_LIGHT_IMG_FMT_LUMA</b> : 8-bit format containing luminance channel only <b>JIDOSHA_LIGHT_IMG_FMT_YUV420</b> : 8-bit YUV format not interlaced with 4:2:0 subsampling

struct JidoshaLightConfig	
<b>Description</b>	Sets the raw format types supported by the library.
<b>Members</b>	<p><i>int configId</i>: field reserved for future use, has its value ignored by the library</p> <p><i>int vehicleType</i>: indicates the type of plate that the OCR should look for. See <a href="#">enum JidoshaLightVehicleType</a></p> <p><i>int processingMode</i>: indicates the processing strategy to be used. See <a href="#">JidoshaLightMode enum</a></p> <p><i>int timeout</i>: indicates the maximum time in milliseconds for plate recognition. The value '0' indicates that there is no timeout. A value other than '0' helps keep the average processing time low - processing is stopped and the function returns as soon as the timeout expires. The value should be determined based on the resolution of the image and CPU used</p> <p><i>int countryCode</i>: indicates the country code whose license plate is to be recognized. See <a href="#">JidoshaLightCountryCode enum</a></p> <p><i>float minProbPerChar</i>: value from '0.0f' to '1.0f' used to define the minimum probability that a given character of the plate must have to be considered valid (<b>recommended: 0.85</b>)</p> <p><i>int maxLowProbabilityChars</i>: maximum number of characters with probability less than 'minProbPerChar' for the recognized plate to be considered valid - a plate recognized as invalid is returned as an empty string '\0'</p> <p><i>LowProbabilityChar char</i>: character that will be used to replace those with a probability lower than 'minProbPerChar'</p> <p><i>float avgPlateAngle</i>: average angle of the plates in the images in relation to the horizontal axis</p> <p><i>float avgPlateSlant</i>: average angle of the slope of the characters in the images in relation to the vertical axis</p> <p><i>int maxCharHeight</i>: maximum acceptable character height, in pixels</p> <p><i>int minCharHeight</i>: minimum acceptable character height, in pixels</p> <p><i>int maxCharWidth</i>: maximum acceptable character width, in pixels</p> <p><i>int minCharWidth</i>: minimum acceptable character width, in pixels</p> <p><i>int avgCharHeight</i>: average character height, in pixels (default: 20)</p> <p><i>int avgCharWidth</i>: average character width, in pixels (default value: 7)</p> <p><i>int xRoi[4]</i> and <i>int yRoi [4]</i>: x and y coordinates of the four points of the region of interest (ROI) in any order.</p> <p>A region of interest is understood as a quadrilateral within the image where it is expected to find the plates to be recognized. The use of ROI benefits the processing time and the rate of hits, since it excludes shoulder regions or places of no importance for the plate recognition process. Setting all coordinates to zero will cause the ROI to be ignored and the entire image to be processed. Values larger than the image dimensions or negative result in the return of</p>

the `JIDOSHA_LIGHT_ERROR_INVALID_ROI` error code. The change of these values causes the recalculation of the ROI region, impacting the processing time of the first image after the change.



**Attention:** The coordinates of the ROI points have their origin (0.0) in the upper left corner of the image and extend to the lower right corner (width-1, height-1). Thus for an 800x600 resolution image, the valid values for the ROI points range from (0.0) to (799.599). It should also be noted that the 4 points cannot be collinear.



Figure 10- How to calculate avgPlateAngle and avgPlateSlant values

<b>struct JidoshaLightRecognitionInfo</b>	
<b>Description</b>	The purpose of this structure is to bring information related to the processing time of <i>JidoshaLight</i> , facilitating the performance diagnosis. All times are given in milliseconds.
<b>Members</b>	<i>double totalTime</i> : total image processing time (sum of the other times). <i>double localizationTime</i> : time spent on the plate localization step in the image. <i>double segmentationTime</i> : time spent in the step of extracting the characters from the plate. <i>double classificationTime</i> : time spent in the step of classifying the characters from the plate. <i>double loadDecodeTime</i> : time spent reading and decoding the image file. <i>int LibVersion[3]</i> : version of the library that processed the image. <i>char LibSHA1[41]</i> : identifier of the library that processed the image.

<b>struct JidoshaLightRecognition</b>	
<b>Description</b>	The purpose of this structure is to store the result of the plate recognition, including: the characters of the plate, the reliability of each character, and the coordinates of the plate in the image.
<b>Members</b>	<i>int frameId</i> : field reserved for future use, its value is always zeroed. <i>char plate[8]</i> : 7-character plate ending with 0, or empty string if the plate was not found. <i>float probabilities[7]</i> : values from 0.0 to 1.0 indicating the reliability, in the form of probability, of the recognition of each character.

	<p><i>int xText</i> and <i>int yText</i>: coordinates of the upper left corner of the plate, if found.</p> <p><i>int widthText</i>: width of the plate rectangle.</p> <p><i>int heightText</i>: height of the plate rectangle.</p> <p><i>int xChar[7]</i> and <i>int yChar[7]</i>: coordinate the upper left corner of each of the recognized characters.</p> <p><i>int widthChar[7]</i>: width of the rectangle of each of the recognized characters.</p> <p><i>int heightChar[7]</i>: height of the rectangle of each of the recognized characters.</p> <p><i>int textColor</i>: plate text color, 0 - dark, 1 - light.</p> <p><i>int isMotorcycle</i>: indicates if plate is motorcycle, 0 - non-motorcycle, 1 - motorcycle.</p> <p><i>int countryCode</i>: indicates the country code (in ISO 3166-1 standard) of the recognized plate. Possible values for this field are defined in the <a href="#">enum JidoshaLightCountryCode</a>.</p> <p><i>JidoshaLightRecognitionInfo</i>: structure containing processing time information.</p>
--	--

### struct JidoshaLightLicenseInfo

<b>Description</b>	Structure used to store information about the license used by the JidoshaLight library.
<b>Members</b>	<p><i>uint64_t serial</i>: serial license number</p> <p><i>char customer[64]</i>: name of the customer who purchased the license</p> <p><i>int maxThreads</i>: maximum number of processing threads enabled</p> <p><i>int maxConnections</i>: maximum number of parallel connections enabled</p> <p><i>int state</i>: license status (see <a href="#">Function return codes</a>)</p> <p><i>int ttl</i>: time-to-live in hours for RTC type licenses. This field has the value <b>-1</b> if the license is not expired</p>

### struct JidoshaLightRecognitionList

<b>Description</b>	<p>Opaque type used by the library to return a list of objects of type <a href="#">JidoshaLightRecognition</a>. Functions that manipulate the list insert new elements at the end of the list. To clear a list, simply destroy and create a new one.</p> <p>To avoid memory leaks, the user must always destroy the lists that are no longer used. This type is not thread safe.</p>
<b>Members</b>	<b>None</b>
<b>Related methods</b>	<p><a href="#">jidoshalight ANPR createList</a></p> <p><a href="#">jidoshalight ANPR duplicateList</a></p> <p><a href="#">jidoshalight ANPR destroyList</a></p> <p><a href="#">jidoshalight ANPR getListSize</a></p> <p><a href="#">jidoshalight ANPR getListElement</a></p>

### struct JidoshaLightImage

<b>Description</b>	<p>Opaque type used by the library to load an image to be processed. Once created, a <i>JidoshaLightImage</i> object can be used to upload numerous images, even if they are of different formats. However, subsequent calls cause the previously uploaded content to be overwritten.</p> <p>The image decode process can be postponed to processing time if <i>LazyDecode</i> mode is enabled. In this situation, the <i>load</i> functions only store the contents of the raw buffer of the image without performing any processing. This behavior is useful for client-server applications since the computational cost of the decode is delegated to the server.</p> <p>To avoid memory leaks, the user must always destroy images that are no longer in use. This type is not thread safe.</p>
<b>Members</b>	<b>None</b>
<b>Related methods</b>	<p><a href="#">jidoshalight ANPR createImage</a></p> <p><a href="#">jidoshalight ANPR duplicateImage</a></p> <p><a href="#">jidoshalight ANPR destroyImage</a></p> <p><a href="#">jidoshalight ANPR setImageLazyDecode</a></p>

	<a href="#">jidoshaLight_ANPR_loadImageFromFile</a> <a href="#">jidoshaLight_ANPR_loadImageFromMemory</a> <a href="#">jidoshaLight_ANPR_loadImageFromRawImgFmt</a>
--	--

## Methods

<b>jidoshaLight_ANPR_createList</b>	
<b>Function Prototype</b>	<code>JidoshaLightRecognitionList* jidoshaLight_ANPR_createList();</code>
<b>Description</b>	Function used to create an empty <a href="#">JidoshaLightRecognitionList</a>
<b>Parameters</b>	None
<b>Return</b>	A valid pointer for the <i>JidoshaLightRecognitionList</i> or <i>NULL</i> type in case of failure.

<b>jidoshaLight_ANPR_duplicateList</b>	
<b>Function Prototype</b>	<code>JidoshaLightRecognitionList* jidoshaLight_ANPR_duplicateList(JidoshaLightRecognitionList* list);</code>
<b>Description</b>	Function used to duplicate a <a href="#">JidoshaLightRecognitionList</a>
<b>Parameters</b>	<i>JidoshaLightRecognitionList* list</i> : pointer to a <i>JidoshaLightRecognitionList</i> object
<b>Return</b>	A valid pointer for the <i>JidoshaLightRecognitionList</i> or <i>NULL</i> type in case of failure.

<b>jidoshaLight_ANPR_destroyList</b>	
<b>Function Prototype</b>	<code>int jidoshaLight_ANPR_destroyList(JidoshaLightRecognitionList* list);</code>
<b>Description</b>	Function used to destroy objects created by <i>jidoshaLight_ANPR_createList</i> and <i>jidoshaLight_ANPR_duplicateList</i> functions
<b>Parameters</b>	<i>JidoshaLightRecognitionList* list</i> : pointer to a <i>JidoshaLightRecognitionList</i> object
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jidoshaLight_ANPR_getListSize</b>	
<b>Function Prototype</b>	<code>int jidoshaLight_ANPR_getListSize(JidoshaLightRecognitionList* list);</code>
<b>Description</b>	Function used to read the amount of elements stored within the list
<b>Parameters</b>	<i>JidoshaLightRecognitionList* list</i> : pointer to a <i>JidoshaLightRecognitionList</i> object
<b>Return</b>	Number of elements present in the list (value greater than or equal to zero) or -1 in case of error (* invalid <i>list</i> ).

<b>jidoshaLight_ANPR_getListElement</b>	
<b>Function Prototype</b>	<code>const JidoshaLightRecognition* jidoshaLight_ANPR_getListElement(JidoshaLightRecognitionList* list, int pos);</code>
<b>Description</b>	Function used to retrieve a pointer to the element present in the <i>pos</i> position of the list. The content of the returned pointer cannot be modified by the user ( <i>const</i> ).
<b>Parameters</b>	<i>JidoshaLightRecognitionList* list</i> : pointer to a <i>JidoshaLightRecognitionList</i> object <i>int pos</i> : position of the element to be retrieved in the range '[0,ListSize)
<b>Return</b>	Valid pointer to an immutable object of type <a href="#">JidoshaLightRecognition</a> or <i>NULL</i> in case of error ( <i>list</i> invalid or <i>pos</i> out of range).

<b>jidoshaLight_ANPR_createImage</b>	
<b>Function Prototype</b>	<code>JidoshaLightImage* jidoshaLight_ANPR_createImage();</code>
<b>Description</b>	Function used to create a <a href="#">JidoshaLightImage</a>
<b>Parameters</b>	None
<b>Return</b>	Valid pointer for type <i>JidoshaLightImage</i> or <i>NULL</i> in case of failure.

<b>jidoshaLight_ANPR_duplicateImage</b>	
<b>Function Prototype</b>	<code>JidoshaLightImage* jidoshaLight_ANPR_duplicateImage(JidoshaLightImage* img);</code>
<b>Description</b>	Function used to create a <a href="#">JidoshaLightImage</a> . The duplicate image inherits the state of the original image.
<b>Parameters</b>	<i>JidoshaLightImage* img</i> : pointer to a <i>JidoshaLightImage</i> object
<b>Return</b>	Valid pointer for type <i>JidoshaLightImage</i> or <i>NULL</i> in case of failure.

<b>jidoshaLight_ANPR_destroyImage</b>	
<b>Function Prototype</b>	<code>int jidoshaLight_ANPR_destroyImage(JidoshaLightImage* img);</code>
<b>Description</b>	Function used to destroy objects created by <a href="#">jidoshaLight_ANPR_createImage</a> and <a href="#">jidoshaLight_ANPR_duplicateImage</a> functions
<b>Parameters</b>	<i>JidoshaLightImage* img</i> : pointer to a <i>JidoshaLightImage</i> object
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jidoshaLight_ANPR_setImageLazyDecode</b>	
<b>Function Prototype</b>	<code>int jidoshaLight_ANPR_setImageLazyDecode(JidoshaLightImage* img, int enable);</code>
<b>Description</b>	Function used to enable <i>LazyDecode</i> mode (see description in <a href="#">JidoshaLightImage</a> ). The change has immediate effect and invalidates any previously uploaded image.
<b>Parameters</b>	<i>JidoshaLightImage* img</i> : pointer to a <i>JidoshaLightImage</i> object int enable: 0 disabled (default), 1 enabled
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jidoshaLight_ANPR_loadImageFromFile</b>	
<b>Function Prototype</b>	<code>int jidoshaLight_ANPR_loadImageFromFile ( JidoshaLightImage* img, const char* filename );</code>
<b>Description</b>	Function used to load a <i>JidoshaLightImage</i> from a file. Supported file formats: JPEG, BMP, PNG and TIFF.
<b>Parameters</b>	<i>JidoshaLightImage* img</i> : pointer to a <i>JidoshaLightImage</i> object <i>const char* filename</i> : absolute path to the file to be uploaded
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jidosaLight_ANPR_loadImageFromMemory</b>	
<b>Function Prototype</b>	<pre>int jidoshaLight_ANPR_loadImageFromMemory (     JidoshaLightImage* img,     const uint8_t* buffer,     int bufferSize );</pre>
<b>Description</b>	Function used to load a <i>JidoshaLightImage</i> from a file already loaded in memory. Supported file formats: JPEG, BMP, PNG, and TIFF.
<b>Parameters</b>	<i>JidoshaLightImage* img</i> : pointer to a <i>JidoshaLightImage</i> object <i>const uint8_t* buffer</i> : pointer to the buffer containing the uploaded file <i>int bufferSize</i> : size in buffer bytes
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jidosaLight_ANPR_loadImageFromRawImgFmt</b>	
<b>Function Prototype</b>	<pre>int jidoshaLight_ANPR_loadImageFromRawImgFmt (     JidoshaLightImage* img,     const uint8_t* buffer,     int width,     int height,     int stride,     JidoshaLightRawImgFmt fmt );</pre>
<b>Description</b>	Function used to load a <i>JidoshaLightImage</i> from a buffer containing an image in RAW format. See supported formats in <a href="#">enum JidoshaLightRawImgFmt</a>
<b>Parameters</b>	<i>JidoshaLightImage* img</i> : pointer to a <i>JidoshaLightImage</i> object <i>const uint8_t* buffer</i> : pointer to the buffer containing the image in RAW format <i>int width</i> : width in pixels of the image <i>int height</i> : height in pixels of the image <i>int stride</i> : size in bytes of an image line <i>JidoshaLightRawImgFmt fmt</i> : image format
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jidosaLight_ANPR_fromFile</b>	
<b>Function Prototype</b>	<pre>int jidoshaLight_ANPR_fromFile (     const char* filename,     JidoshaLightConfig* config,     JidoshaLightRecognition* rec );</pre>
<b>Description</b>	<p>Recognizes a plate from an image file whose path is provided in <i>const char* filename</i>. Uses the settings defined in <i>JidoshaLightConfig* config</i> and returns the recognition result in the <i>JidoshaLightRecognition* rec</i> struct. If a plate cannot be found in the image, the <i>rec-&gt;plate</i> field is returned empty.</p> <p>If an error occurs during processing, the <i>JidoshaLightRecognition* rec</i> structure will be returned empty and a different value of <i>JIDOSHA_LIGHT_SUCCESS</i> will be returned by the function. Possible return values are defined in the enum <i>JidoshaLightReturnCode</i>. See supported formats in <a href="#">jidosaLight_ANPR_loadImageFromFile</a>.</p>
<b>Parameters</b>	<i>const char* filename</i> : path to the image file. <i>JidoshaLightConfig* config</i> : pointer to the 'struct JidoshaLightConfig' with the configuration for the library. A 'NULL' pointer in this parameter causes the use of the library's default settings.

	<i>JidoshaLightRecognition* rec</i> : pointer to the 'struct JidoshaLightRecognition' where the reading result will be stored.
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

### jidoshalight\_ANPR\_fromMemory

<b>Function Prototype</b>	<pre>int jidoshalight_ANPR_fromMemory (     const unsigned char* buffer,     int bufferSize,     JidoshaLightConfig* config,     JidoshaLightRecognition* rec );</pre>
<b>Description</b>	<p>Recognizes a license plate from a <i>buffer</i> containing an image file previously loaded in memory. Uses the settings defined in <i>JidoshaLightConfig* config</i> and returns the recognition result in the <i>JidoshaLightRecognition* rec</i> struct. If a plate cannot be found in the image, the <i>rec-&gt;plate</i> field is returned empty.</p> <p>If an error occurs during processing, the <i>JidoshaLightRecognition* rec</i> structure will be returned empty and a different value of <i>JIDOSHA_LIGHT_SUCCESS</i> will be returned by the function. Possible return values are defined in the <i>enum JidoshaLightReturnCode</i>. See supported formats in <a href="#">jidoshalight_ANPR_loadImageFromMemory</a>.</p>
<b>Parameters</b>	<p><i>const unsigned char* buffer</i>: array of bytes containing the image.</p> <p><i>int bufferSize</i>: size of the byte array.</p> <p><i>JidoshaLightConfig* config</i>: pointer to the 'struct JidoshaLightConfig' with the configuration for the library. A 'NULL' pointer in this parameter causes the use of the library's default settings.</p> <p><i>JidoshaLightRecognition* rec</i>: pointer to the 'struct JidoshaLightRecognition' where the reading result will be stored.</p>
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

### jidoshalight\_ANPR\_fromLuma

<b>Function Prototype</b>	<pre>int jidoshalight_ANPR_fromLuma (     unsigned char* luma,     int width,     int height,     JidoshaLightConfig* config,     JidoshaLightRecognition* rec );</pre>
<b>Description</b>	<p>Recognizes a license plate from a buffer containing an 8-bit raw grayscale image. Uses the settings defined in <i>JidoshaLightConfig* config</i> and returns the recognition result in the <i>JidoshaLightRecognition* rec</i> struct. If a plate cannot be found in the image, the <i>rec-&gt;plate</i> field is returned empty.</p> <p>If an error occurs during processing, the <i>JidoshaLightRecognition* rec</i> structure will be returned empty and a different value of <i>JIDOSHA_LIGHT_SUCCESS</i> will be returned by the function. Possible return values are defined in the <i>enum JidoshaLightReturnCode</i>.</p>
<b>Parameters</b>	<p><i>unsigned char* luma</i>: array of bytes containing the image in 8-bit RAW grayscale format.</p> <p><i>int width</i>: image width.</p> <p><i>int height</i>: image height.</p> <p><i>JidoshaLightConfig* config</i>: pointer to the 'struct JidoshaLightConfig' with the configuration for the library. A 'NULL' pointer in this parameter causes the use of the library's default settings.</p> <p><i>JidoshaLightRecognition* rec</i>: pointer to the 'struct JidoshaLightRecognition' where the reading result will be stored.</p>
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jidoshalight_ANPR_fromRawImgFmt</b>	
<b>Function Prototype</b>	<pre>int jidoshalight_ANPR_fromRawImgFmt (     const unsigned char* buffer,     int width,     int height,     int stride,     JidoshaLightRawImgFmt fmt,     JidoshaLightConfig* config,     JidoshaLightRecognition* rec );</pre>
<b>Description</b>	<p>Reconhece uma placa a partir de um <i>buffer</i> contendo uma imagem em algum dos formatos RAW definidos na enum <i>JidoshaLightRawImgFmt</i>.</p> <p>Uses the settings defined in <i>JidoshaLightConfig* config</i> and returns the recognition result in the <i>JidoshaLightRecognition* rec</i> struct. If a plate cannot be found in the image, the <i>rec-&gt;plate</i> field is returned empty.</p> <p>If an error occurs during processing, the <i>JidoshaLightRecognition* rec</i> structure will be returned empty and a different value of <i>JIDOSHA_LIGHT_SUCCESS</i> will be returned by the function. Possible return values are defined in the enum <i>JidoshaLightReturnCode</i>.</p> <p>See supported formats in <a href="#">jidoshalight_ANPR_loadImageFromRawImgFmt</a>.</p>
<b>Parameters</b>	<p><i>const unsigned char* buffer</i>: array of bytes containing the image in RAW format.</p> <p><i>int width</i>: image width.</p> <p><i>int height</i>: image height.</p> <p><i>int stride</i>: size in bytes of an image line</p> <p><i>JidoshaLightRawImgFmt fmt</i>: image format</p> <p><i>JidoshaLightConfig* config</i>: pointer to the '<i>struct JidoshaLightConfig</i>' with the configuration for the library. A '<i>NULL</i>' pointer in this parameter causes the use of the library's default settings.</p> <p><i>JidoshaLightRecognition* rec</i>: pointer to the '<i>struct JidoshaLightRecognition</i>' where the reading result will be stored.</p>
<b>Return</b>	<p><i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a>)</p>

<b>jidoshalight_ANPR_fromImage</b>	
<b>Function Prototype</b>	<pre>int jidoshalight_ANPR_fromImage(     JidoshaLightImage* img,     JidoshaLightConfig* config,     JidoshaLightRecognition* rec );</pre>
<b>Description</b>	<p>Recognizes a plate from a previously loaded <a href="#">JidoshaLightImage</a>.</p> <p>Uses the settings defined in <i>JidoshaLightConfig* config</i> and returns the recognition result in the <i>JidoshaLightRecognition* rec</i> struct. If a plate cannot be found in the image, the <i>rec-&gt;plate</i> field is returned empty.</p> <p>If an error occurs during processing, the <i>JidoshaLightRecognition* rec</i> structure will be returned empty and a different value of <i>JIDOSHA_LIGHT_SUCCESS</i> will be returned by the function. Possible return values are defined in the enum <i>JidoshaLightReturnCode</i>.</p>
<b>Parameters</b>	<p><i>JidoshaLightImage* img</i>: pointer to a valid <a href="#">JidoshaLightImage</a></p> <p><i>JidoshaLightConfig* config</i>: pointer to the '<i>struct JidoshaLightConfig</i>' with the configuration for the library. A '<i>NULL</i>' pointer in this parameter causes the use of the library's default settings.</p> <p><i>JidoshaLightRecognition* rec</i>: pointer to the '<i>struct JidoshaLightRecognition</i>' where the reading result will be stored.</p>
<b>Return</b>	<p><i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a>)</p>



<b>jidoshalight_ANPR_multi_fromImage</b>	
<b>Function Prototype</b>	<pre>int jidoshaLight_ANPR_multi_fromImage (     JidoshaLightImage* img,     JidoshaLightConfig* config,     int maxPlates,     JidoshaLightRecognitionList* list );</pre>
<b>Description</b>	<p>Recognizes multiple plates from a previously loaded <a href="#">JidoshaLightImage</a>. Uses the settings defined in <i>JidoshaLightConfig* config</i> adds 'maxPlates' recognitions to the end of the '<i>JidoshaLightRecognitionList* list</i>'. If the number of plates found is less than the specified value, empty '<i>JidoshaLightRecognition</i>' elements will be added to the list until filling in 'maxPlates'.</p> <p>If an empty <i>JidoshaLightRecognition</i> element error occurs, it will be added to the list and a return code other than <i>JIDOSHA_LIGHT_SUCCESS</i> will be returned by the function (see <a href="#">enum JidoshaLightReturnCode</a>).</p>
<b>Parameters</b>	<p><i>JidoshaLightImage* img</i>: pointer to a valid <a href="#">JidoshaLightImage</a></p> <p><i>JidoshaLightConfig* config</i>: pointer to the '<i>struct JidoshaLightConfig</i>' with the configuration for the library. A 'NULL' pointer in this parameter causes the use of the library's default settings.</p> <p><i>int maxPlates</i>: maximum number of plates to be recognized (1 or more)</p> <p><i>JidoshaLightRecognitionList* list</i>: pointer to a <a href="#">JidoshaLightRecognitionList</a> object where new <i>JidoshaLightRecognition</i> <a href="#">maxPlates</a> will be added.</p>
<b>Return</b>	<p><i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a>)</p>

<b>jidoshalight_getVersion</b>	
<b>Function Prototype</b>	<pre>int jidoshaLight_getVersion(int* major, int* minor, int* release);</pre>
<b>Description</b>	Used to check the version of the library, in major.minor.release format.
<b>Parameters</b>	<i>int major, minor, release</i> : pointers to int variables where the numbers that make up the version will be written.
<b>Return</b>	Always returns <i>JIDOSHA_LIGHT_SUCCESS</i> .

<b>jidoshalight_getBuildSHA1</b>	
<b>Function Prototype</b>	<pre>const char* jidoshaLight_getBuildSHA1();</pre>
<b>Description</b>	Used to check the SHA1 of the library build.
<b>Parameters</b>	None
<b>Return</b>	Returns a pointer to the beginning of a string ending in \0 containing the value of the build SHA1.

<b>jidoshalight_getBuildFlags</b>	
<b>Function Prototype</b>	<pre>const char* jidoshaLight_getBuildFlags();</pre>
<b>Description</b>	Used to check library build options.
<b>Parameters</b>	None
<b>Return</b>	Returns a pointer to the beginning of a string ending in \0 containing the build options.

<b>jidosaLight_isRemoteApi</b>	
<b>Function Prototype</b>	<code>JL_API int jidoshaLight_isRemoteApi();</code>
<b>Description</b>	Checks whether the application library implements local or remote processing.
<b>Parameters</b>	None
<b>Return</b>	Returns 0 if the API is implemented with local access or different from this value if the processing is remote.

<b>jidosaLight_getLicenseInfo</b>	
<b>Function Prototype</b>	<code>int jidoshaLight_getLicenseInfo(JidoshaLightLicenseInfo* info)</code>
<b>Description</b>	Function used to read the license information used by the JidoshaLight library.
<b>Parameters</b>	<i>JidoshaLightLicenseInfo* info</i> : pointer to a <a href="#">JidoshaLightLicenseInfo struct</a>
<b>Return</b>	Returns JIDOSHA_LIGHT_SUCCESS on success.

<b>jidosaLight_getReturnCodeString</b>	
<b>Function Prototype</b>	<code>const char* jidoshaLight_getReturnCodeString(int rc)</code>
<b>Description</b>	Function used to convert a library return code into a C string.
<b>Parameters</b>	<i>int rc</i> : some return code defined in <a href="#">enum JidoshaLightReturnCode</a> and <a href="#">enum JidoshaLightReturnCodeNetwork</a>
<b>Return</b>	String representative of the error code.

### Function return codes

The function return codes are related to the recognition process ('*enum JidoshaLightReturnCode*') or the remote communication process ('*enum JidoshaLightReturnCodeNetwork*'). Codes:

- JIDOSHA\_LIGHT\_ERROR\_FILE\_not\_FOUND: returned by the [jidosaLight ANPR fromFile](#) and [jidosaLight ANPR loadImageFromFile](#) functions when the specified file path does not exist.
- JIDOSHA\_LIGHT\_ERROR\_INVALID\_IMAGE: returned by image processing and loading functions. Occurs when the past image is corrupted.
- JIDOSHA\_LIGHT\_ERROR\_invalid\_IMAGE\_TYPE: Returned by [jidosaLight ANPR fromFile](#), [jidosaLight ANPR fromMemory](#), and [JidoshaLightImage](#) load-related functions. Occurs when attempting to process an image of unsupported format.
- JIDOSHA\_LIGHT\_ERROR\_invalid\_IMAGE\_SIZE: Returned by [jidosaLight ANPR fromFile](#), [jidosaLight ANPR fromMemory](#), and [JidoshaLightImage](#) load-related functions. Occurs when attempting to process an image whose size exceeds the maximum limits supported by the library (arm Zynq: 1280x960px, Others: 2500x2500px).
- JIDOSHA\_LIGHT\_ERROR\_invalid\_PROPERTY: returned by all functions that have arguments. Occurs when the argument is invalid. In the case of functions that receive pointers, this code is returned when the argument is **NULL** (except in cases where **NULL** is a valid value for the argument).
- JIDOSHA\_LIGHT\_ERROR\_country\_not\_SUPPORTED: returned by **ANPR** functions when the country code provided in the configuration structure is not supported by the library.
- JIDOSHA\_LIGHT\_ERROR\_API\_CALL\_NOT\_SUPPORTED: Returned when an API function is not available for a given platform.

- JIDOSHA\_LIGHT\_ERROR\_INVALID\_ROI: returned when an invalid region of interest is provided. See the description of [JidoshaLightConfig struct](#) for more information.
- JIDOSHA\_LIGHT\_ERROR\_INVALID\_HANDLE: returned when the *handle* passed to the function was not initialized correctly.
- JIDOSHA\_LIGHT\_ERROR\_API\_CALL\_HAS\_NO\_EFFECT: returned when an API function had no effect when executed. It can occur when there is precedence between calls.
- JIDOSHA\_LIGHT\_ERROR\_LICENSE\_INVALID: returned by **ANPR** functions when *hardkey* is not present or has problems. Contact Pumatronix Equipamentos Eletrônicos for more information.
- JIDOSHA\_LIGHT\_ERROR\_LICENSE\_EXPIRED: Returned by **ANPR** functions when a demo-type *hardkey has expired*. Contact Pumatronix Equipamentos Eletrônicos for more information.
- JIDOSHA\_LIGHT\_ERROR\_LICENSE\_MAX\_THREADS\_EXCEEDED: returned by **ANPR** functions when the maximum number of competing threads exceeds that allowed by the license.
- JIDOSHA\_LIGHT\_ERROR\_LICENSE\_UNTRUSTED\_RTC: returned by **ANPR** functions when a license with use-by date does not have a reliable time/date reference available.
- JIDOSHA\_LIGHT\_ERROR\_OTHER: returned when an unexpected error occurs. Contact Pumatronix Equipamentos Eletrônicos for support.
- JIDOSHA\_LIGHT\_ERROR\_SERVER\_CONNECT\_FAILED: Returned when a remote API call cannot connect to the server.
- JIDOSHA\_LIGHT\_ERROR\_SERVER\_DISCONNECTED: Returned when a remote session with the server was unexpectedly closed.
- JIDOSHA\_LIGHT\_ERROR\_SERVER\_QUEUE\_TIMEOUT: Returned when a request was dropped on the server by timeout.
- JIDOSHA\_LIGHT\_ERROR\_SERVER\_QUEUE\_FULL: returned when a request was dropped on the server due to lack of queue space.
- JIDOSHA\_LIGHT\_ERROR\_SOCKET\_IO\_ERROR: Returned when a network IO error occurred in a remote session with the server.
- JIDOSHA\_LIGHT\_ERROR\_SOCKET\_WRITE\_FAILED: Returned when error occurs in sending messages between client and remote server.
- JIDOSHA\_LIGHT\_ERROR\_SOCKET\_READ\_TIMEOUT: returned when error occurs in receiving messages between client and remote server.
- JIDOSHA\_LIGHT\_ERROR\_SOCKET\_INVALID\_RESPONSE: returned when an invalid message was received.
- JIDOSHA\_LIGHT\_ERROR\_HANDLE\_QUEUE\_FULL: returned when the pending request queue reached the maximum for a given asynchronous handle.
- JIDOSHA\_LIGHT\_ERROR\_SERVER\_CONN\_LIMIT\_REACHED: Returned when trying to connect to a server with the maximum number of open sessions.
- JIDOSHA\_LIGHT\_ERROR\_SERVER\_VERSION\_NOT\_SUPPORTED: returned when the server version is not compatible with the client library version.
- JIDOSHA\_LIGHT\_ERROR\_SERVER\_NOT\_READY: returned when the server is starting more is not yet ready to process images. The customer should wait and try to reconnect.

## JidoshaLight C/C++ API (Synchronous Remote)

The Synchronous Remote API extends the Local API, allowing you to configure a remote server to process images remotely rather than locally. It should be used in conjunction with the *libjidoshaLightRemote.so* library.

The *jidoshaLight\_ANPR* calls defined in the Local API remain valid, but processing becomes remote when the application is linked with *libjidoshaLightRemote.so*.

```
//=====
```

```
// FUNCTIONS
//=====
JL_API int jidoshaLight_setRemoteSyncServerIp(
    const char* ip,
    unsigned int port
);
```

## Methods

<b>jidoshaLight_setRemoteSyncServerIp</b>	
<b>Function Prototype</b>	<pre>JL_API int jidoshaLight_setRemoteSyncServerIp(     const char* ip,     unsigned int port );</pre>
<b>Description</b>	Configures globally the IP address and TCP port used for connection to a license plate recognition server. The session is established and closed at each recognition call.
<b>Parameters</b>	<i>const char* ip</i> : string containing the IP address of the server. <i>int port</i> : integer containing the TCP port of the server.
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

## JidoshaLight C/C++ API (Asynchronous Remote)

The Asynchronous Remote API extends the Local API, allowing you to configure a remote server that will process images remotely rather than processing them locally. It should be used in conjunction with the *libjidoshaLightRemote.so* library.

```
//=====
// TYPES
//=====
typedef struct JidoshaLightHandle JidoshaLightHandle;

/* Recognition result callback function pointer */
typedef void (*JCallback) (
    JidoshaLightRecognition rec,
    int rc,
    uint8_t* buffer,
    unsigned int bufferSize,
    void* arg
);

typedef struct JidoshaLightClientConfig
{
    int            queueSize;
    const char*   ip;
    int            port;
    JCallback     callback;
    void*         arg;
} JidoshaLightClientConfig;

typedef struct JidoshaLightServerInfo
{
    JidoshaLightLicenseInfo license;
    int major;
    int minor;
};
```

```
    int release;
} JidoshaLightServerInfo;

//=====
// FUNCTION CALLS
//=====
// HANDLE
//=====
JL_API JidoshaLightHandle* jl_async_create_handle(JidoshaLightClientConfig* clientConfig);
JL_API int jl_async_destroy_handle(JidoshaLightHandle* handle);
JL_API int jl_async_connect(JidoshaLightHandle* handle);
JL_API int jl_async_connect_info(JidoshaLightHandle* handle, JidoshaLightServerInfo* info);
JL_API int jl_async_get_localqueue_size(JidoshaLightHandle* handle);

//=====
// PROCESSING
//=====
JL_API int jl_async_ANPR_fromFile (
    JidoshaLightHandle* handle,
    const char* filename,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromMemory (
    JidoshaLightHandle* handle,
    const unsigned char* buffer,
    unsigned int bufferSize,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromLuma (
    JidoshaLightHandle* handle,
    unsigned char* luma,
    int width,
    int height,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromRawImgFmt (
    JidoshaLightHandle* handle,
    const unsigned char* buffer,
    int width,
    int height,
    int stride,
    JidoshaLightRawImgFmt fmt,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_fromImage (
    JidoshaLightHandle* handle,
    JidoshaLightImage* img,
    JidoshaLightConfig* config
);

JL_API int jl_async_ANPR_multi_fromImage (
    JidoshaLightHandle* handle,
```

```
JidoshaLightImage* img,
JidoshaLightConfig* config,
int maxPlates
);
```

## Types

### struct JidoshaLightHandle

<b>Description</b>	The purpose of this structure is to store the client object of a plate recognition server.
<b>Members</b>	None

### typedef void JCallback

<b>Description</b>	The purpose of this type is to define the format of the user callback for receiving events from the server.
<b>Members</b>	<i>struct JidoshaLightRecognition rec</i> : struct where the recognition result will be stored <i>int rc</i> : requisition return code (see <a href="#">Function return codes</a> ) <i>uint8_t* buffer</i> : pointer to the image where the recognition was performed (this pointer is only valid during the execution of the callback) <i>unsigned int bufferSize</i> : image size <i>void* arg</i> : pointer to opaque structure provided by the user in the creation of the handle

### struct JidoshaLightClientConfig

<b>Description</b>	The purpose of this structure is to define the connection parameters between the client and the server.
<b>Members</b>	<i>int queueSize</i> : maximum size of pending requests for this <i>handle</i> . <i>const char* ip</i> : string containing the IP address of the server. <i>int port</i> : integer containing the TCP port of the server. <i>JCallback callback</i> : function designated for processing the results generated by the server. <i>void* arg</i> : opaque pointer to user data structure used for handling server events. This pointer is passed as user callback parameter.

### struct JidoshaLightServerInfo

<b>Description</b>	Struct used to store license and version information of a JidoshaLight server.
<b>Members</b>	<i>JidoshaLightLicenseInfo license</i> : structure containing server license information - see struct <a href="#">JidoshaLightLicenseInfo</a> <i>int major</i> : major value of the library version used by the server <i>int minor</i> : minor value of the library version used by the server <i>int release</i> : value of the release of the version of the library used by the server

## Methods

### jl\_async\_create\_handle

<b>Function Prototype</b>	<pre>JidoshaLightHandle* jl_async_create_handle(     JidoshaLightClientConfig* config );</pre>
<b>Description</b>	Creates the handle of an assicron client for connection to a plate recognition server.
<b>Parameters</b>	<i>JidoshaLightClientConfig* config</i> : configuration for this <i>handle</i> .
<b>Return</b>	Returns a pointer to the <i>handle</i> of type <i>JidoshaLightHandle</i> or <i>NULL</i> in case of error.

<b>jl_async_destroy_handle</b>	
<b>Function Prototype</b>	<pre>int jl_async_destroy_handle(     JidoshaLightHandle* handle );</pre>
<b>Description</b>	Deallocates the <i>handle</i> of an assicron client, closing the connection to the plate recognition server.
<b>Parameters</b>	<i>JidoshaLightHandle* handle</i> : pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> .
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jl_async_connect</b>	
<b>Function Prototype</b>	<pre>int jl_async_connect(     JidoshaLightHandle* handle );</pre>
<b>Description</b>	Establishes the session with the plate recognition server for a given <i>handle</i> . This function blocks until the connection is established or timeout occurs.
<b>Parameters</b>	<i>JidoshaLightHandle* handle</i> : pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> .
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jl_async_connect_info</b>	
<b>Function Prototype</b>	<pre>int jl_async_connect_info(     JidoshaLightHandle* handle,     JidoshaLightServerInfo* info );</pre>
<b>Description</b>	It has the same functionality as the <a href="#">jl_async_connect</a> function but receives an additional parameter to receive information about the license and the server version.
<b>Parameters</b>	<i>JidoshaLightHandle* handle</i> : pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> . <i>JidoshaLightServerInfo* info</i> : Pointer to a <a href="#">JidoshaLightServerInfo struct</a>
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

<b>jl_async_get_localqueue_size</b>	
<b>Function Prototype</b>	<pre>int jl_async_get_localqueue_size(     JidoshaLightHandle* handle );</pre>
<b>Description</b>	Returns the queue size of pending requests on the client side for a given <i>handle</i> .
<b>Parameters</b>	<i>JidoshaLightHandle* handle</i> : Pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a>
<b>Return</b>	Returns the number of pending requisitions in the local (client) queue.

<b>jl_async_ANPR_fromFile</b>	
<b>Function Prototype</b>	<pre>int jl_async_ANPR_fromFile(     JidoshaLightHandle* handle,     const char* filename,     JidoshaLightConfig* config );</pre>
<b>Description</b>	See description of method <a href="#">jidoshaLight ANPR fromFile</a>

<b>Parameters</b>	<i>JidoshaLightHandle*</i> handle: Pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> <i>const char*</i> filename: See description of method <a href="#">jidoshaLight ANPR fromFile</a> <i>JidoshaLightConfig*</i> config: See description of method <a href="#">jidoshaLight ANPR fromFile</a>
<b>Return</b>	See description of method <a href="#">jidoshaLight ANPR fromFile</a>

<b>jl_async_ANPR_fromMemory</b>	
<b>Function Prototype</b>	<pre>int jl_async_ANPR_fromMemory(     JidoshaLightHandle* handle,     const unsigned char* buffer,     unsigned int bufferSize,     JidoshaLightConfig* config );</pre>
<b>Description</b>	See description of method <a href="#">jidoshaLight ANPR fromMemory</a>
<b>Parameters</b>	<i>JidoshaLightHandle*</i> handle: Pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> <i>const unsigned char*</i> buffer: See description of method <a href="#">jidoshaLight ANPR fromMemory</a> <i>unsigned int</i> bufferSize: See description of method <a href="#">jidoshaLight ANPR fromMemory</a> <i>JidoshaLightConfig*</i> config: See description of method <a href="#">jidoshaLight ANPR fromMemory</a>
<b>Return</b>	See description of method <a href="#">jidoshaLight ANPR fromMemory</a>

<b>jl_async_ANPR_fromLuma</b>	
<b>Function Prototype</b>	<pre>int jl_async_ANPR_fromLuma(     JidoshaLightHandle* handle,     unsigned char* luma,     int width,     int height,     JidoshaLightConfig* config );</pre>
<b>Description</b>	See description of method <a href="#">jidoshaLight ANPR fromLuma</a>
<b>Parameters</b>	<i>JidoshaLightHandle*</i> handle: Pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> <i>unsigned char*</i> luma: See description of method <a href="#">jidoshaLight ANPR fromLuma</a> . <i>int</i> width: See description of method <a href="#">jidoshaLight ANPR fromLuma</a> . <i>int</i> height: See description of method <a href="#">jidoshaLight ANPR fromLuma</a> . <i>JidoshaLightConfig</i> config: See description of method <a href="#">jidoshaLight ANPR fromLuma</a> .
<b>Return</b>	See description of method <a href="#">jidoshaLight ANPR fromLuma</a>

<b>jl_async_ANPR_fromRawImgFmt</b>	
<b>Function Prototype</b>	<pre>int jl_async_ANPR_fromRawImgFmt (     JidoshaLightHandle* handle,     const unsigned char* buffer,     int width,     int height,     int stride,     JidoshaLightRawImgFmt fmt,     JidoshaLightConfig* config,     JidoshaLightRecognition* rec );</pre>
<b>Description</b>	See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a>



<b>Parameters</b>	<i>JidoshaLightHandle* handle</i> : Pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> <i>const unsigned char* buffer</i> : See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a> . <i>int width</i> : See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a> . <i>int height</i> : See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a> . <i>int stride</i> : See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a> . <i>JidoshaLightRawImgFmt fmt</i> : See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a> . <i>JidoshaLightConfig* config</i> : See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a> . <i>JidoshaLightRecognition</i> : See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a> .
<b>Return</b>	See description of method <a href="#">jidoshaLight ANPR fromRawImgFmt</a>

### jl\_async\_ANPR\_fromImage

<b>Function Prototype</b>	<pre>int jl_async_ANPR_fromImage (     JidoshaLightHandle* handle,     JidoshaLightImage* img,     JidoshaLightConfig* config );</pre>
<b>Description</b>	See description of method <a href="#">jidoshaLight ANPR fromImage</a> .
<b>Parameters</b>	<i>JidoshaLightHandle* handle</i> : Pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> <i>JidoshaLightImage* img</i> : See description of method <a href="#">jidoshaLight ANPR fromImage</a> . <i>JidoshaLightConfig* config</i> : See description of method <a href="#">jidoshaLight ANPR fromImage</a> .
<b>Return</b>	See description of method <a href="#">jidoshaLight ANPR fromImage</a> .

### jl\_async\_ANPR\_multi\_fromImage

<b>Function Prototype</b>	<pre>int jl_async_ANPR_multi_fromImage (     JidoshaLightHandle* handle,     JidoshaLightImage* img,     JidoshaLightConfig* config,     int maxPlates );</pre>
<b>Description</b>	Recognizes multiple plates from a previously loaded <i>JidoshaLightImage</i> , <a href="#">causing multiple calls to the callback function</a> . A set of recognitions belonging to the same image can be identified by the <i>int frameId</i> field of the <a href="#">JidoshaLightRecognition structure</a> . See description of method <a href="#">jidoshaLight ANPR multi fromImage</a> for details.
<b>Parameters</b>	<i>JidoshaLightHandle* handle</i> : Pointer to the <i>handle</i> created by <a href="#">jl_async_create_handle</a> <i>JidoshaLightImage* img</i> : See description of method <a href="#">jidoshaLight ANPR multi fromImage</a> . <i>JidoshaLightConfig* config</i> : See description of method <a href="#">jidoshaLight ANPR multi fromImage</a> . <i>int maxPlates</i> : See description of method <a href="#">jidoshaLight ANPR multi fromImage</a> .
<b>Return</b>	See description of method <a href="#">jidoshaLight ANPR multi fromImage</a> .

## JidoshaLight C/C++ API (Server)

The Server API extends the Local API, allowing you to create and configure a plate-reading server for use with remote APIs. It should be used in conjunction with the *libjidoshaLight.so* library.

```
//=====
// TYPES
//=====
typedef struct JidoshaLightServer JidoshaLightServer;

typedef struct JidoshaLightServerConfig
```

```

{
    int port;
    int conns;
    int threads;
    int threadQueueSize;
    int queueTimeout;
} JidoshaLightServerConfig;

//=====
// FUNCTIONS
//=====
JL_API JidoshaLightServer* jidoshaLightServer_create(
    JidoshaLightServerConfig* serverConfig
    );

JL_API int jidoshaLightServer_destroy(
    JidoshaLightServer* handler
    );

```

### Types

#### struct JidoshaLightServer

<b>Description</b>	The purpose of this structure is to store the plate recognition server object.
<b>Members</b>	None

#### struct JidoshaLightServerConfig

<b>Description</b>	The purpose of this structure is to configure the behavior of the library when acting as a plate recognition server.
<b>Members</b>	<i>int port</i> : TCP port number used for message exchange. <i>int conns</i> : number of concurrent client connections accepted by the server. <i>int threads</i> : number of parallel processing threads started by the server. <i>int threadQueueSize</i> : Maximum request queue size for each processing thread. <i>int queueTimeout</i> : maximum waiting time of a request in the processing queue in milliseconds (ms). The value 0 indicates that there is no timeout.

### Methods

#### jidosaLightServer\_create

<b>Function Prototype</b>	<pre>JidoshaLightServer* jidoshaLightServer_create(     JidoshaLightServerConfig* serverConfig     );</pre>
<b>Description</b>	Creates a plate recognition server instance. Uses the configuration structure pointed out by <i>JidoshaLightServerConfig* serverConfig</i> and returns the handler pointer of type <i>JidoshaLightServer</i> .
<b>Parameters</b>	<i>serverConfig</i> : pointer to <i>struct JidoshaLightServerConfig</i> structure with server configuration
<b>Return</b>	Returns a pointer to the handle of type <i>JidoshaLightServer</i> or <i>NULL</i> in case of error.

#### jidosaLightServer\_destroy

<b>Function Prototype</b>	<pre>int jidoshaLightServer_destroy(     JidoshaLightServer* handler     );</pre>
<b>Description</b>	Deallocates the server instance identified by its handler.

<b>Parameters</b>	handler: Pointer to server instance.
<b>Return</b>	<i>JIDOSHA_LIGHT_SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

## Java JidoshaLight API

The Java API of the JidoshaLight library has variations between the Linux and Android™ versions of the SDK.

The Linux version is a simple *wrapper* over API C while the Android™ version has specialized processing functions that better fit this development environment. Methods specific to one or the other platform are specified in the method description.

### Java JidoshaLight API (Local)

```
public class JidoshaLight {

    //=====
    // CODES
    //=====
    /* enum JidoshaLightVehicleType */
    public static final int VEHICLE_TYPE_CAR           = 1;
    public static final int VEHICLE_TYPE_MOTO         = 2;
    public static final int VEHICLE_TYPE_BOTH         = 3;

    /* enum JidoshaLightMode */
    public static final int MODE_DISABLE              = 0;
    public static final int MODE_FAST                 = 1;
    public static final int MODE_NORMAL               = 2;
    public static final int MODE_SLOW                 = 3;
    public static final int MODE_ULTRA_SLOW           = 4;

    /* enum JidoshaLightCountryCode */
    public static final int COUNTRY_CODE_CONESUL     = 0;
    public static final int COUNTRY_CODE_SAFETYPLATES = 3;
    public static final int COUNTRY_CODE_ARGENTINA    = 32;
    public static final int COUNTRY_CODE_BOLIVIA     = 68;
    public static final int COUNTRY_CODE_BRAZIL      = 76;
    public static final int COUNTRY_CODE_CHILE       = 152;
    public static final int COUNTRY_CODE_COLOMBIA    = 170;
    public static final int COUNTRY_CODE_COSTA_RICA  = 188;
    public static final int COUNTRY_CODE_ECUADOR     = 218;
    public static final int COUNTRY_CODE_FRANCE      = 250;
    public static final int COUNTRY_CODE_ITALY       = 380;
    public static final int COUNTRY_CODE_MEXICO      = 484;
    public static final int COUNTRY_CODE_NETHERLANDS = 528;
    public static final int COUNTRY_CODE_PANAMA      = 591;
    public static final int COUNTRY_CODE_PARAGUAY    = 600;
    public static final int COUNTRY_CODE_PERU        = 604;
    public static final int COUNTRY_CODE_EGYPT      = 818;
    public static final int COUNTRY_CODE_USA        = 840;
    public static final int COUNTRY_CODE_URUGUAY    = 858;

    /* enum JidoshaLightReturnCode */

```

```

/* success */
public static final int SUCCESS = 0;
/* basic errors */
public static final int ERROR_FILE_NOT_FOUND = 1;
public static final int ERROR_INVALID_IMAGE = 2;
public static final int ERROR_INVALID_IMAGE_TYPE = 3;
public static final int ERROR_INVALID_PROPERTY = 4;
public static final int ERROR_COUNTRY_NOT_SUPPORTED = 5;
public static final int ERROR_API_CALL_NOT_SUPPORTED = 6;
public static final int ERROR_INVALID_ROI = 7;
public static final int ERROR_INVALID_HANDLE = 8;
public static final int ERROR_API_CALL_HAS_NO_EFFECT = 9;
public static final int ERROR_INVALID_IMAGE_SIZE = 10;
/* license errors */
public static final int ERROR_LICENSE_INVALID = 16;
public static final int ERROR_LICENSE_EXPIRED = 17;
public static final int ERROR_LICENSE_MAX_THREADS_EXCEEDED = 18;
public static final int ERROR_LICENSE_UNTRUSTED_RTC = 19;
/* others */
public static final int ERROR_OTHER = 999;

/* enum JidoshaLightReturnCodeNetwork */
/* network errors */
public static final int ERROR_SERVER_CONNECT_FAILED = 100;
public static final int ERROR_SERVER_DISCONNECTED = 101;
public static final int ERROR_SERVER_QUEUE_TIMEOUT = 102;
public static final int ERROR_SERVER_QUEUE_FULL = 103;
public static final int ERROR_SOCKET_IO_ERROR = 104;
public static final int ERROR_SOCKET_WRITE_FAILED = 105;
public static final int ERROR_SOCKET_READ_TIMEOUT = 106;
public static final int ERROR_SOCKET_INVALID_RESPONSE = 107;
public static final int ERROR_HANDLE_QUEUE_FULL = 108;
public static final int ERROR_SERVER_CONN_LIMIT_REACHED = 213;
public static final int ERROR_SERVER_VERSION_NOT_SUPPORTED = 214;
public static final int ERROR_SERVER_NOT_READY = 215;

/* Raw image pixel format */
public static final int IMG_FMT_XRGB_8888 = 0;
public static final int IMG_FMT_RGB_888 = 1;
public static final int IMG_FMT_LUMA = 2;
public static final int IMG_FMT_YUV420 = 3;

//=====
// TYPES
//=====
public static class Config {
    public int vehicleType = VEHICLE_TYPE_BOTH;
    public int processingMode = MODE_ULTRA_SLOW;
    public int timeout = 0;
    public int countryCode = COUNTRY_CODE_BRAZIL;

    public float minProbPerChar = 0.85f;
    public int maxLowProbabilityChars = 0;
    public byte lowProbabilityChar = '?';
    public float avgPlateAngle = 0.0f;
    public float avgPlateSlant = 0.0f;

```

```
public int    maxCharHeight    = 60;
public int    minCharHeight    = 9;
public int    maxCharWidth     = 40;
public int    minCharWidth     = 1;
public int    avgCharHeight    = 20;
public int    avgCharWidth     = 7;

public int[]  xRoi              = new int[4];
public int[]  yRoi              = new int[4];
}

public static class Recognition {
    public String plate;
    public float[] probabilities;

    public int xText;
    public int yText;
    public int widthText;
    public int heightText;

    public int[] xChar;
    public int[] yChar;
    public int[] widthChar;
    public int[] heightChar;

    public int textColor;
    public int isMotorcycle;
    public int countryCode;

    /* JidoshaLightJidoshaLightRecognitionInfo */
    public double totalTime;
    public double localizationTime;
    public double segmentationTime;
    public double classificationTime;
    public double loadDecodeTime;
    public int[] libVersion;
    public String libSHA1;
}

public static class LicenseInfo {
    public String serial;
    public String customer;
    public int maxThreads;
    public int maxConnections;
    public int state;
    public int ttl;
}

public static class Version {
    public int major;
    public int minor;
    public int release;
}

/* STATIC METHODS */
/* PROCESSING [LINUX ONLY] */
```

```
public static native int ANPR_fromFile(
    String filename,
    Config config,
    Recognition rec
);

public static native int ANPR_fromMemory(
    byte[] buffer,
    int bufferSize,
    Config config,
    Recognition rec
);

public static native int ANPR_fromLuma(
    byte[] luma,
    int width,
    int height,
    Config config,
    Recognition rec
);

/* PROCESSING [ANDROID ONLY] */
public static native int ANPR_fromBitmap(
    Bitmap bitmap,
    Config config,
    Recognition rec
);

public static int ANPR_fromUri(
    Context context,
    Uri uri,
    Config config,
    Recognition rec
);

/* PROCESSING [LINUX AND ANDROID] */
public static native int ANPR_fromImage(
    JidoshaLightImage img,
    Config config,
    Recognition rec
);

public static native int ANPR_multi_fromImage(
    JidoshaLightImage img,
    Config config,
    int maxPlates,
    List<Recognition> recList
);

//=====
// LICENSE [ANDROID]
//=====
public static final int LICENSE_REQUEST_OK = 200;
public static final int LICENSE_REQUEST_BAD_REQUEST = 400;
public static final int LICENSE_REQUEST_NOT_FOUND = 404;
public static final int LICENSE_REQUEST_UNAUTHORIZED = 401;
```

```

public static final int LICENSE_REQUEST_FORBIDDEN = 403;
public static final int LICENSE_REQUEST_PAYMENT_REQUIRED = 402;
public static final int LICENSE_REQUEST_INTERNAL_SERVER_ERROR = 500;
public static final int LICENSE_REQUEST_SERVICE_UNAVAILABLE = 503;
public static final int LICENSE_REQUEST_ORIGIN_IS_UNREACHABLE = 523;

public static native String getAndroidFingerprint(Activity androidActivity);
public static native int getLicenseFromServer(Activity activity, String savePath,
String user, String key);
public static native int setLicenseFromData(Activity androidActivity, byte[] data, int
dataSize);

/* STATUS */
public static native int getVersion(Version version);
public static native String getBuildSHA1();
public static native String getBuildFlags();

//=====
// LICENSE STATUS
//=====
public static native int getLicenseInfo(LicenseInfo info);

//=====
// SHARED LIBRARY LOADER
//=====
public static void loadLibrary() {
    System.loadLibrary("jidoshalightjava");
}
}

```

## Types

class JidoshaLightImage	
<b>Description</b>	It has the same features as the <a href="#">struct JidoshaLightImage</a> from API C.
<b>Public methods</b>	<p><b>public JidoshaLightImage();</b>  Constructs a new object of type '<i>JidoshaLightImage</i>'. If the native handle allocation fails, launches a '<i>RuntimeException</i>'.  To prevent memory leaks, every '<i>JidoshaLightImage</i>' object created must be explicitly destroyed by the user using the '<i>destroy()</i>' function.</p> <p><b>public JidoshaLightImage duplicate();</b>  Duplicates an object of type '<i>JidoshaLightImage</i>' previously created and loaded in memory. The new object needs to be destroyed by the user using the '<i>destroy()</i>' function.</p> <p><b>public int destroy();</b>  Releases the memory allocated by the object.</p> <p><b>public int setLazyDecode(boolean enable);</b>  See <a href="#">struct JidoshaLightImage</a> from API C.</p> <p><b>public int loadFromFile(String filename);</b>  See <a href="#">struct JidoshaLightImage</a> from API C.</p>

	<pre>public int loadFromMemory(byte[] buffer);</pre> <p>See <a href="#">struct JidoshaLightImage</a> from API C.</p>
	<pre>public int loadFromRawImgFmt(byte[] buffer, int width, int height, int stride, int fmt);</pre> <p>See <a href="#">struct JidoshaLightImage</a> from API C.</p>

### class JidoshaLight.Config

Description	It has the same features as the <a href="#">struct JidoshaLightConfig</a> from API C.
Members	<p><i>int vehicleType</i>: indicates the type of plate that the OCR should look for. Possible values for this field are:</p> <ul style="list-style-type: none"> <li>JidoshaLight.VEHICLE_TYPE_CAR: ver JIDOSHA_LIGHT_VEHICLE_TYPE_CAR.</li> <li>JidoshaLight.VEHICLE_TYPE_MOTO: ver JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO.</li> <li>JidoshaLight.VEHICLE_TYPE_BOTH: ver JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH.</li> </ul> <p><i>int processingMode</i>: indicates the processing strategy adopted by the recognition algorithm. Possible values for this field are:</p> <ul style="list-style-type: none"> <li>JidoshaLight.MODE_DISABLE: ver JIDOSHA_LIGHT_MODE_DISABLE.</li> <li>JidoshaLight.MODE_FAST: ver JIDOSHA_LIGHT_MODE_FAST.</li> <li>JidoshaLight.MODE_NORMAL: ver JIDOSHA_LIGHT_MODE_NORMAL.</li> <li>JidoshaLight.MODE_SLOW: ver JIDOSHA_LIGHT_MODE_SLOW.</li> <li>JidoshaLight.MODE_ULTRA_SLOW: ver JIDOSHA_LIGHT_MODE_ULTRA_SLOW.</li> </ul> <p><i>int timeout</i>: see API C.  <i>int countryCode</i>: see API C.  <i>float minProbPerChar</i>: see API C.  <i>int maxLowProbabilityChars</i>: see API C.  <i>byte lowProbabilityChar</i>: see API C.  <i>float avgPlateAngle</i>: see API C.  <i>float avgPlateSlant</i>: see API C.  <i>int maxCharHeight</i>: see API C.  <i>int minCharHeight</i>: see API C.  <i>int maxCharWidth</i>: see API C.  <i>int minCharWidth</i>: see API C.  <i>int avgCharHeight</i>: see API C.  <i>int avgCharWidth</i>: see API C.  <i>int xRoi[]</i> and <i>int yRoi []</i>: x and y coordinates of the four points of the region of interest of the image (ROI - Region Of Interest). See API C for more information.</p>

### class JidoshaLight.Recognition

Description	Concatenates the features of the <a href="#">struct JidoshaLightRecognition</a> and <a href="#">struct JidoshaLightRecognitionInfo</a> types from API C.
Members	<p><i>String plate</i>: string containing the characters of the recognized plate or empty if the plate was not found.  <i>float probabilities[]</i>: see API C.  <i>int frameId</i>: see API C.  <i>int xText</i> and <i>int yText</i>: see API C.  <i>int widthText</i>: see API C.  <i>int heightText</i>: see API C.  <i>int xChar[]</i> and <i>int yChar[]</i>: see API C.  <i>int widthChar[]</i>: see API C.  <i>int heightChar[]</i>: see API C.  <i>int textColor</i>: see API C.</p>



	<i>int isMotorcycle</i> : see API C. <i>double totalTime</i> : see API C. <i>double localizationTime</i> : see API C. <i>double segmentationTime</i> : see API C. <i>double classificationTime</i> : see API C. <i>double loadDecodeTime</i> : see API C. <i>int libVersion[]</i> : see API C. <i>String libSHA1[]</i> : see API C.
--	--

<b>class JidoshaLight.LicenseInfo</b>	
<b>Description</b>	Type used by the <a href="#">getLicenseInfo</a> function to return information about the library license.
<b>Members</b>	<i>Serial string</i> : serial number of the license in decimal <i>String customer</i> : name of the customer who purchased the license <i>int maxThreads</i> : maximum number of processing threads enabled <i>int maxConnections</i> : maximum number of parallel connections enabled <i>int state</i> : license status (see <a href="#">Function return codes</a> ) <i>int ttl</i> : time-to-live in hours for RTC type licenses. This field has the value <b>-1</b> if the license is not expired

<b>class JidoshaLight.Version</b>	
<b>Description</b>	Type used by the <i>getVersion</i> function to return the library version.
<b>Members</b>	<i>int major</i> : Major value of the version. <i>int minor</i> : Minor value of the version. <i>int release</i> : Value of the release of the version.

## Methods

<b>JidoshaLight.ANPR_fromImage [LINUX e ANDROID]</b>	
<b>Function Prototype</b>	<pre>public static native int ANPR_fromImage(     JidoshaLightImage img,     Config config,     Recognition rec );</pre>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight ANPR_fromImage</a> function of API C.
<b>Parameters</b>	<i>img</i> : object of type <a href="#">JidoshaLightImage</a> containing the image to be recognized. <i>config</i> : object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. Passing 'null' in this parameter implies using the default library settings. <i>rec</i> : object of type <a href="#">JidoshaLight.Recognition</a> where the reading result will be stored.
<b>Return</b>	<a href="#">JidoshaLight.SUCCESS</a> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> ).

<b>JidoshaLight.ANPR_multi_fromImage [LINUX and ANDROID]</b>	
<b>Function Prototype</b>	<pre>public static native int ANPR_multi_fromImage(     JidoshaLightImage img,     Config config,     int maxPlates,     List&lt;Recognition&gt; recList );</pre>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight ANPR_multi_fromImage</a> function of API C.
<b>Parameters</b>	<i>img</i> : object of type <a href="#">JidoshaLightImage</a> containing the image to be recognized.

	<i>config</i> : object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. Passing 'null' in this parameter implies using the default library settings. <i>maxPlates</i> : maximum number of plates to be recognized in the image (1 to 8) <i>recList</i> : list of objects of type <a href="#">JidoshaLight.Recognition</a> where the reading results will be stored. It has a size equal to maxPlates if the function returns successfully.
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> ).

### JidoshaLight.ANPR\_fromFile [LINUX]

<b>Function Prototype</b>	<pre>public static native int ANPR_fromFile(     String filename,     Config config,     Recognition rec );</pre>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight ANPR_fromFile</a> function of API C.
<b>Parameters</b>	<i>filename</i> : string containing the path to the image file to be recognized. <i>config</i> : object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. Passing 'null' in this parameter implies using the default library settings. <i>rec</i> : object of type <a href="#">JidoshaLight.Recognition</a> where the reading result will be stored.
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> ).

### JidoshaLight.ANPR\_fromMemory [LINUX]

<b>Function Prototype</b>	<pre>public static native int ANPR_fromMemory(     byte[] buffer,     int bufferSize,     Config config,     Recognition rec );</pre>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight ANPR_fromMemory</a> function of API C.
<b>Parameters</b>	<i>buffer</i> : array of bytes that contain the image. <i>int bufferSize</i> : size of the byte array. <i>config</i> : object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. A 'null' object in this parameter implies using the library's default settings. <i>rec</i> : object of type <a href="#">JidoshaLight.Recognition</a> where the reading result will be stored.
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> ).

### JidoshaLight.ANPR\_fromLuma [LINUX]

<b>Function Prototype</b>	<pre>public static native int ANPR_fromLuma(     byte[] luma,     int width,     int height,     Config config,     Recognition rec );</pre>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight ANPR_fromLuma</a> function of API C.
<b>Parameters</b>	<i>luma</i> : array of bytes containing the image in 8-bit raw grayscale format. <i>width</i> : image width.

	<p><i>height</i>: altura da imagem.</p> <p><i>config</i>: object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. A 'null' object in this parameter implies using the library's default settings.</p> <p><i>rec</i>: object of type <a href="#">JidoshaLight.Recognition</a> where the reading result will be stored.</p>
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> ).

### JidoshaLight.ANPR\_fromBitmap [ANDROID]

<b>Function Prototype</b>	<pre>public static native int ANPR_fromBitmap (     Bitmap bitmap,     Config config,     Recognition rec );</pre>
<b>Description</b>	Recognises a plate from the ' <i>bitmap</i> ' object using the settings in ' <i>config</i> '. The recognition result is returned in ' <i>rec</i> '. If an error occurs in the recognition process, the ' <i>rec</i> ' object will contain an empty string as a plate and a value other than ' <i>JidoshaLight.SUCCESS</i> ' will be returned by the function. Possible return values are defined in the ' <i>JidoshaLight</i> ' class and contain the prefix ' <i>ERROR_</i> '.
<b>Parameters</b>	<p><i>bitmap</i>: object of type '<i>android.graphics.Bitmap</i>' containing the image to be recognized in the format '<i>ARGB8888</i>'.</p> <p><i>config</i>: object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. A 'null' object in this parameter implies using the library's default settings.</p> <p><i>rec</i>: object of type <a href="#">JidoshaLight.Recognition</a> where the reading result will be stored.</p>
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

### JidoshaLight.ANPR\_fromUri [ANDROID]

<b>Function Prototype</b>	<pre>public static int ANPR_fromUri(     Context context,     Uri uri,     Config config,     Recognition rec );</pre>
<b>Description</b>	Recognises a plate from the 'Uri' of an image file. Internally this method calls the function <i>ANPR_fromBitmap</i> . The recognition result is returned in ' <i>rec</i> '. If an error occurs in the recognition process, the ' <i>rec</i> ' object will contain an empty string as a plate and a value other than ' <i>JidoshaLight.SUCCESS</i> ' will be returned by the function. Possible return values are defined in the ' <i>JidoshaLight</i> ' class and contain the prefix ' <i>ERROR_</i> '.
<b>Parameters</b>	<p><i>context</i>: object of type '<i>android.content.Context</i>' containing the context of the Activity.</p> <p><i>uri</i>: object of type '<i>android.net.Uri</i>' containing the '<i>uri</i>' of the image to be recognized.</p> <p><i>config</i>: object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. A 'null' object in this parameter implies using the library's default settings.</p> <p><i>rec</i>: object of type <a href="#">JidoshaLight.Recognition</a> where the reading result will be stored.</p>
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

### JidoshaLight.getAndroidFingerprint [ANDROID]

<b>Function Prototype</b>	<pre>static native String getAndroidFingerprint(Activity androidActivity);</pre>
<b>Description</b>	Returns the unique identifier generated by the library installation.

<b>Parameters</b>	<i>androidActivity</i> : object of type ' <i>android.app.Activity</i> ' containing the reference to the main Activity of the application.
<b>Return</b>	String containing the unique facility identifier required for license file generation. This string must not be changed.

### JidoshaLight.getLicenseFromServer [ANDROID]

<b>Function Prototype</b>	<code>static native int getLicenseFromServer(Activity activity, String savePath, String user, String key);</code>
<b>Description</b>	Requires a license file from the Pumatronix's license server.
<b>Parameters</b>	<p><i>activity</i>: object of type '<i>android.app.Activity</i>' containing the reference to the main Activity of the application.</p> <p><i>savePath</i>: path where the received license file should be saved in case of success.</p> <p><i>user</i>: user to be used in the request or '<i>null</i>' otherwise.</p> <p><i>key</i>: key to be used in the request or '<i>null</i>' otherwise.</p>
<b>Return</b>	<ul style="list-style-type: none"> <li>• JidoshaLight.LICENSE_REQUEST_OK</li> <li>• JidoshaLight.LICENSE_REQUEST_BAD_REQUEST</li> <li>• JidoshaLight.LICENSE_REQUEST_NOT_FOUND</li> <li>• JidoshaLight.LICENSE_REQUEST_UNAUTHORIZED</li> <li>• JidoshaLight.LICENSE_REQUEST_FORBIDDEN</li> <li>• JidoshaLight.LICENSE_REQUEST_PAYMENT_REQUIRED</li> <li>• JidoshaLight.LICENSE_REQUEST_INTERNAL_SERVER_ERROR</li> <li>• JidoshaLight.LICENSE_REQUEST_SERVICE_UNAVAILABLE</li> <li>• JidoshaLight.LICENSE_REQUEST_ORIGIN_IS_UNREACHABLE</li> </ul> <p>See sample Android for more information.</p>

### JidoshaLight.setLicenseFromData [ANDROID]

<b>Function Prototype</b>	<code>static native int setLicenseFromData(Activity androidActivity, byte[] data, int dataSize);</code>
<b>Description</b>	This method is used to configure the library license file from the contents of the ' <i>byte[] data</i> ' buffer.
<b>Parameters</b>	<p><i>androidActivity</i>: object of type '<i>android.app.Activity</i>' containing the reference to the main Activity of the application.</p> <p><i>data</i>: buffer with the contents of the license file.</p> <p><i>dataSize</i>: license file size - '<i>data.len</i>'</p>
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> return code in case of success, another code otherwise (see <a href="#">Function return codes</a> )

### JidoshaLight.getVersion

<b>Function Prototype</b>	<code>public static native int getVersion(Version version);</code>
<b>Description</b>	Returns the version of the library in major.minor.release format.
<b>Parameters</b>	<i>version</i> : object of type <a href="#">JidoshaLight.Version</a> with version number
<b>Return</b>	Always returns <i>JidoshaLight.SUCCESS</i> .

JidoshaLight.getBuildSHA1	
<b>Function Prototype</b>	<code>String getBuildSHA1();</code>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight_getBuildSHA1</a> function of API C.
<b>Parameters</b>	None
<b>Return</b>	Returns a String containing the value of the build SHA1.

JidoshaLight.getBuildFlags	
<b>Function Prototype</b>	<code>String getBuildFlags();</code>
<b>Description</b>	It has the same behavior as the <a href="#">JidoshaLight_getBuildFlags</a> function of API C.
<b>Parameters</b>	None
<b>Return</b>	Returns a String containing the build flags of the library.

JidoshaLight.getLicenseInfo	
<b>Function Prototype</b>	<code>public static native int getLicenseInfo(LicenseInfo info);</code>
<b>Description</b>	Function used to read the license information used by the JidoshaLight library.
<b>Parameters</b>	<i>info</i> : object of type <a href="#">JidoshaLight.LicenseInfo</a>
<b>Return</b>	Returns JIDOSHA_LIGHT_SUCCESS on success.

### Function return codes

Codes returned by JidoshaLight library functions are defined as '*public static final int*' attributes within the *JidoshaLight* class. The codes returned in the Linux and ANDROID versions of the SDK are:

- `JidoshaLight.ERROR_FILE_NOT_FOUND`: returned by the functions '*ANPR\_fromFile*' and '*ANPR\_fromUri*' when the specified file path does not exist.
- `JidoshaLight.ERROR_INVALID_IMAGE`: returned by the functions '*ANPR*'. Occurs when the past image is corrupted.
- `JidoshaLight.ERROR_INVALID_IMAGE_TYPE`: returned by the functions '*ANPR*'. Occurs when attempting to process an image of unsupported format. This error code is not returned by the Android version of the API.
- `JidoshaLight.ERROR_INVALID_PROPERTY`: returned by all functions that have arguments. Occurs when the argument is invalid.
- `JidoshaLight.ERROR_COUNTRY_NOT_SUPPORTED`: returned by the '*ANPR*' functions when the country code provided in the configuration structure is not supported by the library.
- `JidoshaLight.ERROR_API_CALL_NOT_SUPPORTED`: returned when an API function is not available for a given platform.
- `JidoshaLight.ERROR_INVALID_ROI`: returned when an invalid region of interest is provided. See the description of the '*struct JidoshaLightConfig*' for more information.
- `JidoshaLight.ERROR_INVALID_HANDLE`: returned when the *handle* passed to the function was not initialized correctly.
- `JidoshaLight.ERROR_API_CALL_HAS_NO_EFFECT`: Returned when an API function had no effect when executed. It can occur when there is precedence between calls.

- JidoshaLight.ERROR\_LICENSE\_INVALID: returned by the functions 'ANPR' when the license provided is not valid (for licenses of the *hardkey* type, it means that it is not connected or has problems). Contact Pumatronix Equipamentos Eletrônicos for more information.
- JidoshaLight.ERROR\_LICENSE\_EXPIRED: returned by the functions 'ANPR' when the license usage period expired. This type of error only happens for demo type licenses. Contact Pumatronix Equipamentos Eletrônicos for more information.
- JidoshaLight.ERROR\_LICENSE\_MAX\_THREADS\_EXCEEDED: returned by the 'ANPR' functions when the maximum number of competing threads exceeds that allowed by the license.
- JidoshaLight.ERROR\_LICENSE\_UNTRUSTED\_RTC: returned by the functions 'ANPR' when a license with use-by date does not have a reliable time/date reference available.
- JidoshaLight.ERROR\_OTHER: Returned when an unexpected error occurs. Contact Pumatronix Equipamentos Eletrônicos for support.

## JidoshaLight Java API (Asynchronous Remote)



**Note: All functions of the Asynchronous Remote API are available for Android and Linux.**

```
package br.gaussian.jidoshalight;

public class JidoshaLightRemote {

    //=====
    // TYPES
    //=====
    public static class Config {
        public int    queueSize;
        public String ip;
        public int    port;
    }

    public static class ServerInfo {
        public JidoshaLight.LicenseInfo license;
        public JidoshaLight.Version version;
    }

    //=====
    // Callback interface
    //=====
    public interface Callbacks {
        void on_lpr_result_cb(JidoshaLight.Recognition rec, int code, byte[] buffer);
    }

    //=====
    // FUNCTION CALLS
    //=====
    public static native long create_handle(Config config, Callbacks callbacks);
    public static native int destroy_handle(long handle);
    public static native int connect(long handle);
    public static native int connect_info(long handle, ServerInfo info);
    public static native int get_localqueue_size(long handle);
    public static native int ANPR_fromMemory (
        long handle,
        byte[] buffer,
```

```

    JidoshaLight.Config config
);
public static native int ANPR_fromRawImgFmt (
    long handle,
    byte[] buffer,
    int width,
    int height,
    int stride,
    int fmt,
    JidoshaLight.Config config
);

//=====
// LIBRARY STATUS
//=====
public static class Version {
    public int major;
    public int minor;
    public int release;
}

public static native int getVersion(Version version);
public static native String getBuildSHA1();
public static native String getBuildFlags();
}

```

## Types

<b>class JidoshaLightRemote.Config</b>	
<b>Description</b>	The purpose of this structure is to store the client object of a plate recognition server.
<b>Members</b>	<i>queueSize</i> : maximum size of pending requests for the <i>handle</i> . <i>ip</i> : string containing the IP address of the server. <i>port</i> : integer containing the TCP port of the server.

<b>JidoshaLightRemote.Callbacks interface</b>	
<b>Description</b>	Interface that defines the format of the callback for receiving events from the server.
<b>Members</b>	<i>on_lpr_result_cb(JidoshaLight.Recognition rec, int code, byte[] buffer)</i> <ul style="list-style-type: none"> <li><i>rec</i>: object containing the result of the recognition</li> <li><i>code</i>: requisition return code</li> <li><i>buffer</i>: buffer with the image used in the recognition</li> </ul>

<b>class JidoshaLightRemote.ServerInfo</b>	
<b>Description</b>	Struct used to store license and version information of a JidoshaLight server.
<b>Members</b>	<i>JidoshaLight.LicenseInfo license</i> : information about the license used by the server <i>JidoshaLight.Version version</i> : server library version

## Methods

<b>JidoshaLightRemote.create_handle</b>	
<b>Function Prototype</b>	<code>long create_handle (Config config, Callbacks callbacks);</code>

<b>Description</b>	Creates the handle of an assicron client for connection to a plate recognition server. A call to <i>destroy_handle</i> must be made to release the allocated resources.
<b>Parameters</b>	A ' <i>JidoshaLightRemote.Config</i> ' object containing the server configuration parameters and an object that implements the ' <i>JidoshaLightRemote.CallBacks</i> ' interface.
<b>Return</b>	If successful, returns the memory address of the created <i>handle</i> . Otherwise, returns '0'.

### **JidoshaLightRemote.destroy\_handle**

<b>Function Prototype</b>	<code>int destroy_handle (long handle);</code>
<b>Description</b>	Releases the resources allocated to the <i>handle</i> . To prevent an already released <i>handle</i> from being improperly reused, it is recommended that after calling this function, assign '0' to the <i>handle</i> value, i.e. <code>mHandle = 0;</code>
<b>Parameters</b>	A ' <i>long</i> ' containing the memory address of a valid ' <i>JidoshaLightRemote</i> ' <i>handle</i> .
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> in case of success.

### **JidoshaLightRemote.connect**

<b>Function Prototype</b>	<code>int connect (long handle);</code>
<b>Description</b>	Establishes the session with the plate recognition server for a given handle.
<b>Parameters</b>	<i>long handle</i> : long containing the memory address of a valid <i>JidoshaLightRemote</i> handle.
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> on success, otherwise see error codes.

### **JidoshaLightRemote.connect\_info**

<b>Function Prototype</b>	<code>int connect_info(long handle, ServerInfo info);</code>
<b>Description</b>	It has the same functionality as the <a href="#">connect</a> function but receives an additional parameter to receive information about the license and the server version.
<b>Parameters</b>	<i>long handle</i> : long containing the memory address of a valid <i>JidoshaLightRemote</i> handle. <i>ServerInfo* info</i> : Object of type <i>JidoshaLightRemote.ServerInfo</i>
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> on success, otherwise see error codes.

### **JidoshaLightRemote.get\_localqueue\_size**

<b>Function Prototype</b>	<code>int get_localqueue_size (long handle);</code>
<b>Description</b>	Returns the queue size of pending requests on the client side for a given handle.
<b>Parameters</b>	A ' <i>long</i> ' containing the memory address of a valid <i>JidoshaLightRemote</i> handle.
<b>Return</b>	Returns the number of pending requisitions in the local (client) queue.

### **JidoshaLightRemote.ANPR\_fromMemory**

<b>Function Prototype</b>	<code>int ANPR_fromMemory (     long handle,     byte[] buffer,     JidoshaLight.Config config );</code>
---------------------------	--



<b>Description</b>	Remote version of the ' <i>JidoshaLight.ANPR_fromMemory</i> ' call.
<b>Parameters</b>	<i>handle</i> : long containing the memory address of a valid <i>JidoshaLightRemote</i> handle. <i>buffer</i> : array of bytes containing the image to be recognized in JPEG, PNG or bmp format. <i>config</i> : object of type <a href="#">JidoshaLight.Config</a> containing the settings for the library. A 'null' object in this parameter implies using the library's default settings.
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> on success, otherwise see error codes.

### JidoshaLightRemote.ANPR\_fromRawImgFmt

<b>Function Prototype</b>	<pre>int ANPR_fromRawImgFmt (     long handle,     byte[] buffer,     int width,     int height,     int stride,     int fmt,     JidoshaLight.Config config );</pre>
<b>Description</b>	Sends a plate recognition request from an image in RAW format.
<b>Parameters</b>	<i>handle</i> : long containing the memory address of a valid <i>JidoshaLightRemote</i> handle. <i>buffer</i> : array of bytes containing the image to be recognized in any of the supported raw formats (see definitions in class ' <i>JidoshaLight</i> '). <i>width</i> : image width. <i>height</i> : image height <i>stride</i> : number of bytes per image line <i>fmt</i> : image format (see definitions in class ' <i>JidoshaLight</i> '). <i>config</i> : object of type <i>JidoshaLight.Config</i> containing the settings for the library. A 'null' object in this parameter implies using the library's default settings.
<b>Return</b>	<i>JidoshaLight.SUCCESS</i> on success, otherwise see error codes.

### JidoshaLightRemote.getVersion

<b>Function Prototype</b>	<pre>public static native int getVersion(Version version);</pre>
<b>Description</b>	Returns the version of the library in major.minor.release format.
<b>Parameters</b>	<i>version</i> : object of type ' <i>Version</i> ' with version number
<b>Return</b>	Always returns ' <i>JidoshaLight.SUCCESS</i> '.

### JidoshaLightRemote.getBuildSHA1

<b>Function Prototype</b>	<pre>String getBuildSHA1();</pre>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight_getBuildSHA1</a> function of API C.
<b>Parameters</b>	None
<b>Return</b>	Returns a String containing the value of the build SHA1.

JidoshaLightRemote.getBuildFlags	
<b>Function Prototype</b>	<code>String getBuildFlags();</code>
<b>Description</b>	It has the same behavior as the <a href="#">JidoshaLight.getBuildFlags</a> function of API C.
<b>Parameters</b>	None
<b>Return</b>	Returns a String containing the build flags of the library.

JidoshaLightRemote.getBuildFlags	
<b>Function Prototype</b>	<code>String getBuildFlags();</code>
<b>Description</b>	It has the same behavior as the <a href="#">JidoshaLight.getBuildFlags</a> function of API C.
<b>Parameters</b>	None
<b>Return</b>	Returns a String containing the build flags of the library.

## Java JidoshaLight API (Server)



**Note: All functions of the Server API are available for Android and Linux.**

```

package br.gaussian.jidoshalight;

public class JidoshaLightServer {

    //=====
    // TYPES
    //=====
    public static class Config {
        public int port          = 51000;
        public int conns         = 1;
        public int threads       = 8;
        public int threadQueueSize = 1000;
        public int queueTimeout  = 0;
    }

    //=====
    // FUNCTION CALLS
    //=====
    public static native long create_handle(Config config);
    public static native int  destroy_handle(long handle);

    //=====
    // LIBRARY STATUS
    //=====
    public static class Version {
        public int major;
        public int minor;
        public int release;
    }
}

```

```

public static native int getVersion(Version version);
public static native String getBuildSHA1();
public static native String getBuildFlags();
}

```

## Types

<b>class JidoshaLightServer.Config</b>	
<b>Description</b>	Configuration structure for a plate reading server.
<b>Members</b>	<i>port</i> : server connection port. <i>conns</i> : maximum number of simultaneous connections the server can accept (maximum value limited by license) <i>threads</i> : maximum number of processing threads that the server can use (maximum value limited by the license). Threads are shared between connections. <i>threadQueueSize</i> : Maximum size of the processing queue for each thread. <i>queueTimeout</i> : Maximum time a request can wait in the processing queue.

## Methods

<b>JidoshaLightServer.create_handle</b>	
<b>Function Prototype</b>	<code>long create_handle (Config config);</code>
<b>Description</b>	Creates a <i>handle</i> for the plate recognition server and initializes it. A call to ' <i>destroy_handle</i> ' must be made to release the allocated resources.
<b>Parameters</b>	A ' <i>JidoshaLightServer.Config</i> ' object containing the server configuration parameters.
<b>Return</b>	If successful, returns the memory address of the created <i>handle</i> . Otherwise, returns '0'.

<b>JidoshaLightServer.destroy_handle</b>	
<b>Function Prototype</b>	<code>void destroy_handle (long handle);</code>
<b>Description</b>	Releases the resources allocated to the <i>handle</i> and stops the server. To prevent an already released <i>handle</i> from being improperly reused, it is recommended that after calling this function, assign '0' to the <i>handle</i> value, i.e. <code>mHandle = 0;</code>
<b>Parameters</b>	A ' <i>long</i> ' containing the memory address of a valid <i>JidoshaLightServer</i> handle.
<b>Return</b>	'0' if the <i>handle</i> cannot be created. Otherwise, it returns non-zero.

<b>JidoshaLightServer.getVersion</b>	
<b>Function Prototype</b>	<code>public static native int getVersion(Version version);</code>
<b>Description</b>	Returns the version of the library in major.minor.release format.
<b>Parameters</b>	<i>version</i> : object of type ' <i>Version</i> ' with version number
<b>Return</b>	Always returns 'JidoshaLight.SUCCESS'.

<b>JidoshaLightServer.getBuildSHA1</b>	
<b>Function Prototype</b>	<code>String getBuildSHA1();</code>
<b>Description</b>	It has the same behavior as the <a href="#">jidoshaLight_getBuildSHA1</a> function of API C.

<b>Parameters</b>	None
<b>Return</b>	Returns a String containing the value of the build SHA1.

<b>JidoshaLightServer.getBuildFlags</b>	
<b>Function Prototype</b>	<code>String getBuildFlags();</code>
<b>Description</b>	It has the same behavior as the <a href="#">JidoshaLight_getBuildFlags</a> function of API C.
<b>Parameters</b>	None
<b>Return</b>	Returns a String containing the build flags of the library.

## Java JidoshaLight API (IO/Mjpeg)

This API provides a video receiver in MJPEG (Motion JPEG) format. This video format is widely used by IP cameras.



**Note: All IO/Mjpeg API functions are available for Android and Linux.**

```
package br.gaussian.io;

public class Mjpeg {

    //=====
    // Error Codes
    //=====
    public static final int JL_FRAME_QUEUE_FULL           = 211;
    public static final int JL_LAST_FRAME_UNAVAILABLE    = 212;
    public static final int JL_MJPEG_HTTP_HEADER_OVERFLOW = 1001;
    public static final int JL_MJPEG_HTTP_RESPONSE_NOT_OK = 1002;
    public static final int JL_MJPEG_HTTP_CONTENT_TYPE_ERROR = 1003;
    public static final int JL_MJPEG_HTTP_CONTENT_LENGTH_ERROR = 1004;
    public static final int JL_MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND = 1005;
    public static final int JL_MJPEG_CONNECTION_CLOSED    = 1006;
    public static final int JL_MJPEG_CONNECT_FAILED      = 1007;

    //=====
    // Config interface
    //=====
    public static class Config {
        public String url;
        public int timeout;
        public int bufferSize;
    }

    //=====
    // Callback interface
    //=====
    public interface Callbacks {
        void frame_cb(byte[] frame);
        void error_cb(int code);
    }
}
```



```

public static native long   create_handle(Callbacks callbacks, Config config);
public static native void   destroy_handle(long handle);
public static native int    connect(long handle);
public static native byte[] get_frame(long handle);
}

```

### Types

<b>class Mjpeg.Config</b>	
<b>Description</b>	Configuration structure for a Mjpeg flow.
<b>Members</b>	<i>url</i> : String containing the URL of the Mjpeg stream in <code>http://&lt;IP&gt;[:PORT]/[PATH]</code> format. <i>timeout</i> : maximum interval between frames in milliseconds. Delays greater than 'timeout' are considered as connection loss. (Recommended values: 1000 to 5000). <i>bufferSize</i> : maximum number of frames that can be queued. This parameter should be greater than <b>0</b> and preferably equal to <b>1</b> . Values greater than <b>1</b> should be considered for cases where the callback ' <i>frame_cb</i> ' may take longer than the flow framerate.

<b>interface Mjpeg.Callbacks</b>	
<b>Description</b>	Interface that defines the callbacks generated by the Mjpeg flow.  <div style="display: flex; align-items: center;">  <p><b>Note 1: the execution of the callback cannot take long (see parameter '<i>bufferSize</i>').</b></p> </div> <div style="display: flex; align-items: center;">  <p><b>Note 2: Under no circumstances can one call '<i>destroy_handle(long handle)</i>' within a callback.</b></p> </div>
<b>Members</b>	<i>void frame_cb(byte[] frame)</i> : callback whenever a new frame is available. The frame comes in JPEG format. <i>void error_cb(int code)</i> : callback whenever an error occurs in the Mjpeg flow (see definition of errors below). Whenever there is a disconnect error, the flow will attempt to reestablish connectivity automatically. To stop the process, it is enough for the user to destroy the <i>handle</i> .

### Methods

<b>Mjpeg.create_handle</b>	
<b>Function Prototype</b>	<pre>long create_handle (     Callbacks callbacks,     Config config );</pre>
<b>Description</b>	Creates a handle for use in Mjpeg class functions. A call to 'destroy_handle' must be made to release the allocated resources.
<b>Parameters</b>	An object that implements the ' <i>Mjpeg.Callbacks interface</i> ' and a ' <i>Mjpeg.Config</i> ' object containing the flow configuration parameters.
<b>Return</b>	If successful, returns the memory address of the created <i>handle</i> . Otherwise, returns '0'.

<b>Mjpeg.destroy_handle</b>	
<b>Function Prototype</b>	<pre>void destroy_handle (long handle);</pre>

<b>Description</b>	Releases the resources allocated to the <i>handle</i> . To prevent an already released <i>handle</i> from being improperly reused, it is recommended that after calling this function, '0' is assigned to the <i>handle</i> value, i.e. <code>mHandle = 0;</code>
<b>Parameters</b>	A 'long' containing the memory address of a valid <i>Mjpeg</i> handle.
<b>Return</b>	'0' if the <i>handle</i> cannot be created. Otherwise, it returns non-zero.

### Mjpeg.connect

<b>Function Prototype</b>	<code>void connect (long handle);</code>
<b>Description</b>	Attempts to establish a connection to the URL defined in the creation of the <i>Mjpeg</i> handle.
<b>Parameters</b>	A 'long' containing the memory address of a valid <i>Mjpeg</i> handle.
<b>Return</b>	' <i>JidoshaLight.SUCCESS</i> ' in case of success. ' <i>Mjpeg.JL_MJPEG_CONNECT_FAILED</i> ' if the connection cannot be established immediately. In this case, the <i>handle</i> will not try to reconnect automatically, leaving it up to the user to call 'connect' again in time.

### Mjpeg.get\_frame

<b>Function Prototype</b>	<code>byte[] get_frame(long handle)</code>
<b>Description</b>	Returns the most recent frame of the Mjpeg receive queue. Consecutive calls to this function can return the same frame if no new frames have been received in the range.
<b>Parameters</b>	A 'long' containing the memory address of a valid <i>Mjpeg</i> handle.
<b>Return</b>	A byte array containing the last frame received in JPEG format. If no frame has been received by the time of the call, the function returns an array of size 0 ' <code>byteArray.length == 0</code> ' and the callback ' <code>error_cb</code> ' is called with code ' <code>JL_LAST_FRAME_UNAVAILABLE</code> '.

## Migration Guide - API 1 C/C++ JIDOSHA

The process of migrating a PC application that uses API 1 from the JIDOSHA library to an application embedded with the JidoshaLight library is simple and fast:

- 1) the '`lePlaca`' function must be replaced by the '`jidoshaLight_ANPR_fromFile`' function;
- 2) the '`struct JidoshaConfig`' must be replaced by the '`struct JidoshaLightConfig`';
- 3) the '`struct Recognition`' by the '`struct JidoshaLightRecognition`'.

The user must pay attention to the new JidoshaLight configuration fields, which must necessarily be filled with the correct values.

The following example shows how to achieve the same behavior as JIDOSHA with JidoshaLight:

- **JIDOSHA:**

```
#include <stdio.h>
#include "jidoshaCore.h"

int main(int argc, char* argv[])
{
    Rec recognition;
    JidoshaConfig config;
    config.typePlate = JIDOSHA_TYPE_PLATE_BOTH;
    config.timeout = 1000;
    readPlate(argv[1], &config, &rec);
}
```

```
printf("placa: %s\n", rec.placa);
return 0;
}
```

- **JidoshaLight**

```
#include <stdio.h>
#include "anpr/api/jidosha_light_api.h"

int main(int argc, char* argv[])
{
    JidoshaLightRecognition rec;
    JidoshaLight.Config config = {0};
    config.vehicleType         = JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH;
    config.processingMode     = JIDOSHA_LIGHT_MODE_ULTRA_SLOW;
    config.timeout             = 1000;
    config.countryCode        = JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL;
    config.maxLowProbabilityChars = 0;
    config.minProbPerChar     = 0.85;
    config.lowProbabilityChar  = '?';
    jidoshaLight_ANPR_fromFile(argv[1], &config, &rec);
    printf("placa: %s\n", rec.plate);
    return 0;
}
```

Remarks:

- The '*struct JidoshaLightConfig config*' is initialized with zero '*{0}*', ensuring that the fields '*int xRoi[4]*' and '*int yRoi[4]*' are zero and disabling the use of ROI.
- The processing mode '*JIDOSHA\_LIGHT\_MODE\_ULTRA\_SLOW*' is the one that most closely resembles the processing strategy used by the JIDOSHA library.

The SDK accompanies a more detailed application example.

## 7. JIDOSHA User APIs

For ease of use and migration to the JidoshaLight library, the JIDOSHA APIs are also made available through the '*libjidoshaCore.so*' and '*jidoshaCore.dll*' libraries'. It is possible to exchange the JIDOSHA library for these new files while maintaining the same behavior (with a few caveats; see [Special legacy API Builds](#). Files with the JIDOSHA interface are found inside the *jidoshapc* folder of the Windows or Linux SDK.

JIDOSHA's native Application Programming Interface (API) is written in C language, which allows its use from any language. The SDK also includes wrapper libraries to simplify library usage from .NET (C# and VB.NET), Java, and Delphi. These wrappers simply encapsulate calls to library functions, making any necessary conversion of parameters and results.

The entire API C is available through a single header file, *jidoshaCore.h*, the contents of which are presented below. A more detailed description is also presented.

The library can be used in two ways: via API 1 or API 2:

API 1, which was JIDOSHA's first API, is primarily motivated by ease of use. It is possible to read plates through a single function call (*'lePlaca'* or '*lePlacaFromMemory*', in the case of C language).

API 2 was created to provide greater flexibility in library configuration and image loading. For example, it is possible to set the minimum number of characters that must be read with good reliability for the plate to be considered valid. You can add new configuration parameters to API 2 without affecting existing library users (i.e., these users can update the JIDOSHA DLL/.so to a newer version, without having to recompile). In addition, API 2 allows the use of RAW type images, both grayscale and RGB/BGR. Compatibility with other formats can be added as needed.

We recommend API 1 for those who need to integrate JIDOSHA into their application as soon as possible, and API 2 for those who would like more control over the operation of the library.

## jidoshacore.h

```
#define JIDOSHA_TYPE_PLATE_CAR 1 /* recognize only non-run plates (other vehicles) */
#define JIDOSHA_TYPE_PLATE_MOTORCYCLE 2 /* only recognizes motorcycle plates */
#define JIDOSHA_TYPE_PLATE_BOTH 3 /* recognize any plate */

enum jidoshaError {
    JIDOSHA_SUCCESS = 0,
    JIDOSHA_ERROR_HARDKEY_NOT_FOUND,
    JIDOSHA_ERROR_HARDKEY_NOT_AUTHORIZED,
    JIDOSHA_ERROR_FILE_NOT_FOUND,
    JIDOSHA_ERROR_INVALID_IMAGE,
    JIDOSHA_ERROR_INVALID_IMAGE_TYPE,
    JIDOSHA_ERROR_INVALID_PROPERTY,
    JIDOSHA_ERROR_COUNTRY_NOT_SUPPORTED,
    JIDOSHA_ERROR_OTHER = 999,
};

/* OCR Parameters */
typedef struct JidoshaConfig
{
    int typePlate; /* indicates the type of plate the OCR should look for
                  use JIDOSHA_TYPE_PLATE_CAR,
                  JIDOSHA_TYPE_PLATE_MOTORCYCLE,
                  or JIDOSHA_TYPE_PLATE_BOTH */
    int timeout; /* timeout in milliseconds */
} JidoshaConfig;

/* OCR Result */
typedef struct Recognition
{
    char plate[8]; /* 7 character plate ending with 0, or empty string if plate not found */
    double probabilities[7]; /* values from 0.0 to 1.0 indicating reliability of the
    recognition of each character */
    int xText; /* xText and yText are the top left point */
    int yText; /* of the plate rectangle */
    int widthText; /* plate rectangle width */
    int heightText; /* plate rectangle height */
    int textColor; /* text color, 0 - dark, 1 - light */
    int isMotorcycle; /* 0 - non-motorcycle, 1 - motorcycle */
} Recognition;

/* API 1 *****/
```



```
/* Runs the OCR from a buffer containing an encoded image (JPG, BMP, etc.)
returns empty plate if the hardkey has not been found or is invalid */
int lePlacaFromMemory(const unsigned char* stream, int n, JidoshaConfig* config,
Recognition* rec);

/* Runs the OCR from a file whose name is provided
returns empty plate if the hardkey has not been found or is invalid */
int readPlate(const char* filename, JidoshaConfig* config, Recognition* rec);

/* Library version */
int getVersion(int* major, int* minor, int* release);

/* Hardkey serial number */
int getHardkeySerial(unsigned long* serial);

/* Hardkey status
state == 0 -> Unauthorized
state == 1 -> authorized
return == 0 -> hardkey found
return == 1 -> hardkey not found */
int getHardkeyState(int* state);

/* Demo hardkey time remaining
days== -1 and hours== -1: hardkey is not demonstration (infinite duration)
*/
int getHardkeyRemainingTime(int* days, int* hours);

/* API 2 *****/

/* API Default Configuration:
int typePlate          = 3 (JIDOSHA_TYPE_PLATE_BOTH)
int timeout            = 0
int minNumChars        = 7
int maxNumChars        = 7
int minCharWidth       = 1
int avgCharWidth       = 7
int maxCharWidth       = 40
int minCharHeight      = 9
int avgCharHeight      = 20
int maxCharHeight      = 60
double minPlateAngle   = -30.0
double avgPlateAngle   = 0.0
double maxPlateAngle   = 30.0
double avgPlateSlant   = 0.0
int adjustPerspective  = 0
int autoSlope          = 1
int autoSlant          = 1
double minProbPerCharacter = 0.8
char lowProbabilityChar = '*'
double excellentProb   = 0.95
int ocrModel           = 1
int checkSyntax        = 1
*/

/* Chained list of recognitions */
```

```
typedef struct ResultList
{
    struct ResultList* next;
    struct Recognition* recognition;
} ResultList;

/* Releases memory from a list of recognitions */
void jidoshaFreeResultList(ResultList* list);

typedef void JidoshaHandle; /* handle used in API2 */
typedef void JidoshaImage; /* handle for image allocated in API2 */

/* API2 handle initializes
in multithread processing, one handle per thread must be used */
JIDOSHACORE_API JidoshaHandle* jidoshaInit();

/* Finishes a previously allocated handle */
JIDOSHACORE_API int jidoshaDestroy(JidoshaHandle* handle);

/* Writes an integer type configuration property */
JIDOSHACORE_API int jidoshaSetIntProperty(JidoshaHandle* handle, const char* name, int
value);
/* Read an integer type configuration property */
JIDOSHACORE_API int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int*
value);

/* Writes a configuration property of type double */
JIDOSHACORE_API int jidoshaSetDoubleProperty(JidoshaHandle* handle, const char* name,
double value);
/* Read a double type configuration property */
JIDOSHACORE_API int jidoshaGetDoubleProperty(JidoshaHandle* handle, const char* name,
double* value);

/* Writes a char type configuration property */
JIDOSHACORE_API int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char
value);
/* Read a char type configuration property */
JIDOSHACORE_API int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char*
value);

/* Runs the OCR on a loaded image */
JIDOSHACORE_API int jidoshaFindFirst(JidoshaHandle* handle, JidoshaImage* image,
ResultList* list);

/* Runs the OCR into a loaded image to read from the second plate onwards.
The first plate should be read by jidoshaFindFirst. */
JIDOSHACORE_API int jidoshaFindNext(JidoshaHandle* handle, JidoshaImage* image, ResultList*
list);

/* Loads a jpg or bmp image from a file */
JIDOSHACORE_API int jidoshaLoadImage(const char* filename, JidoshaImage** img);

/* Loads a jpg, bmp or raw image (grayscale or RGB/BGR)
from an in-memory buffer */
JIDOSHACORE_API int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type,
int width, int height, JidoshaImage** img);
```

```

/* Releases memory from a uploaded image */
JIDOSHACORE_API int jidoshaFreeImage(JidoshaImage** img);

/* String to identify the library build */
JIDOSHACORE_API const char* jidoshaBuildInfo();

/* Number of authorized threads */
JIDOSHACORE_API int jidoshaNumThreads();

```

## API1 JIDOSHA C/C++

### Types

<b>struct JidoshaConfig</b>	
<b>Description</b>	The purpose of this structure is to configure the behavior of the library in the plate recognition call.
<b>Members</b>	<p><i>int typePlate</i>: indicates the type of plate that the OCR should look for, and should be one of the following values:</p> <ul style="list-style-type: none"> <li>JIDOSHA_TYPE_PLATE_CAR: Only license plates will be searched, where "car" means "non-motorcycle", i.e., it includes cars, trucks, buses etc.</li> <li>JIDOSHA_TYPE_PLATE_MOTORCYCLE: Only motorcycle plates will be searched.</li> <li>JIDOSHA_TYPE_PLATE_BOTH: Both motorcycle and non-motorbike plates will be searched.</li> </ul> <p><i>int timeout</i>: Indicates the maximum time that plate recognition should take, in milliseconds. A value of zero indicates that there is no timeout. A non-zero value helps keep the average processing time low. The value should be determined based on the resolution of the image and CPU used</p>

<b>struct Recognition</b>	
<b>Description</b>	The purpose of this structure is to store the result of the plate recognition, including: the characters of the plate, the reliability of each character, and the coordinates of the plate in the image.
<b>Members</b>	<p><i>char placa[8]</i>: 7-character plate ending with 0, or empty string if the plate was not found.</p> <p><i>double probabilities[7]</i>: values from 0.0 to 1.0 indicating the reliability, in the form of probability, of the recognition of each character.</p> <p><i>int xText</i> and <i>int yText</i>: coordinates of the point on the top left of the plate, if found.</p> <p><i>int widthText</i>: width of the plate rectangle.</p> <p><i>int heightText</i>: height of the plate rectangle.</p> <p><i>int textColor</i>: plate text color, 0 - dark, 1 - light.</p> <p><i>int isMotorcycle</i>: indicates if plate is motorcycle, 0 - non-motorcycle, 1 - motorcycle.</p>

### Methods

<b>readPlate</b>	
<b>Function Prototype</b>	<code>int readPlate(const char* filename, JidoshaConfig* config, Recognition* rec);</code>
<b>Description</b>	Recognize the plate and store it in a 'Recognition' object. The image should be passed as a parameter in the path format from where the image is located. If no plate is found, or if

	hardkey is not authorized or not found, the ' <i>Recognition</i> ' object will contain an empty string as the plate. The image file must be a bitmap, jpeg or png.
<b>Parameters</b>	<i>filename</i> : path to the image file. <i>config</i> : pointer to the struct ' <i>JidoshaConfig</i> ' with the configuration for the library. <i>rec</i> : pointer to the ' <i>Recognition</i> ' struct where the reading result will be stored.
<b>Return</b>	Error code: 0 (zero) in case of success, non-zero number otherwise.

### readPlateFromMemory

<b>Function Prototype</b>	<code>int readPlateFromMemory(const unsigned char* stream, int n, JidoshaConfig* config, Recognition*rec);</code>
<b>Description</b>	Recognize the plate and store it in a ' <i>Recognition</i> ' object. The image should be passed as a parameter in the byte array format, and the number of bytes given by the ' <i>n</i> ' parameter. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the ' <i>Recognition</i> ' object will contain an empty string as the plate. The image file must be a bitmap, jpeg or png.
<b>Parameters</b>	<i>stream</i> : array of bytes containing the image. <i>n</i> : size of the byte array. <i>config</i> : pointer to the struct ' <i>JidoshaConfig</i> ' with the configuration for the library. <i>rec</i> : pointer to the ' <i>Recognition</i> ' struct where the reading result will be stored.
<b>Return</b>	Error code: 0 (zero) in case of success, non-zero number otherwise.

### getVersion

<b>Function Prototype</b>	<code>int getVersion(int* major, int* minor, int* release);</code>
<b>Description</b>	Used to check the version of the library, in major.minor.release format.
<b>Parameters</b>	<i>major</i> : pointer to variable int where the major will be written. <i>minor</i> : pointer to variable int where the minor will be written. <i>release</i> : pointer to the int variable where the release will be written.
<b>Return</b>	Always returns 0 (zero).

### getHardkeySerial

<b>Function Prototype</b>	<code>int getHardkeySerial(unsigned long* serial);</code>
<b>Description</b>	Used to verify the serial number of the hardkey.
<b>Parameters</b>	<i>serial</i> : pointer to variable unsigned long where the hardkey serial number will be written.
<b>Return</b>	Returns 0 on success, 1 if hardkey is not found.

### getHardkeyState

<b>Function Prototype</b>	<code>int getHardkeyState(int* state);</code>
<b>Description</b>	Used to check the state of the hardkey. If state is equal to 0, hardkey is not authorized; if state is equal to 1, hardkey is authorized.
<b>Parameters</b>	<i>state</i> : pointer to variable int the hardkey state will be written.
<b>Return</b>	Returns 0 on success, 1 if hardkey is not found.

<b>getHardkeyRemainingTime</b>	
<b>Function Prototype</b>	<code>int getHardkeyRemainingTime(int* days, int* hours);</code>
<b>Description</b>	Used to check the remaining time for demo licenses. If days and hours are equal to -1 there is no time limit.
<b>Parameters</b>	<i>days</i> : pointer to variable int where the number of days remaining will be written. <i>hours</i> : pointer to variable int where the number of hours remaining will be written.
<b>Return</b>	Returns 0 on success, 1 if hardkey is not found.

## API2 JIDOSHA C/C++

### Types

<b>struct ResultList</b>	
<b>Description</b>	The purpose of this structure is to store the linked list with the processing results of the ' <i>jidosaFindFirst</i> ' and ' <i>jidosaFindNext</i> ' functions.
<b>Members</b>	<i>struct ResultList* next</i> : pointer to the next node in the list. NULL if the current node is the last. <i>struct Recognition* recognition</i> : pointer to the struct that contains a plate recognition result.

<b>typedef void JidoshaHandle</b>	
<b>Description</b>	Type used to represent the memory allocated to the configuration.

<b>typedef void JidoshaImage</b>	
<b>Description</b>	Type used to represent the memory allocated to an image.

### Methods

<b>jidosaFreeResultList</b>	
<b>Function Prototype</b>	<code>void jidoshaFreeResultList(ResultList* list);</code>
<b>Description</b>	Releases the memory allocated to the chained list of results.
<b>Parameters</b>	<i>list</i> : pointer to a struct ' <i>ResultList</i> '.
<b>Return</b>	It has no return.

<b>jidosaInit</b>	
<b>Function Prototype</b>	<code>JidoshaHandle* jidoshaInit();</code>
<b>Description</b>	Allocates memory to the library configuration. In the case of multithreaded use, each thread should call ' <i>jidosaInit</i> ' and use its own ' <i>JidoshaHandle</i> '.
<b>Parameters</b>	None.
<b>Return</b>	Returns a pointer to a ' <i>JidoshaHandle</i> ' that will be used in subsequent function calls.

<b>jidosaDestroy</b>	
<b>Function Prototype</b>	<code>int jidoshaDestroy(JidoshaHandle* handle);</code>
<b>Description</b>	Releases the memory allocated by the jidoshaInit function.
<b>Parameters</b>	<i>handle</i> : pointer to a variable 'JidoshaHandle'.
<b>Return</b>	JIDOSHA_SUCCESS.

<b>jidosaSetIntProperty</b>	
<b>Function Prototype</b>	<code>int jidoshaSetIntProperty(JidoshaHandle* handle, const char* name, int value);</code>
<b>Description</b>	Changes the value of a variable of the int type of the configuration.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle'. <i>name</i> : string that contains the name of the property to be changed. <i>value</i> : value that should be assigned to the property.
<b>Return</b>	JIDOSHA_SUCCESS if the variable value is changed, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type int.

<b>jidosaGetIntProperty</b>	
<b>Function Prototype</b>	<code>int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int* value);</code>
<b>Description</b>	Reads the value of a variable of type int from the configuration.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle'. <i>name</i> : string that contains the name of the property to be read. <i>value</i> : pointer to variable int where the property value will be written.
<b>Return</b>	JIDOSHA_SUCCESS if the variable value is read, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type int.

<b>jidosaSetDoubleProperty</b>	
<b>Function Prototype</b>	<code>int jidoshaSetDoubleProperty(JidoshaHandle* handle, const char* name, double value);</code>
<b>Description</b>	Changes the value of a configuration double variable.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle'. <i>name</i> : string that contains the name of the property to be changed. <i>value</i> : value that should be assigned to the property.
<b>Return</b>	JIDOSHA_SUCCESS if the variable value is changed, JIDOSHA_ERROR_INVALID_PROPERTY if the PROPERTY does not exist or is not of the double type.

<b>jidosaGetDoubleProperty</b>	
<b>Function Prototype</b>	<code>int jidoshaGetDoubleProperty(JidoshaHandle* handle, const char* name, double* value);</code>
<b>Description</b>	Reads the value of a configuration double variable.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle'. <i>name</i> : string that contains the name of the property to be read. <i>value</i> : pointer to double variable where the property value will be written.

<b>Return</b>	JIDOSHA_SUCCESS if the variable value is read, JIDOSHA_ERROR_INVALID_PROPERTY if the PROPERTY does not exist or is not of the double type.
---------------	--

### jidosaSetCharProperty

<b>Function Prototype</b>	<code>int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char value);</code>
<b>Description</b>	Changes the value of a variable of the int type of the configuration.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle'. <i>name</i> : string that contains the name of the property to be changed. <i>value</i> : value that should be assigned to the property.
<b>Return</b>	JIDOSHA_SUCCESS if the variable value is changed, JIDOSHA_ERROR_INVALID_PROPERTY if the PROPERTY does not exist or is not of the char type.

### jidosaGetCharProperty

<b>Function Prototype</b>	<code>int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char* value);</code>
<b>Description</b>	Reads the value of a configuration char variable.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle'. <i>name</i> : string that contains the name of the property to be read. <i>value</i> : pointer to the char variable where the property value will be written.
<b>Return</b>	JIDOSHA_SUCCESS if the variable value is read, JIDOSHA_ERROR_INVALID_PROPERTY if the PROPERTY does not exist or is not of the char type.

### jidosaFindFirst

<b>Function Prototype</b>	<code>int jidoshaFindFirst(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);</code>
<b>Description</b>	Recognises the plate and stores it in a 'Recognition' object found on the first node of the 'ResultList'. The image should be loaded using the 'jidosaLoadImage' or 'jidosaLoadImageFromMemory' functions. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the 'Recognition' object will contain an empty string as the plate. This function should only be called with an empty 'ResultList'.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle' that contains the library configuration. <i>image</i> : pointer to a 'JidoshaImage' that contains the image to be processed. <i>list</i> : pointer to a 'ResultList' where the processing result will be stored.
<b>Return</b>	JIDOSHA_SUCCESS if the image is processed, otherwise another value of the enum 'jidosaError'.

### jidosaFindNext

<b>Function Prototype</b>	<code>int jidoshaFindNext(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);</code>
<b>Description</b>	The purpose of this function is to allow the user to recognize multiple plates in the same image. The function recognises the plate and stores it in a 'Recognition' object that is on the last node of the 'ResultList'. The image should be loaded using the 'jidosaLoadImage' or 'jidosaLoadImageFromMemory' functions. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the 'Recognition' object will contain an empty string as the plate. This function should only be called with a 'ResultList' previously processed by the 'jidosaFindFirst' or 'jidosaFindNext' function.
<b>Parameters</b>	<i>handle</i> : pointer to a 'JidoshaHandle' that contains the library configuration. <i>image</i> : pointer to a 'JidoshaImage' that contains the image to be processed.

	<i>list</i> : pointer to a 'ResultList' where the processing result will be stored.
<b>Return</b>	JIDOSHA_SUCCESS if the image is processed, otherwise another value of the enum 'jidshaError'.

<b>jidoshaLoadImage</b>	
<b>Function Prototype</b>	<code>int jidoshaLoadImage(const char* filename, JidoshaImage** img);</code>
<b>Description</b>	Loads an image from a file and saves the reference as a JidoshaImage. The image file must be a bitmap, jpeg or png.
<b>Parameters</b>	<i>filename</i> : path to the image file. <i>img</i> : pointer-to-pointer to the 'JidoshaImage' struct where the image will be stored.
<b>Return</b>	JIDOSHA_SUCCESS if the image is loaded correctly, JIDOSHA_ERROR_FILE_NOT_FOUND if the file is not found or does not exist, JIDOSHA_ERROR_INVALID_IMAGE or JIDOSHA_ERROR_INVALID_IMAGE_TYPE if there are problems with the image load.

<b>jidoshaLoadImageFromMemory</b>	
<b>Function Prototype</b>	<code>int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type, int width, int height, JidoshaImage** img);</code>
<b>Description</b>	Loads an image from a byte array and saves the reference as a 'JidoshaImage'. The image must be in some structured format (bmp, jpg, png, etc.) or raw (Grayscale 8bit, RGB, or BGR).
<b>Parameters</b>	<i>buf</i> : array of bytes containing the image. <i>n</i> : array size in bytes. <i>type</i> : image type: structured types =0, GRAY8=1, RGB=2, BGR=3. <i>width</i> : image width, ignored if type==0. <i>height</i> : image height, ignored if type==0. <i>img</i> : pointer-to-pointer to a 'JidoshaImage' where the image will be stored.
<b>Return</b>	JIDOSHA_SUCCESS if the image is loaded correctly, JIDOSHA_ERROR_FILE_NOT_FOUND if the file is not found or does not exist, JIDOSHA_ERROR_INVALID_IMAGE or JIDOSHA_ERROR_INVALID_IMAGE_TYPE if there are problems with the image load.

<b>jidoshaFreeImage</b>	
<b>Function Prototype</b>	<code>int jidoshaFreeImage(JidoshaImage** img);</code>
<b>Description</b>	Releases memory allocated to store an image.
<b>Parameters</b>	<i>img</i> : pointer-to-pointer for a 'JidoshaImage' that will be unallocated.
<b>Return</b>	JIDOSHA_SUCCESS.

<b>jidoshaBuildInfo</b>	
<b>Function Prototype</b>	<code>const char* jidoshaBuildInfo();</code>
<b>Description</b>	Checks the build information of the library and is used to verify that the version being executed is the expected one.
<b>Parameters</b>	None.
<b>Return</b>	Constant string that has 12 or 13 characters representing BuildInfo plus a terminator ('\0').



<b>jidoshaNumThreads</b>	
<b>Function Prototype</b>	<code>int jidoshaNumThreads();</code>
<b>Description</b>	Checks the number of authorized threads in hardkey.
<b>Parameters</b>	None.
<b>Return</b>	Integer representing how many threads are allowed to simultaneously execute the OCR functions of the library. Returns 1 if the hardkey is not found.

## API 2 - Configuration

In this section we detail all the configuration parameters available in API 2. This is true for API C, Java, .NET, and Python.

<b><i>typePlate</i> Parameter</b>	
<b>Description</b>	It serves to restrict the type of license plate that should be recognized. It is mainly interesting to reduce processing time. In particular, when ' <i>tipoPlaca = JIDOSHA_TYIPE_PLATE_CAR</i> ', a faster plate location method can be used by the library. The valid values are:
<b>Name</b>	<i>tipoPlaca</i>
<b>Type</b>	int
<b>Default value</b>	JIDOSHA_TYPE_PLATE_BOTH
<b>Other values</b>	<ul style="list-style-type: none"> <li>• JIDOSHA_TYPE_PLATE_CAR' == 1</li> <li>• JIDOSHA_TYPE_PLATE_MOTORCYCLE' == 2</li> <li>• JIDOSHA_TYPE_PLATE_BOTH' == 3</li> </ul>

<b><i>timeout</i> parameter</b>	
<b>Description</b>	After ' <i>timeout</i> ' milliseconds from the start of processing an image the search of the plate will be terminated and the best plate found will be returned. If ' <i>timeout</i> ' is zero, there is no timeout. It is recommended to use a non-zero ' <i>timeout</i> ' when the application requires images to be processed quickly (with low latency) or when the CPU load is too high.
<b>Name</b>	<i>timeout</i>
<b>Type</b>	int
<b>Default value</b>	0

<b><i>minNumChars</i> Parameter</b>	
<b>Description</b>	Indicates the minimum number of characters a plate must have. If the version in use of the library has multiple plate syntaxes enabled (for example, multi-country plates), this parameter is ignored and ' <i>numAllowedBadChars</i> ' should be used instead.
<b>Name</b>	<i>minNumChars</i>
<b>Type</b>	int
<b>Default value</b>	7

<b>numAllowedBadChars Parameter</b>	
<b>Description</b>	Indicates the maximum number of missing characters a plate can have, this parameter is used when it is desired that partially recognized plates be returned.
<b>Name</b>	<i>numAllowedBadChars</i>
<b>Type</b>	int
<b>Default value</b>	0

<b>MaxNumChars Parameter</b>	
<b>Description</b>	Indicates the maximum number of characters a plate should have. This parameter is currently ignored.
<b>Name</b>	<i>maxNumChars</i>
<b>Type</b>	int
<b>Default value</b>	7

<b>minCharWidth Parameter</b>	
<b>Description</b>	Minimum width a character should have, in pixels.
<b>Name</b>	<i>minCharWidth</i>
<b>Type</b>	int
<b>Default value</b>	1

<b>avgCharWidth Parameter</b>	
<b>Description</b>	Expected average width of a character, in pixels. This parameter is currently not used.
<b>Name</b>	<i>avgCharWidth</i>
<b>Type</b>	int
<b>Default value</b>	1

<b>maxCharWidth Parameter</b>	
<b>Description</b>	Maximum width a character should have, in pixels.
<b>Name</b>	<i>maxCharWidth</i>
<b>Type</b>	int
<b>Default value</b>	7

<b>minCharHeight Parameter</b>	
<b>Description</b>	Minimum height a character should have, in pixels.
<b>Name</b>	<i>minCharHeight</i>
<b>Type</b>	int
<b>Default value</b>	9

<b>avgCharHeight Parameter</b>	
<b>Description</b>	Expected average height of a character, in pixels. This parameter can be used when the plates are too large. When ' <i>avgCharHeight</i> > 30', the image will be internally reduced before it is processed. The minimum and maximum character size limits will be adjusted according to the resizing factor.
<b>Name</b>	<i>avgCharHeight</i>
<b>Type</b>	int
<b>Default value</b>	20

<b>maxCharHeight Parameter</b>	
<b>Description</b>	Maximum height of one character, in pixels.
<b>Name</b>	<i>maxCharHeight</i>
<b>Type</b>	int
<b>Default value</b>	60

<b>ocrModel Parameter</b>	
<b>Description</b>	Defines the OCR model to be used in character recognition. This parameter exists to allow you to easily switch the OCR model to models from previous versions of the library, without the need to recompile the user application or switch the library. Do not use values other than the default, except when recommended by the Pumatronix Equipamentos Eletrônicos' support team.
<b>Name</b>	<i>ocrModel</i>
<b>Type</b>	int
<b>Default value</b>	1

<b>checkSyntax Parameter</b>	
<b>Description</b>	When ' <i>checkSyntax</i> =1' the library applies an additional processing step to verify that the recognized characters have the expected syntax (letter or number), which reduces the incidence of false recognitions (texts that are not plates). Note: Even when ' <i>checkSyntax</i> =0', the library will never return a recognition with a syntax different from that defined. For example, for Brazilian plates, the returned plate will always have 3 letters followed by 4 numbers. However, a non-plate text, such as "SCHOOL", can be confused with a plate, which would result in a recognition as "SCH0148". The syntax is according to a Brazilian plate, although it is not a plate. Using ' <i>checkSyntax</i> =1' can help discard false recognitions as in the example.
<b>Name</b>	<i>checkSyntax</i>
<b>Type</b>	int
<b>Default value</b>	1

<b>minPlateAngle Parameter</b>	
<b>Description</b>	Minimum slope angle in degrees allowed for a plate. For more details, refer to the perspective configuration section of the image.
<b>Name</b>	<i>minPlateAngle</i>

<b>Type</b>	double
<b>Default value</b>	-30.0

#### maxPlateAngle Parameter

<b>Description</b>	Maximum slope angle in degrees allowed for a plate. For more details, refer to the perspective configuration section of the image.
<b>Name</b>	<i>maxPlateAngle</i>
<b>Type</b>	double
<b>Default value</b>	30.0

#### minProbPerCharacter Parameter

<b>Description</b>	<p>Minimum probability (reliability) required in the recognition of each character. It is extremely important for the proper functioning of OCR, and it is not recommended to change the default configuration. However, in specific cases it may be interesting to adjust it.</p> <p>If '<i>minProbPerCharacter</i>' is smaller than the default, the number of plates that are not recognized will reduce, but in contrast the number of plates with some wrong character may increase.</p> <p>If '<i>minProbPerCharacter</i>' is greater than the default, the number of plates that are not recognized may increase, but the number of errors will be smaller.</p>
<b>Name</b>	<i>minProbPerCharacter</i>
<b>Type</b>	double
<b>Default value</b>	0.8

#### excellentProb Parameter

<b>Description</b>	<p>This parameter exists to reduce the average processing time. If all recognized characters have a probability greater than or equal to '<i>excellentProb</i>', the recognition will be considered excellent and returned to the user immediately, without further processing. Otherwise, processing will continue until one of the following conditions is met: excellent recognition is found; the timeout is reached; or there are no further processing steps to do.</p> <p>Higher values of '<i>excellentProb</i>' result in higher recognition rates and lower error rates (confusion between characters), but with longer processing time.</p> <p>Lower values of '<i>excellentProb</i>' result in lower recognition rates and higher error rates (confusion between characters), but with shorter processing time.</p>
<b>Name</b>	<i>excellentProb</i>
<b>Type</b>	double
<b>Default value</b>	0.95

#### lowProbabilityChar Parameter

<b>Description</b>	Replacement character to be used when a plate character is recognized with probability less than ' <i>minProbPerCharacter</i> '. It will take effect only if ' <i>minNumChars</i> ' is less than ' <i>maxNumChars</i> '.
--------------------	--

	For example, if 'lowProbabilityChar='-' and 'minNumChars =6', the plate "ABC1234" will be returned as "A-C1234" if the probability of the second character is less than 'minProbPerCharacter'.
<b>Name</b>	<i>lowProbabilityChar</i>
<b>Type</b>	char
<b>Default value</b>	'*'

<i>country</i> Parameter	
<b>Description</b>	Replacement character to be used when a plate character is recognized with probability less than 'minProbPerCharacter'. It will take effect only if 'minNumChars' is less than 'maxNumChars'. For example, if 'lowProbabilityChar='-' and 'minNumChars =6', the plate "ABC1234" will be returned as "A-C1234" if the probability of the second character is less than 'minProbPerCharacter'.
<b>Name</b>	<i>country</i>
<b>Type</b>	int
<b>Default value</b>	76

## API 2 - Image Perspective Configuration

In general, it is recommended that the installation of the camera to capture vehicular plates be done so that the plates are aligned with the horizontal and vertical axes of the image. However, in some situations this is not possible, and it ends up obtaining plates inclined in relation to the axes of the image, which can impair the recognition of the plates. In these cases, the library can be informed of the perspective of the plate. The library will then perform a perspective correction to maximize the plate recognition index.

In the case of multi-camera equipment, it is recommended to create one API 2 '*handle*' per camera (via the '*jidoshaInit*' function) and configure the perspective parameters individually for each '*handle*'.

The parameters '*avgPlateAngle*', '*avgPlateSlant*' and '*adjustPerspective*' are used to inform the plate perspective in the image (horizontal and vertical slope) and correct it. Horizontal slope ('*avgPlateAngle*') and vertical slope ('*avgPlateSlant*') shall be measured on typical installation images.

In addition to the manual perspective configuration, it is also possible to enable algorithms in the library that seek to automatically correct the perspective. See the '*autoSlope*' and '*autoSlant*' parameters for more details.



Figure 11- How to calculate avgPlateAngle and avgPlateSlant values

avgPlateAngle Parameter	
<b>Description</b>	Average horizontal slope angle in degrees expected for a plate. It is used to make perspective adjustment of the image. It will only take effect if 'adjustPerspective' is non-zero. The angle must be measured according to the convention in the image above.
<b>Name</b>	avgPlateAngle
<b>Type</b>	double
<b>Default value</b>	0.0

avgPlateSlant Parameter	
<b>Description</b>	Average vertical slope angle in degrees expected for a plate. It is used to make perspective adjustment of the image. It will only take effect if 'adjustPerspective' is non-zero. The angle must be measured according to the convention in the image above.
<b>Name</b>	avgPlateSlant
<b>Type</b>	double
<b>Default value</b>	0.0

adjustPerspective Parameter	
<b>Description</b>	'adjustPerspective=1' enables perspective adjustment configured through 'avgPlateAngle' and 'avgPlateSlant'. 'adjustPerspective=0' disables perspective adjustment ('avgPlateAngle' and 'avgPlateSlant' are ignored).

<b>Name</b>	<i>adjustPerspective</i>
<b>Type</b>	int
<b>Default value</b>	0

<i>autoSlope</i> Parameter	
----------------------------	--

<b>Description</b>	' <i>autoSlope=1</i> ' enables automatic adjustment of the plate's horizontal slope. If used in conjunction with the manual perspective adjustment (' <i>avgPlateAngle</i> ' when ' <i>adjustPerspective=1</i> '), the manual adjustment will be applied before the automatic adjustment algorithm. ' <i>autoSlope=0</i> ' disables automatic adjustment of the plate's horizontal slope.
<b>Name</b>	<i>autoSlope</i>
<b>Type</b>	int
<b>Default value</b>	1

<i>autoSlant</i> Parameter	
----------------------------	--

<b>Description</b>	' <i>autoSlant=1</i> ' enables automatic adjustment of the vertical slope of the plate. If used in conjunction with the manual perspective adjustment (' <i>avgPlateSlant</i> ' when ' <i>adjustPerspective=1</i> '), the manual adjustment will be applied before the automatic adjustment algorithm. ' <i>autoSlant=0</i> ' disables the automatic adjustment of the vertical slope of the plate.
<b>Name</b>	<i>autoSlant</i>
<b>Type</b>	int
<b>Default value</b>	1

## API JIDOSHA C# / VB.NET

The library's .NET API features three overloaded functions, which facilitate plate recognition from three sources: a byte array containing the encoded image (JPG or bmp), an object of type '*Image*', or a file name. All require as a parameter a '*JidoshaConfig*' object that serves to configure the behavior of the library.

### API 1

#### Methods

<b>recognizesPlate 1</b>	
<b>Function Prototype</b>	<code>Recognition recognizesPlate(byte[] array, JidoshaConfig config)</code>
<b>Description</b>	Returns a ' <i>Recognition</i> ' object that represents the recognition result of the plate. The image (JPG, bmp, etc.) should be passed as an array of bytes.
<b>Return</b>	' <i>Recognition</i> ' object containing the string representing the license plate of the vehicle, an array of doubles containing the probabilities of the characters, the coordinates of the text of the license plate, the color of the text (dark or light), and a field indicating whether the license

	plate is a motorcycle. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the 'Recognition' object will contain an empty string as the plate.
--	---

### recognizesPlate 2

<b>Function Prototype</b>	<code>Recognition recognizesPlate(Image image, JidoshaConfig config)</code>
<b>Description</b>	Returns a 'Recognition' object that represents the recognition result of the plate. The image should be passed as a parameter in the form of an 'Image' object.
<b>Return</b>	'Recognition' object containing the string representing the license plate of the vehicle, an array of doubles containing the probabilities of the characters, the coordinates of the text of the license plate, the color of the text (dark or light), and a field indicating whether the license plate is a motorcycle. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the 'Recognition' object will contain an empty string as the plate.

### recognizesPlate 3

<b>Function Prototype</b>	<code>Recognition recognizesPlate(string filename, JidoshaConfig config)</code>
<b>Description</b>	Returns a 'Recognition' object that represents the recognition result of the plate. The image should be passed as a parameter in the path format from where the image is located.
<b>Return</b>	'Recognition' object containing the string representing the license plate of the vehicle, an array of doubles containing the probabilities of the characters, the coordinates of the text of the license plate, the color of the text (dark or light), and a field indicating whether the license plate is a motorcycle. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the 'Recognition' object will contain an empty string as the plate.

### getVersionString

<b>Function Prototype</b>	<code>String getVersionString()</code>
<b>Description</b>	Used to check the version of the library, in major.minor.release format.
<b>Return</b>	Returns a string formatted with the version.

### getHardkeySerial

<b>Function Prototype</b>	<code>int getHardkeySerial()</code>
<b>Description</b>	Used to verify the serial number of the hardkey.
<b>Return</b>	Returns an int containing the serial number of the hardkey.

### getHardkeyState

<b>Function Prototype</b>	<code>int getHardkeyState()</code>
<b>Description</b>	Used to check the state of the hardkey. If state is equal to 0, hardkey is not authorized; if state is equal to 1, hardkey is authorized.
<b>Return</b>	Returns the state of the hardkey (0 or 1, as described above).



## JIDOSHA C# / VB.NET API Examples

## Example C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

using JidoshaNET;

namespace JidoshaSample
{
    class JidoshaSample
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Jidosha build {0}", Jidosha.jidoshaBuildInfo());
            Console.WriteLine("Hardkey serial {0}", Jidosha.getHardKeySerial());
            Console.WriteLine("Hardkey {0}", Jidosha.getHardKeyState() == 1 ? "authorized"
: "unauthorized");

            if (args.Length < 1)
            {
                Console.WriteLine("use: jidoshaNETSample image");
                Console.WriteLine("Press Enter to exit");
                Console.ReadLine();
                return;
            }

            // Upload image
            string filename = args[0];
            Image image = Image.FromFile(filename);
            System.IO.MemoryStream stream = new System.IO.MemoryStream();
            image.Save(stream, image.RawFormat);
            byte[] array = stream.ToArray();
            stream.Close();
            stream.Dispose();

            // Sample API1
            JidoshaConfig cfg = new JidoshaConfig();
            cfg.timeout = 0;
            cfg.typePlate = TypePlate.BOTH;
            Recognition r = Jidosha.recognizesPlate(filename, cfg);
            System.Console.WriteLine("recognizePlate: {0}", r.plate);
            r = Jidosha.recognizesPlate(array, cfg);
            System.Console.WriteLine("recognizePlateFromMemory: {0}", r.plate);

            // Sample API2

            // Initializes
            IntPtr JidoshaHandle = Jidosha.jidoshaInit();

            // SetProperty
            Jidosha.jidoshaSetIntProperty(JidoshaHandle, "avgCharHeight", 20);
            Jidosha.jidoshaSetIntProperty(JidoshaHandle, "minNumChars", 6);
```

```
Jidosha.jidoshaSetDoubleProperty(JidoshaHandle, "minProbPerCharacter", 0.7);
Jidosha.jidoshaSetCharProperty(JidoshaHandle, "lowProbabilityChar", '_');

// GetProperty
int maxCharHeight = 0;
double minProb = 0;
maxCharHeight = Jidosha.jidoshaGetIntProperty(JidoshaHandle, "avgCharHeight");
minProb = Jidosha.jidoshaGetDoubleProperty(JidoshaHandle,
"minProbPerCharacter");

Console.WriteLine("Average height: {0}", maxCharHeight);
Console.WriteLine("Minimum Probability: {0}", minProb);

// Upload an image
IntPtr JidoshaImg = Jidosha.jidoshaLoadImage(array, 0, 0, 0);

// Recognizes license plate
ResultList resultList = new ResultList();
Jidosha.jidoshaFindFirst(JidoshaHandle, JidoshaImg, ref resultList);
while (resultList.recognition[resultList.recognition.Count - 1].plate != "")
{
    Jidosha.jidoshaFindNext(JidoshaHandle, JidoshaImg, ref resultList);
}

// Prints the result
foreach (Recognition rec in resultList.recognition)
{
    Console.WriteLine("Plate: {0}", rec.plate);
    Console.WriteLine("Probs:");
    foreach (double d in rec.probabilities)
        Console.WriteLine(" {0},", d);
    Console.WriteLine("");
}

// Delete the list of acknowledgments
Jidosha.jidoshaFreeResultList(resultList);

// Releases the image
Jidosha.jidoshaFreeImage(JidoshaImg);

// Release the jidosha handle
Jidosha.jidoshaDestroy(JidoshaHandle);

Console.WriteLine("Press Enter to exit");
Console.ReadLine();
}
}
}
```

### Example VB.NET

```
Imports JidoshaNET
```

```
Module Module1
```

```
    Sub Main()
```

```
        Dim args() As String = Environment.GetCommandLineArgs()
```

```

Dim filename As String = args(1)
Dim config As JidoshaConfig = New JidoshaConfig()
config.typePlate = TypePlate.BOTH
config.timeout = 1000
Dim rec As Recognition = Jidosha.recognizesPlate(filename, config)
Console.WriteLine("plate: " + rec.plate)
End Sub

End Module

```

## API JIDOSHA Delphi

### API 1

#### Methods

recognisesPlate	
<b>Function Prototype</b>	<code>function recognizesPlate(filename: String; config: JidoshaConfig) : Recognition;</code>
<b>Description</b>	Returns a ' <i>Recognition</i> ' object that represents the recognition result of the plate. The image should be passed as a parameter in the path format from where the image is located.
<b>Return</b>	' <i>Recognition</i> ' object containing the string representing the license plate of the vehicle, an array of doubles containing the probabilities of the characters, the coordinates of the text of the license plate, the color of the text (dark or light), and a field indicating whether the license plate is a motorcycle. If no plate is found, or if hardkey is not authorized or not found, the ' <i>Recognition</i> ' object will contain an empty string as the plate.

recognizesFromMemoryPlate	
<b>Function Prototype</b>	<code>function recognizesPlateFromMemory(byteArray: array of byte; config: JidoshaConfig) : Recognition;</code>
<b>Description</b>	Returns a ' <i>Recognition</i> ' object that represents the recognition result of the plate. The image (JPG, bmp, etc.) should be passed as an array of bytes.
<b>Return</b>	' <i>Recognition</i> ' object containing the string representing the license plate of the vehicle, an array of doubles containing the probabilities of the characters, the coordinates of the text of the license plate, the color of the text (dark or light), and a field indicating whether the license plate is a motorcycle. If no plate is found, or if hardkey is not authorized or not found, the ' <i>Recognition</i> ' object will contain an empty string as the plate.

### Example JIDOSHA Delphi API

Note: This example is for Delphi 2007. In newer versions of Delphi, you may need to convert the filepath string to AnsiString before moving to the C library. You may also need to convert the AnsiString plate string to Unicode.

```

program JidoshaDelphiSample;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  jidoshaDelphi in 'jidoshaDelphi.pas';

var

```

```

filename: String;
rec: rec: Recognition;
config: JidoshaConfig;
begin
  if ParamCount < 1
  then begin
    Writeln('uso: jidoshaDelphiSample.exe image.jpg');
    Exit;
  end;

  filename := ParamStr(1);
  Writeln(filename);

  config.typePlate := JIDOSHA_TYPE_PLATE_BOTH;
  config.timeout := 1000;
  rec := recognizesPlate(filename, config);
  Writeln('plate: ', rec.plate);
end.

```

## API JIDOSHA Java

### API 1

#### Methods

<b>recognisesPlate</b>	
<b>Function Prototype</b>	<code>public static native Recognition recognizesPlate(String filename, JidoshaConfig config);</code>
<b>Description</b>	Returns a ' <i>Recognition</i> ' object that represents the recognition result of the plate. The image should be passed as a parameter in the path format from where the image is located.
<b>Parameters</b>	' <i>Recognition</i> ' object containing the string representing the license plate of the vehicle, an array of doubles containing the probabilities of the characters, the coordinates of the text of the license plate, the color of the text (dark or light), and a field indicating whether the license plate is a motorcycle. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the ' <i>Recognition</i> ' object will contain an empty string as the plate.

<b>recognizesFromMemoryPlate</b>	
<b>Function Prototype</b>	<code>public static native Recognition recognizesFromMemoryPlate(byte[] buf, JidoshaConfig config);</code>
<b>Description</b>	Returns a ' <i>Recognition</i> ' object that represents the recognition result of the plate. This object contains the plate string and the probability (reliability) of each recognized character. The image should be passed as a parameter in the byte array format.
<b>Parameters</b>	' <i>Recognition</i> ' object containing the string representing the license plate of the vehicle, an array of doubles containing the probabilities of the characters, the coordinates of the text of the license plate, the color of the text (dark or light), and a field indicating whether the license plate is a motorcycle. If no plate is found, or if <i>hardkey</i> is not authorized or not found, the ' <i>Recognition</i> ' object will contain an empty string as the plate.

### JIDOSHA Java API Example

```

import br.com.gaussian.jidosha.Jidosha;
import br.com.gaussian.jidosha.JidoshaConfig;
import br.com.gaussian.jidosha.Recognition;

```

```

class JidoshaSample {
    public static void main(String args[]) throws java.io.IOException {
        JidoshaConfig config = new JidoshaConfig(JidoshaConfig.JIDOSHA_TIPO_PLACA_AMBOS,
0);
        for (int i=0; i < args.length; i++) {
            System.out.println(args[i]);
            Recognition rec = Jidosha.recognizesPlate(args[i], config);
            System.out.println("plate: " + rec.plate);
        }
    }
}

```

## Legacy API Special Builds

For several reasons, the JIDOSHA library had different types of builds for the same version number, which are generally not compatible with each other. The build can be verified by returning the '*jidoshaBuildInfo*' function. The buildInfo string has the following format: "hash\_build", where "hash" is the commit hash, and "build" is a string denoting the build type.

Until v3.4.0 JidoshaLight is only compatible with JIDOSHA's build '*std*', which is the default build. As of v3.5.0, JidoshaLight is also compatible with build '*charpos*' ("character positions"), as long as there is a key in the registry or environment variable, as detailed below. The only difference from the '*std*' version to the '*charpos*' version consists of four additional fields in the '*Recognition*' struct in the '*jidoshaCore.h*' header, which contains the coordinates of the plate characters when the recognition is successful. This difference in the API makes the '*std*' and '*charpos*' builds incompatible (an executable compiled for one of these builds cannot be used with another).



**Note: Compatibility mode for build 'charpos' is only supported in the C language API.**

For reference, the structs of the '*std*' and '*charpos*' builds are listed below.

- **Build std**

```

typedef struct Recognition
{
char plate[7+1];
    double probabilities[7];
    int xText;
    int yText;
    int widthText;
    int heightText;
    int textColor;
    int isMotorcycle;
} Recognition;

```

- **Build charpos**

```

typedef struct Recognition
{
char plate[7+1];
    double probabilities[7];
    int xText;
    int yText;
    int widthText;
    int heightText;
} Recognition;

```

```
int xChar[7];
int yChar[7];
int widthChar[7];
int heightChar[7];
int textColor;
int isMotorcycle;
} Recognition;
```

To enable build '*charpos*' compatibility mode in **Windows**, it is necessary to create a key in the Windows registry, in '*HKLM\SOFTWARE\PUMATRONIX*', named '*JL\_LEGACY\_API\_TYPE*', of type '*REG\_SZ*', and value '*charpos*'. Any other value will cause JidoshaLight to revert to default behavior (build '*std*' compatibility). Instead of registering, you can use an environment variable, named '*JL\_LEGACY\_API\_TYPE*' and value '*charpos*'.

The key in the registry can be created with the following command at the prompt (Administrator credentials are required):

```
REG ADD HKLM\SOFTWARE\PUMATRONIX /v JL_LEGACY_API_TYPE /t REG_SZ /d charpos /f
```

To turn off the build '*charpos*' compatibility mode, change the value of the variable to an empty string, or simply delete the key:

```
REG DELETE HKLM\SOFTWARE\PUMATRONIX /v JL_LEGACY_API_TYPE
```

To enable build '*charpos*' compatibility mode on **Linux**, you need to create an environment variable, named '*JL\_LEGACY\_API\_TYPE*' and value '*charpos*'. Any other value will cause JidoshaLight to revert to default behavior (build '*std*' compatibility).

Remarks:

- If build '*charpos*' compatibility mode is enabled ('*JL\_LEGACY\_API\_TYPE= charpos*'), but the user code is mistakenly using the build '*std*' '*Recognition*' struct, invalid memory access or silent data corruption may occur.
- It is recommended to migrate to the JidoshaLight API as soon as possible.



[www.pumatronix.com](http://www.pumatronix.com)

