JidoshaLight

User Manual

Automatic License Plate Recognition Software Library

Release: v3.20.0
Date: 29/09/2021

Summary

- **Change history**
- 1. Overview
  - 1.1. General Conditions
  - 1.2. Software license
- 2. Introduction
  - 2.1. Objective
  - 2.2. Supported Plates
  - 2.3. JidoshaLight SDK Structure
    - 2.3.1. Supported APIs
- 3. JidoshaLight Linux
  - 3.1. Use conditions
  - 3.2. Software architecture
  - 3.3. Restrictions
  - 3.4. Installation
    - 3.4.1. Hardkey permissions configuration
    - 3.4.2. Environment variables configuration
    - 3.4.2.1 Log and audit system
    - 3.4.3. Preferences file configuration
  - 3.5. Sample applications
- 4. JidoshaLight Windows
  - 4.1. Use conditions
  - 4.3. Installation
  - 4.4. Environment variables configuration
    - 4.4.1 Log and audit system
  - 4.5 Preferences file configuration
  - 4.6. Sample applications
- 5. JidoshaLight Linux/FPGA
  - 5.1. Use conditions
  - 5.2. Software architecture
  - 5.3. Restrictions
  - 5.4. Installation
    - 5.4.1. License configuration
    - 5.4.2. Environment variables configuration
    - 5.4.3. Linux kernel configuration
  - 5.5. Sample applications
- 6. JidoshaLight Android™
  - 6.1. Use conditions
  - 6.2. Software architecture
  - 6.3. Restrictions
  - 6.4. Installation
    - 6.4.1 Licensing
    - 6.4.2 Permissions
  - 6.5. Sample application
    - 6.5.1 Package `br.gaussian.io`
    - 6.5.2 Package `br.gaussian.jidoshalight`
    - 6.5.3 Package `br.gaussian.jidoshalight.camera`
    - 6.5.4 Package `br.gaussian.jidoshalight.sample`
- 7. User APIs
  - 7.1. JidoshaLight C/C++ API
    - 7.1.1. JidoshaLight C/C++ API (Local)
    - 7.1.2. JidoshaLight C/C++ (Synchronous Remote)
    - 7.1.3. JidoshaLight C/C++ API (Asynchronous Remote)
    - 7.1.4. JidoshaLight C/C++ API (Server)
  - 7.2. JidoshaLight Java API
    - 7.2.1. JidoshaLight Java API (Local)
    - 7.2.2. JidoshaLight Java API (Asynchronous Remote)
    - 7.2.3. JidoshaLight Java API (Server)
    - 7.2.4. JidoshaLight Java API (IO/Mjpeg)

- 7.3. Migration Guide - Jidosha C/C++ API 1

- 8. JIDOSHA user APIs
  - 8.1.1. JIDOSHA C/C++ API 1
  - 8.1.2. JIDOSHA C/C++ API 2
  - 8.2.1. JIDOSHA C# / VB.NET API
  - 8.3.1. JIDOSHA Delphi API
  - 8.4.1. JIDOSHA Java API
  - 8.5.1. Special legacy API builds
- 9. Known limitations

Change history

Date	Version	Revision
22/06/2017	2.3.6	• Initial Version
29/06/2017	2.3.8	• Changed architecture diagram for Android platform • Added new functions to Java API
03/10/2017	2.3.10	• Changed API to support maximum and minimum character size configuration
05/12/2017	2.4.4	• Added multiple recognitions feature to the API
05/12/2017	2.4.6	• Added new error codes to the API • Added new configuration fields to struct JidoshaLightConfig • Tutorial for ROI selection in Android app
26/03/2018	2.6.0	• Added new country codes NETHERLANDS and FRANCE • Added new error code INVALD_IMAGE_SIZE • Corrected description of some error codes
06/06/2018	2.7.0	• Added JIDOSHA API wrapper • Initial support for Mercosul-format Brazilian plates • Added automatic perspective correction
08/06/2018	2.8.0	• Improved Brazilian plate recognition for PC platforms • Added VersionInfo to Windows DLL • Corrected typos
12/06/2018	2.8.2	• Bug fixes in DSP version
14/06/2018	2.8.4	• Bug fixes in DSP version
04/07/2018	2.9.0	• Improved Brazilian plate recognition
29/08/2018	3.0.0	• Improved support for Mercosul-format Brazilian plates
06/09/2018	3.1.0	• Improved partial Brazilian plate recognition
04/10/2018	3.2.0	• Official support for Mercosul-format Brazilian plates • Adopted GLIBC 2.12 by default • Added Jidosha.jar to SDK • Added Windows service example and installation script • Changed preferences file search mechanism • Changed log configuration file search mechanism • Fixed bug when loading color RAW images
19/10/2018	3.3.0	• Default log behavior changed
04/12/2018	3.4.0	• Reduced false recognitions of Brazilian plates.
07/02/2019	3.4.1	• Support for legacy licenses of type Evasao, Movel, and Vigia+
25/01/2019	3.5.0	• Added compatibility mode with legacy "charpos" builds • Partial correction of error when using JidoshaLight concurrently with container or rail OCR libraries • Fixed behavior when hardkey is removed and replugged • Fixed error in return of function getState in the legacy API • Removed wrong #ifdefs from C# samples
18/02/2019	3.5.2	• Made hardkey reading more robust

27/03/2019	3.6.0	• Added Chile version for ITSCAM • Improved Argentina Mercosul plate recognition • Improved Uruguay Mercosul plate recognition
05/04/2019	3.7.0	• Added new localization processing modes • Fixed error return code when server is not ready
18/04/2019	3.8.0	• Initial support for Colombian plates
14/05/2019	3.8.1	• Fixed JidoshaLightServer behavior when no license source is available
26/08/2019	3.9.0	• Fixed issue in ITSCAM version when a ROI is used • Improved Argentina plate recognition • Reduced processing time when using jidoshapc wrapper
30/09/2019	3.10.0	• Improved Mexico plate recognition • Improved Conesul plate recognition • Fixed jidohapc jar package
15/01/2020	3.10.1	• Fixed JIDOSHA API symbol export error
26/12/2019	3.11.0	• Improved Brazilian plate recognition in PC platform • Dropped Windows XP support • Reduced processing time in ARM platforms
27/01/2020	3.11.1	• Fixed freeze when using Java wrapper on Windows 32 bits • Added Python wrapper
10/02/2020	3.12.0	• Improved Chile plate recognition • Added Chile to Conesul group • Added experimental Peru plate support
02/03/2020	3.13.0	• Improved plate recognition in ARM architectures • Added hazard identification plate support
01/04/2020	3.14.0	• Added alternative core support • Added ITSCAM600 architecture support
04/06/2020	3.14.1	• Fixed link error in Linux PC architectures • Fixed Panama country code • Added missing country codes to wrappers
24/06/2020	3.14.2	• Fixed ITSCAM600 hardware acceleration • Improved license plate recognition in shadow condition in ITSCAM600 architecture • Fixed segmentation fault in FPGA architectures
28/07/2020	3.15.0	• California (USA) license plate support • DNA base license support on ITSCAM600 • Android 10 support
14/09/2020	3.16.0	• Enhanced motorcycle recognition performance on PC • Enhanced overall recognition performance on AARCH64 and ITSCAM600
09/11/2020	3.17.0	• Enhanced recognition performance of Peru license plates • Added ARMv8 support to Android SDK (API 26) • Minor documentation fixes
04/03/2021	3.18.0	• Improved Argentina license plate recognition performance • Improved Paraguay license plate recognition performance
24/05/2021	3.19.0	• Improved Brazil license plate recognition performance • Updated SDK documentation

29/09/2021	3.20.0	• Improved Argentina license plate recognition performance • Improved Brazilian license plate recognition performance • Improved processing time with Brazilian license plates • Bug fix for displaced license plate coordinates with RoI enabled

1. Overview

1.1. General Conditions

The data and information contained in this document may not be altered without written permission by Pumatronix. No part of this document may be reproduced or transmitted for whatever purpose, whether by electronic or physical means.

Copyright © Pumatronix Equipamentos Eletrônicos. All rights reserved.

1.2. Software license

The software and this document are protected by author rights. On installing the software, you are agreeing with the conditions of the license contract.

2. Introduction

This document, **JidoshaLight - User Manual**, details the Application Programming Interface of the automatic license plate recognition library, namely JidoshaLight.

The JidoshaLight library is compatible with PCs (x86/x86_64) with Windows™ or Linux and ARM™ processors running Android™ or Linux OS.

2.1. Objective

The main feature of the software library JidoshaLight consists in recognizing vehicle license plates from images.

Due to its high accuracy, JidoshaLight is the ideal tool for those who need to obtain license plate information in an automatic fashion, without external intervention, through image analysis methods.

2.2. Supported Plates

Country	Syntax	Support
Argentina (032)	LLLNNN (Car) LLNNNLL (Car Mercosur)	Partial
Brazil (076)	LLLNNNN (Car and Motorcycle) LLLNLNN (Car and Motorcycle Mercosur)	Full
Chile (152)	LLXXNN (Car) LLNNN (Motorcycle 1984) LLLNN (Motorcycle 2014)	Full
Colombia (170)	LLLNNN (Car)	Partial
France (250)	LL-NNN-LL (Car SIV)	Partial
Mexico (484)	LL-NNNN-L (Car) LLL-NN-NN (Car) LLL-NNN-L (Car) LL-NN-NNN (Truck) NNN-LL-N (Truck) NNLLNL (Truck) NNNNNNL (Truck) LNNNNNL (Truck) XNN-LLL (Car CDMX) NN-NN-LL (Truck CDMX)	Full
Netherlands (528)	NN-LL-LL (Car 1999) XX-XXX-X (Car 2008) N-LLL-NN (Car 2013)	Partial
Paraguay (600)	LLLNNN (Car) NNNLLL (Motorcycle) LLLLNNN (Car Mercosur) NNNLLLL (Motorcycle Mercosur)	Full
Peru (604)	LXXNNN (Car)	Partial
Uruguay (858)	LLLNNNN (Car) LNNNNNN (Car Punta Cana) LLLNNN (Car Salto)	Full

Legend

L: Letter
N: Number
X: Alphanumeric

2.3. JidoshaLight SDK Structure

The JidoshaLight software development kit (SDK) includes, besides the license plate recognition libraries `libjidoshaLight.so`,

`libjidoshaLightRemote.so`, `libjidoshaLightJava.so` and their C and Java APIs, precompiled sample applications, the source code for those applications, a basic build script, this manual, and an image of a license plate for testing.

All paths used in this manual are relative to the root directory of the SDK, namely `JidoshaLight_TARGET_x.y.z`

2.3.1. Supported APIs

Jidosha's primary API is the `JidoshaLight C/C++ API` and it can be found inside `include` and `lib` folders. Wrappers (bindings) for other languages are also provided alongside the SDK and can be seen inside `wrappers`.

For unsupported languages contact `suporte@pumatronix.com.br`.

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

JIDOSHA legacy API can be found inside `jidoshapc` directory. It should not be used for new designs.

3. JidoshaLight Linux

3.1. Use conditions

The software library JidoshaLight Linux was created to work along with a USB hardkey (security key) which accompanies the library. In other words, for correct functioning the hardkey must be connected to a USB port of the device in which the software license will be used.

There are two hardkey versions: demonstration and full. The demonstration version has an expiry date, while the full one does not. When the expiry date is reached, the library will automatically start returning empty plates. If your demonstration hardkey has expired and you would like to purchase a license or extend the demonstration period, please contact Pumatronix (**contato@pumatronix.com.br**).

Minimum Requirements

Plataform	Libraries
PC_LINUX_64 PC_LINUX_32	• **GLIBC 2.7** • **GLIBCXX 3.4.11**
ARM_A9 ARM_A9_HF ZYNQ7000	• **GLIBC 2.17** • **GLIBCXX 3.4.15**

3.2. Software architecture

The library API calls can be either local, or remote through an IP network.

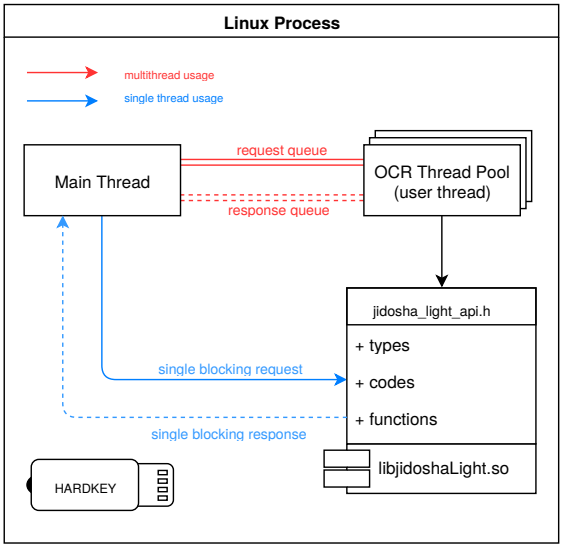
Local calls are executed in the same thread in which the call was made. For licenses with more than 1 thread enabled or in situations where the main thread must not be blocked while the image is processed, new threads must be created for processing.

Remote calls can be either synchronous or asynchronous. In both cases the calls are made locally and the images are processed remotely in a server. The use of a license is necessary only for the server processing the images, and is not needed for the client machine device using the remote library.

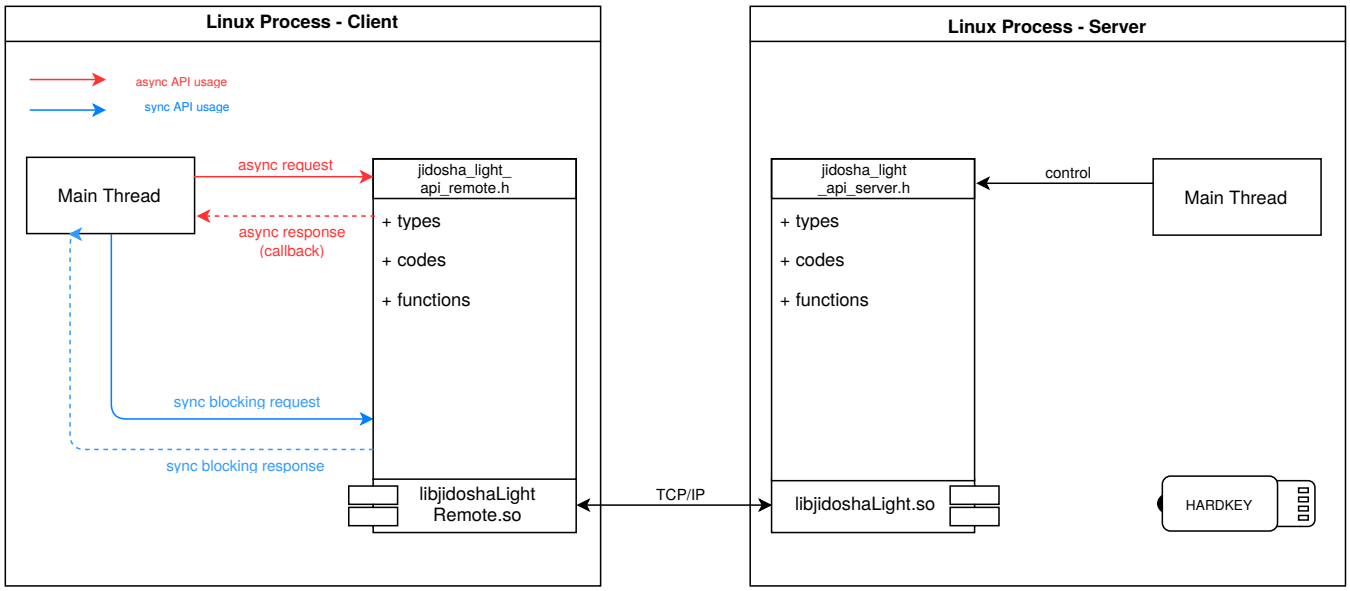
Synchronous calls are blocking and return the processing result at the end of the call.

For the asynchronous interface, the call returns immediately and the processing result is returned through a user-supplied callback.

The following figure shows the suggested architecture for a Linux or Windows™ using JidoshaLight with local calls, whether single thread or multithread. For correct functioning of the application, the `libjidoshaLight.so` library must be linked to the application and the hardkey must be plugged in the machine. Next, for the single thread case, simply call the API functions. For the multithread case, the application must create the necessary processing threads and make calls to the JidoshaLight API functions from those threads.



The following figure shows the suggested architecture for using the library through remote calls. For correct functioning of the applications, the `libjidoshaLightRemote.so` library must be linked to the client application. The `libjidoshaLight.so` library must be linked to the server application and the hardkey must be plugged in the server. The client appications and the server must be connected through a TCP/IPv4 network, whether real or virtual (for example, loopback). Though not shown in the figure, and like the local calls case, the client application can have multiple threads. The server might limit the number of concurrent active sessions based on the license.



3.3. Restrictions

The library supports multithread and multiprocess applications. The maximum number of threads from all processes is limited by the license.

The library does not support process forks.

3.4. Installation

3.4.1. Hardkey permissions configuration

For correct functioning of the USB hardkey, the access permissions of its **udev** must be changed. For your help, in the Jidosha SDK there is a script available at `res/scripts/install_udev.sh` that when the `-i` option is passed, it will automatically setup the hardkey permissions. If you want to do it manually add the following line:

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

to 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 onward:       /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 onward:      /lib/udev/rules.d/50-udev-default.rules
openSUSE 11.2 onward:    /lib/udev/rules.d/50-udev-default.rules
```

For Debian, add the following lines:

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

to the end of the file:

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

For instructions on how to enable the hardkey for other Linux distributions, please contact Pumatronix.

3.4.2. Environment variables configuration

Before running the test applications included in the SDK, or any other application using JidoshaLight Linux, it is necessary to configure some environment variables.

First, it is necessary to add to the system's search path the directory containing the libraries, as follows:

```
```

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

3.4.2.1 Log and audit system

> ATTENTION: starting from version 3.3.0 the log and audit system comes **DISABLED** by default

The JidoshaLight library has a built-in log system capable of auditing the library behavior on the field. To enable some pre-configured debug messages, export the environment variable JL_LOGCFG as "default".

```
$ export JL_LOGCFG=default
```

The log system allows other debug messages to be enabled. It also supports redirecting its output to a file. Those feature are specified by a configuration file whose structure is provided next.

The configuration file is read only once during library load and the search order is:

1. an absolute path in environment variable **JL_LOGCFG** (if any)
2. a file named **jlog.conf** in the current directory [./]

Configuration file structure:

```
# 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}
#
# Especial 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 log configuration file above will configure the library to generate all enabled messages, both to the relative file `log.txt` and to stdout. If you wish to disable one or more types of messages, comment the line with '#' or remove it.

To disable messages to stdout and instruct the library to generate a log file, follow the example below for each desired type of message:

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

3.4.3. Preferences file configuration

The library allows an optional preferences file to be used. Through this file it is possible to configure the `struct JidoshaLightConfig` fields, overriding the values of the `struct` passed to 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 searches for the file `jl_anpr_preferences.json` in the working directory. If the file exists, it is loaded; otherwise, the library will search for it in the path indicated by the environment variable `JL_ANPR_PREFS`. If the environment variable does not exist, no preferences file is loaded. If it exists and the indicated path is a valid file, the file is loaded.

If a preferences file was loaded, the preferences will only be applied if the field `ENABLE_CONFIG_OVERRIDE` is `true`. `struct JidoshaLightConfig` fields that are absent from the preferences file will receive their default value, as defined by the library.

Since the preferences file is loaded upon library initialization (usually at the start of the process that uses the library), changes to the file will only take effect when the library is reloaded. This behavior might change in future versions.

3.5. Sample applications

The SDK includes some sample applications with source code:

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

If you wish to recompile the samples, use the make_samples.sh script. See the example for ARM Linux below:

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

After configuring the hardkey and environment variables and plugging in the hardkey, you will be able to run the samples.

To run JidoshaLightSample, open a terminal and run it with the supplied reference image:

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

The application should inform the library version as well as the license plate recognition result.

```
-- 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 samples, run the server program from a terminal:

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

The application should inform that it was initialized on TCP port 51000 and that 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
```

Next, in another terminal, run the client program. The client should inform the library version as well as the license plate recognition result and statistics:

```
$ ./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 %)
```

Go back to the server terminal and verify the additional log messages informing about the 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
```

4. JidoshaLight Windows

4.1. Use conditions

The software library JidoshaLight Windows was created to work along with a USB hardkey (security key) which accompanies the library. In other words, for correct functioning the hardkey must be connected to a USB port of the device in which the software license will be used.

There are two hardkey versions: demonstration and full. The demonstration version has an expiry date, while the full one does not. When the expiry date is reached, the library will automatically start returning empty plates. If your demonstration hardkey has expired and you would like to purchase a license or extend the demonstration period, please contact Pumatronix (**contato@pumatronix.com.br**).

Minimum Requirements

Plataform	Versions
PC_WINDOWS_64 PC_WINDOWS_32	• **Windows 7** • **Windows 7**

4.3. Installation

For installation you only need to plug the hardkey in the Windows machine that will run the library. Windows should automatically install a driver when the hardkey is first plugged in. To test whether the installation was successful, you can run the sample applications detailed in section 4.5.

4.4. Environment variables configuration

Before running the test applications included in the SDK, or any other application using JidoshaLight Windows, it is necessary to configure some environment variables.

First, it is necessary to add to the system's search path the directory containing the libraries. Access the SDK folder and run the following command:

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

NOTE: The previous command will only change the PATH for the current terminal session. To set the variable permanently, go to the Advanced tab in System Properties > Environment Variables and add the path to the SDK lib folder, either to the system or to the user variable named `Path`.

4.4.1 Log and audit system

Ver **4.4.2.1 Log and audit system**.

4.5 Preferences file configuration

See **3.4.3. Preferences file configuration**.

4.6. Sample applications

The SDK includes some sample applications with source code:

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

To run JidoshaLightSample, for example, run the command below from the SDK folder. You should 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 bits --
    Compilation Date: month day year time
    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 1500
-- LicenseInfo --
>> Serial: 0x8f230e5
>> Customer: empresa
>> 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
```

5. JidoshaLight Linux/FPGA

5.1. Use conditions

The JidoshaLight Linux software library with FPGA acceleration is licensed from a license file linked to the hardware device, hence a hardkey is not needed. The library supports hardware acceleration with Xilinx FPGAs of the **Zynq-7000** family. The standard version has support for the `XC7Z020-CLG400` device, and can be adapted for devices with higher capacity.

See the **Minimum Requirements** table for more information.

5.2. Software architecture

The software architecture is similar to the non-accelerated Linux version, described in **3.2. Software architecture**

The main differences are due to the additional programming interfaces and in the reserved shared memory area. Configuration and installation are detailed in **5.4. Installation**

The figures below illustrate the suggested architecture for both local and remote API.

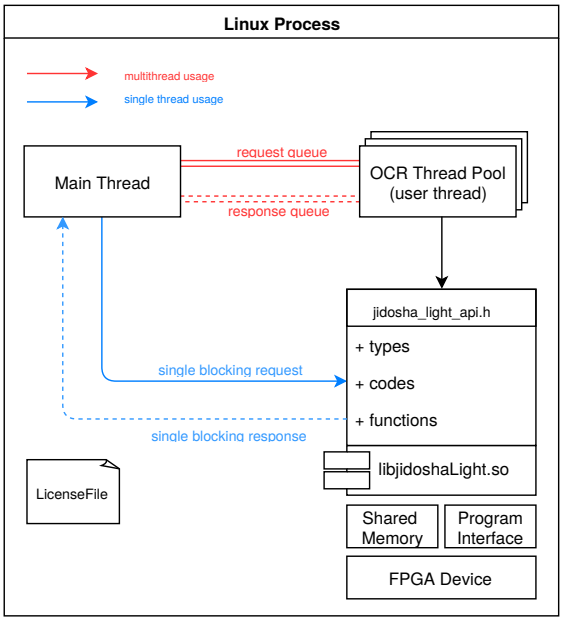


Diagram with local API use cases

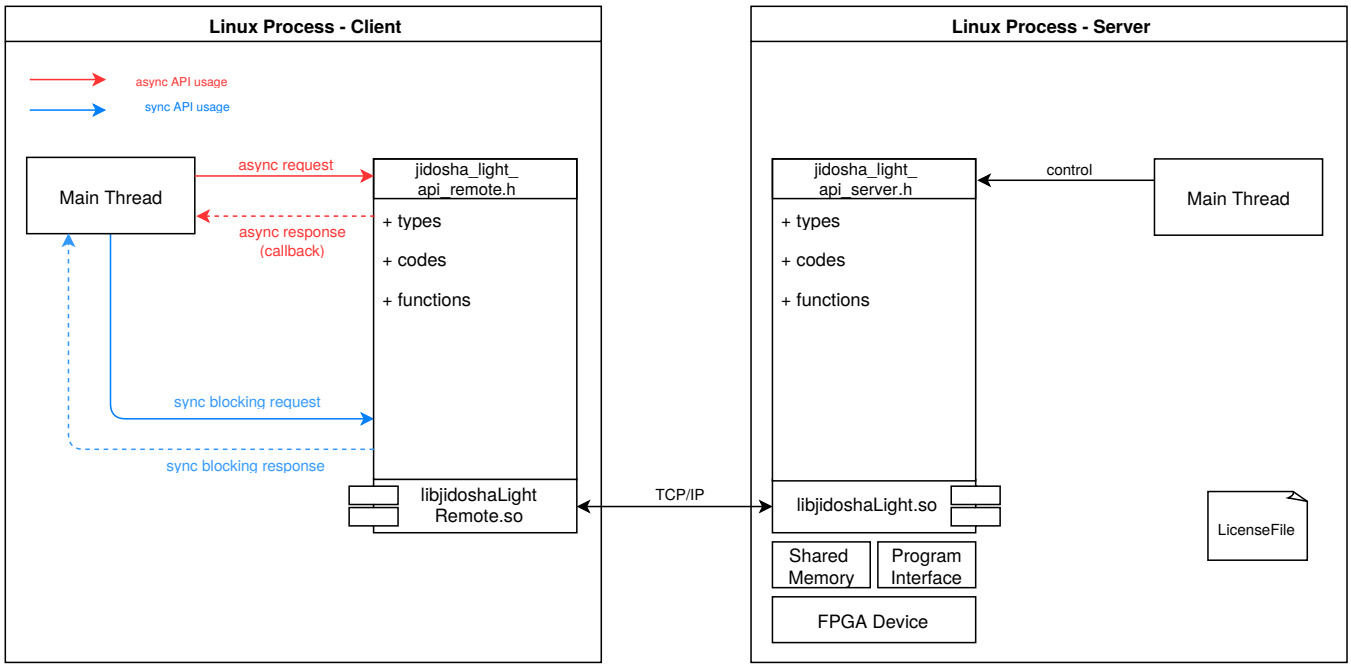


Diagram with remote API use cases

5.3. Restrictions

The library with FPGA acceleration supports multithread applications, where the maximum number of threads is limited by the acquired license. There is no support for multiprocess applications.

5.4. Installation

5.4.1. License configuration

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

To obtain the identifier for you hardware, you need to execute the JidoshaLightDna application from a terminal running in the device, which must be previously configured as described in **5.4.2. Environment variables configuration**.

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

IMPORTANT: The concurrent use of this application with any other that uses the library is not allowed and may freeze the device.

5.4.2. Environment variables configuration

Before running the test applications included in the SDK, or any other application using JidoshaLight Linux/FPGA, it is necessary to configure some environment variables.

When using the FPGA-accelerated version of the library, besides the variables described in **3.4.2. Environment variables configuration**, the following configurations are needed:

- JL_MPOOL_BASE: Base address of memory allocated for communication between library and FPGA. If undefined, the default value is 0x3A000000. For example:

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

- JL_MPOOL_BUFFERNUM: Number of memory buffers needed to run the library. If undefined, the default value is 32 buffers, while the minimum required value is 12 buffers per concurrent thread that uses the library. For example:

```
$ export JL_MPOOL_BUFFERNUM=32
```

- JL_MPOOL_BUFFERSIZE: Size in bytes of each memory buffer, which has to be a multiple of 4096 bytes. If undefined, the default value is 2097152 bytes (2MB). This is the required value to process images of size up to 800x600 pixels. This value needs to be larger than 4 times the image resolution. For example:

```
$ export JL_MPOOL_BUFFERSIZE=2097152
```

- JL_LICENSE_FILE: Path to the license file. The license file is linked to the device being used. To obtain the device identifier, see **5.4.1. License configuration**. Ex.:

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

5.4.3. Linux kernel configuration

Two interfaces are needed for communication between the library and the FPGA device: one dedicated to FPGA configuration and a shared memory for data exchange.

The configuration interface is provided by Xilinx through a *char device* (/dev/xdevcfg) and is not currently part of the standard Linux kernel. Source code and installation instructions can be found at the **Xilinx Wiki page**.

The shared memory needs to be visible in `/dev/mem` and reserved for exclusive use by the library, and must not be used by the Linux kernel.

To achieve this setting, it is necessary to limit the amount of memory used by the kernel on its initialization.

Below is an example of how to reserve the last 96MB of memory of a device with 1GB of RAM.

In u-boot, configure:

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

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

Also add to the Linux device tree the following configurations:

```
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;
  };
};
```

5.5. Sample applications

See **3.5. Sample applications**.

6. JidoshaLight Android™

6.1. Use conditions

The software library JidoshaLight Android™ was created to work along with a license file that must be generated after the user installs the application. The license file is generated per installation and is linked to the device hardware, which means that it is necessary to generate a new license when the application is reinstalled or when there is a hardware change (the device's SIM card is included in the hardware category). It is not necessary to generate a new license when changing the device's battery. For temporary (time-to-live) licenses, the device's date and time must be synchronized over the mobile network provider.

The library has support for multithread applications, where the **maximum number of threads** and the **minimum processing time** are limited by the acquired license. When using the server API, the **maximum number of simultaneous connections** accepted by the server is also limited by the license.

The library functionalities are accessed through the Java API. The present version is compatible with ARM™ processors (armv7-a) with Android™ 4.4 or newer for library only use (shared libraries and basic Java classes), and Android™ 8 or newer for installing the sample application `(CameraView.java, BackCamera.java, BaseCamera.java)`.

6.2. Software architecture

The recommended way of working with the JidoshaLight library on Android is through the **asynchronous client and server** topology. This topology allows optimizing the flow of the license plate recognition process, since all processing and memory allocation happen in native code. This topology also allows processing images externally without any further changes in the application. The sample application included with the SDK implements this kind of topology.

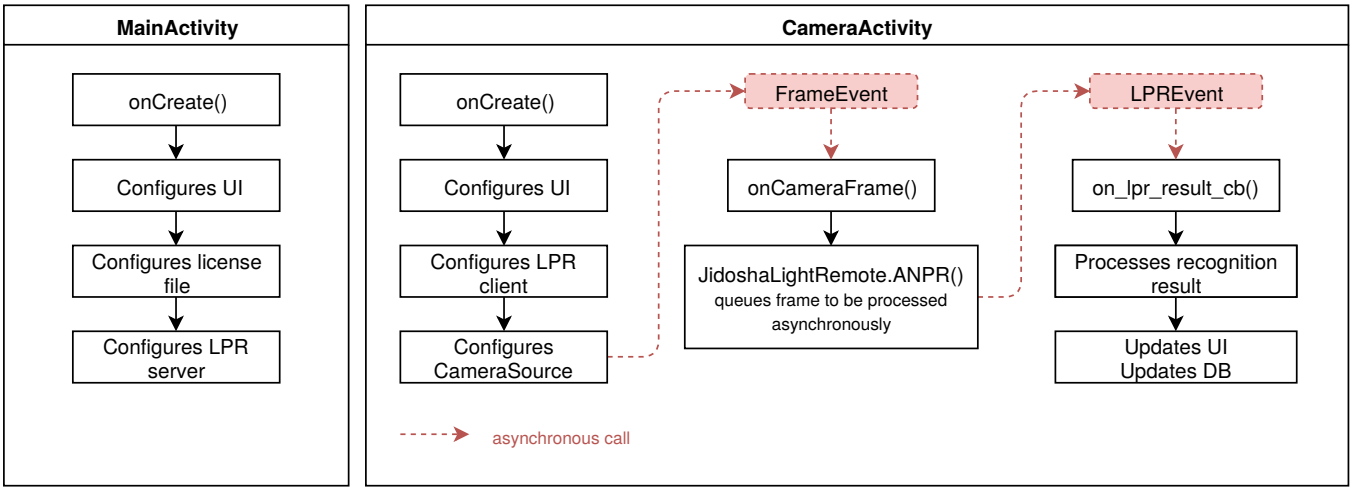
Usually the best experience is obtained in *freeflow* mode. In this mode, as opposed to *point and shoot* mode, the license plate recognition process is applied to all images sent by the camera, with no user intervention (shooting) necessary. Therefore, as soon as the camera is activated, the processing begins and callbacks with the recognition results start being generated in an asynchronous manner. The **asynchronous client and server** topology is fit to be used in *freeflow* mode without any significant changes in the application implementation.

I. Pros of asynchronous client in freeflow mode:

1. Simplified application code, which makes it easier to integrate the library
2. Better user experience (faster recognitions)
3. Automatic resource management (queues, threads, network)
4. Less coupling between image acquisition (camera), license plate recognition processing, and output (UI and DB)
5. Ability to process locally or remotely without source code change

II. Cons of asynchronous client in freeflow mode:

1. Callbacks happen in a thread separate from 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



Example of flow of an synchronous client application in freeflow mode
MainActivity configures the library license and initiates the processing server
CameraActivity configures the license plate recognition client, the camera (CameraSource), and handles events

6.3. Restrictions

> **NOTE: The memory restrictions below do not apply to natively allocated memory (inside the shared library).**
>
> **For high-performance applications, we recommend using the asynchronous client API.**

The Android™ operating system has strict restrictions regarding use of RAM by applications. The maximum amount of memory an application is allowed to allocated on the heap differs between devices, but is between 24MB and 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 smartphone cameras and tablets is getting larger and larger, developers must mind the size of images they are trying to recognize in order to decrease the chances of an `OutOfMemoryError` exception. For example, and 8MP image in ARGB8888 Bitmap format takes 24MB, which is larger than the memory limit allowed by several devices.

Since **JidoshaLight needs the plate characters to be at most 30 pixels high**, a **1280x720** resolution image is enough for license plate recognition. If you wish to show the user a high resolution image, you can acquire and store the image in high resolution, and use decoding and resizing methods supported by Android™'s `android.graphics.Bitmap` class. In that case, you must keep in mind the reduction of character size when resizing the image, and guarantee a size between 15 and 30 pixels in the reduced-resolution image.

Another Android™ operating system pitfall relates to blocking the graphical user interface thread. By default, the graphical user interface thread is the only one created by the application, and all processing happens within it. In order for the graphical interface to remain responsive to the user, it must not be blocked for more than a few milliseconds. If that happens, the operating system will issue an alert to the user informing that the application has stopped responding, or simply terminate the application.

For more information on memory management on Android:

- **https://developer.android.com/training/articles/memory.html**
- **https://developer.android.com/training/displaying-bitmaps/index.html**

6.4. Installation

> **NOTE: All Java classes marked as `native` cannot have their package modified.**
>
> **All other classes can be moved freely.**

The JidoshaLight for Android SDK includes the API and native C language shared libraries (which can be accessed by any JNI code), the wrappers and libraries for Java, and a sample application described in **6.5. Sample application**.

6.4.1 Licensing

A valid license is needed to use the JidoshaLight library on Android™ systems. The licensing is made by device and for a limited time period, which means the license must be generated again upon hardware change or expiration.

The API function `JidoshaLight.setLicenseFromData()` should be used to pass the license content to the library and this must be done **before** any other API calls. A helper class `JidoshaLightAndroidHelper.java` is supplied with the sample application to help with this procedure.

The license request procedure is fully automated by the function `JidoshaLight.getLicenseFromServer`, beforehand however, the user must register the **Device ID** with Pumatronix and the device must have an active internet connection during the request.

For more information on licensing, send an e-mail to **suporte@pumatronix.com.br**.

6.4.2 Permissions

The following permissions have to be added for correct functioning of the library and must be included in the application's `AndroidManifest.xml`.

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

6.5. Sample application

The sample application included in the SDK was developed to be used with Android™ Studio 4 or newer. It shows how to use the library to recognize license plates in a video stream from the smartphone's back camera or from an external MJPEG camera. It also shows how to implement the

configuration screen for the library parameters, camera Activity with grid and zoom support, recognition list, recognition details, licensing system, and integration with database in order to search for information related to the detected license plate.

6.5.1 Package `br.gaussian.io`

- **Mjpeg.java**: Wrapper class for the native MJPEG decoder API. See class `MjpegCamera.java` for a higher-level implementation.

6.5.2 Package `br.gaussian.jidoshalight`

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

6.5.3 Package `br.gaussian.jidoshalight.camera`

- **BaseCameraSource.java**: Base class for all camera implementations
- **BackCamera.java**: Back camera implementation
- **MjpegCamera.java**: External MJPEG camera implementation
- **CameraFrame.java**: Class that stores a frame from a camera
- **CameraView.java**: View capable of displaying an image flow from a BaseCameraSource; provides grid support, overlay for license plates, pinch-to-zoom and ROI selection

6.5.4 Package `br.gaussian.jidoshalight.sample`

6.5.4.1 Common

- **common/JidoshaLightAndroidHelper.java**: Helper class containing support methods for license file reading and writing, besides other utility methods.
- **common/JidoshaLightServerHelper.java**: Helper class containing support methods for initializing a local license plate recognition server.

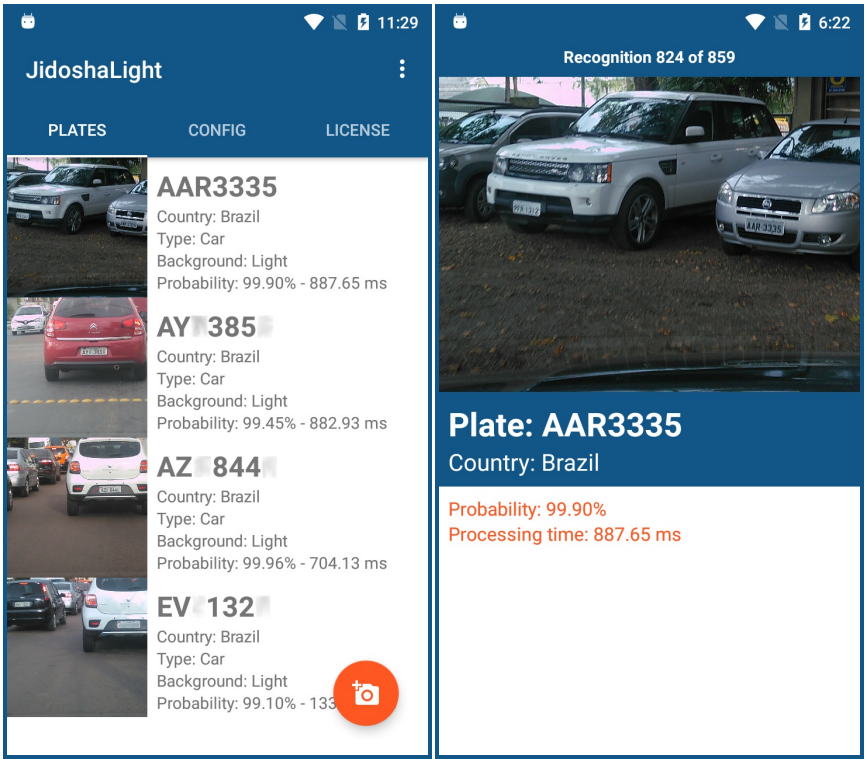
6.5.4.2 Activities

MainActivity

The application's main Activity, it shows how to configure the license file and initialize a local license plate recognition server.

DetailActivity

This Activity is triggered when selecting an item from the recognition list. It expands the information about a specific recognition.



MainActivity and DetailActivity, respectively
Characters redacted for privacy

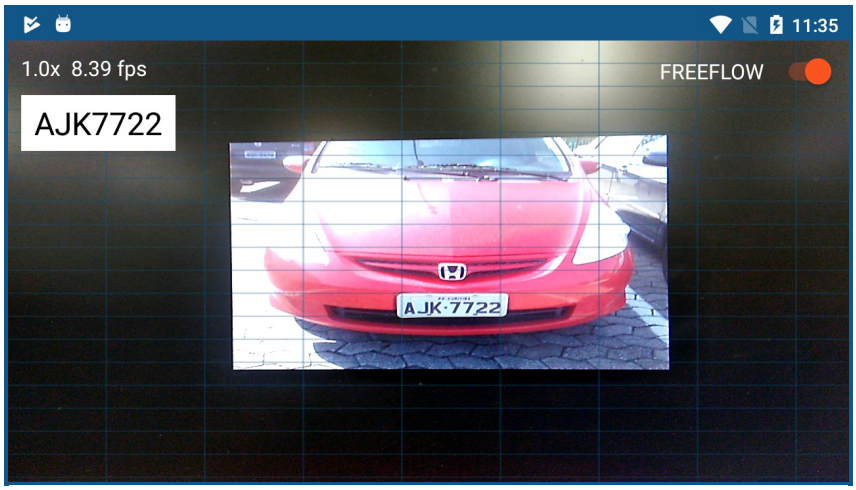
CameraActivity

Shows how to configure and capture camera images, allowing to:

- Instantiate a camera from the configuration;
- Configure a license plate processing client;
- Configure the optical zoom with the pinch gesture;
- Select the region of interest (ROI) by tapping;
- Enable/disable processing.

For better recognition performance, the camera's zoom and focus must allow capturing sharp and properly sized images. The plate height must be between 30 and 50 pixels. The guides help in framing the plate, guaranteeing correct plate size and orientation when the image is captured. The plate must have approximately the size of a grid rectangle.

Because the smartphone or tablet camera is constantly moving, ideally the autofocus and video stabilization resources should be enabled, if present. Depending on the application, we recommend exporting manual focus and exposure settings for best results.



CameraActivity
The ideal plate height for recognition should be same as that of a grid rectangle (the plate does not need to be aligned with the grid)

ROI region selection
a) Tap and hold on the screen to enable ROI use
b) Tap twice to begin selecting the ROI points
c) Tap the four points that delimit the region (clockwise)

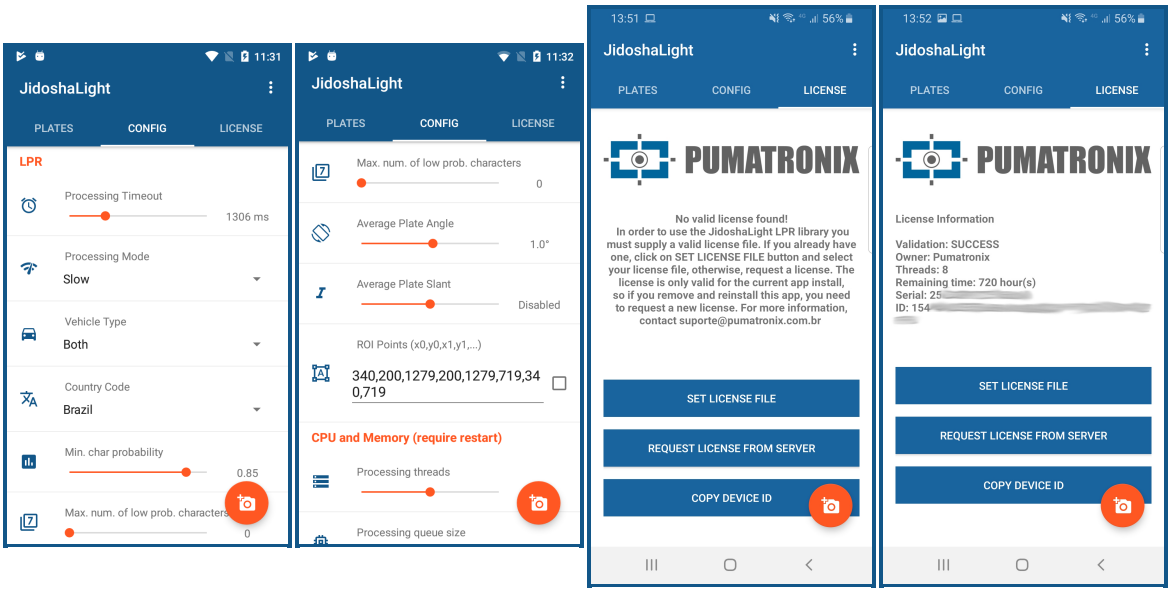
6.5.4.3 Fragments

ConfigurationFragment

Fragment used to show and configure the JidoshaLight library parameters. The configurable parameters are the same as those available for the Java/C API.

LicenseManagerFragment

Fragment used to implement the licensing system. It shows how to extract the **Device ID** and how to request a license file.



ConfigurationFragment (1,2) and LicenseManagerFragment (3,4)
Changing the license file or some configurations requires restarting the application

7. User APIs

The JidoshaLight library exports 4 different APIs for automatic license plate recognition: Local, Synchronous Remote, Asynchronous Remote, and Server. The Synchronous Remote API will be discontinued in the future and should not be used in new projects. Besides the recognition APIs, the library provides some utility functions, such as a MJPEG-format video receiver and a library license reader. These additional APIs are partially documented in this manual. For more information on their use, consult the header files in folder `include/gaussian/common`.

By default, the languages supported by the API and included in the SDK are C/C++ and Java. Wrapper for Python, C# and Delphi may be supplied on demand.

For questions or inquiries regarding support for other languages, send an email to **contato@pumatronix.com.br** with subject "JidoshaLight API".

7.1. JidoshaLight C/C++ API

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

7.1.1. JidoshaLight C/C++ API (Local)

The Local API contains the types, definitions, and basic functions for local processing of images. Since release 2.4.4 the contents are divided between the `jidosha_light_api_common.h` and `jidosha_light_api.h` files.

> For backward compatibility, `jidosha_light_api_common.h` is included by `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_ARGENTINA                = 32,
    JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL                   = 76,
    JIDOSHA_LIGHT_COUNTRY_CODE_CHILE                    = 152,
    JIDOSHA_LIGHT_COUNTRY_CODE_COLOMBIA                 = 170,
    JIDOSHA_LIGHT_COUNTRY_CODE_MEXICO                   = 484,
    JIDOSHA_LIGHT_COUNTRY_CODE_PARAGUAY                 = 600,
    JIDOSHA_LIGHT_COUNTRY_CODE_PERU                     = 604,
    JIDOSHA_LIGHT_COUNTRY_CODE_URUGUAY                  = 858,
    JIDOSHA_LIGHT_COUNTRY_CODE_NETHERLANDS              = 528,
    JIDOSHA_LIGHT_COUNTRY_CODE_FRANCE                   = 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
);
```

7.1.1.1. Types

`enum JidoshaLightVehicleType`

Description

Defines the license plate types that the library should search for in the image.

Members

`JIDOSHA_LIGHT_VEHICLE_TYPE_CAR`: only car/truck/bus plates (single-line layout)

`JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO`: only motorcycle plates (two-line layout)

`JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH`: both types of plates

`enum JidoshaLightMode`

Description

Defines the processing strategies that may be used by the license plate recognition algorithm. The **FAST** option uses the least processing effort available for reading, while **ULTRA_SLOW** uses the most. The more the processing effort, the higher the probability of successfully reading the plate.

Members

`JIDOSHA_LIGHT_MODE_DISABLE`: value reserved for future use. Currently has the same effect as ULTRA_SLOW.

`JIDOSHA_LIGHT_MODE_FAST`: the fastest processing strategy, recommended only for cases where processing time is critical

`JIDOSHA_LIGHT_MODE_NORMAL`: a moderate processing strategy with longer processing time and higher recognition rate than the FAST strategy

`JIDOSHA_LIGHT_MODE_SLOW`: slow processing strategy, with longer processing time and higher recognition rate than the NORMAL strategy

`JIDOSHA_LIGHT_MODE_ULTRA_SLOW`: this strategy has the longest processing time and highest recognition rate

One of the above modes must necessarily be used. Optionally, one may also select the localization strategy used by the license plate recognition algorithm. The default option is **LOCALIZATION_MODE_0**. The other options currently affect only the processing of Brazilian license plates.

`JIDOSHA_LIGHT_LOCALIZATION_MODE_0`: localization strategy with highest processing time and recognition rate

`JIDOSHA_LIGHT_LOCALIZATION_MODE_1`: faster localization strategy, with lower recognition rate

`JIDOSHA_LIGHT_LOCALIZATION_MODE_2`: fastest localization strategy, with lowest recognition rate, applicable only to car license plates and not motorcycle plates

---

`enum JidoshaLightCountryCode`

Description

Defines the ISO 3166-1 numeric code of countries supported by the library. The availability of each country is limited by the license.

Members

`JIDOSHA_LIGHT_COUNTRY_CODE_CONESUL`

`JIDOSHA_LIGHT_COUNTRY_CODE_ARGENTINA`

`JIDOSHA_LIGHT_COUNTRY_CODE_BRAZIL`

`JIDOSHA_LIGHT_COUNTRY_CODE_CHILE`

`JIDOSHA_LIGHT_COUNTRY_CODE_COLOMBIA`

`JIDOSHA_LIGHT_COUNTRY_CODE_MEXICO`

`JIDOSHA_LIGHT_COUNTRY_CODE_PARAGUAY`

`JIDOSHA_LIGHT_COUNTRY_CODE_URUGUAY`

`JIDOSHA_LIGHT_COUNTRY_CODE_NETHERLANDS`

`JIDOSHA_LIGHT_COUNTRY_CODE_FRANCE`

`enum JidoshaLightRawImgFmt`

Description

Defines the RAW image formats supported by the library.

Members

`JIDOSHA_LIGHT_IMG_FMT_XRGB_8888` : 32-bit XRGB format, with the least significant byte used for the blue channel. The most significant byte is ignored

`JIDOSHA_LIGHT_IMG_FMT_RGB_888` : 24-bit RGB format, with the least significant byte used for the blue channel

`JIDOSHA_LIGHT_IMG_FMT_LUMA` : 8-bit format containing only the luminance channel

`JIDOSHA_LIGHT_IMG_FMT_YUV420` : non-interlaced 8-bit YUV format with 4:2:0 sampling

`struct JidoshaLightConfig`

Description

The purpose of this structure is to configure the behavior of the library during the license plate recognition call.

Members

`int configId`: reserved for future use and currently ignored by the library

`int vehicleType`: type of plate that the library should search for. See **enum JidoshaLightVehicleType**

`int processingMode`: processing strategy to be used. See **enum JidoshaLightMode**

`int timeout`: maximum time in milliseconds for license plate recognition. The value `0` indicates there is no timeout. Values different than `0` help keep the average processing time short - as soon as the timeout expires, the processing is interrupted and the function returns. A suitable value should be determined based on the image resolution and the CPU

`int countryCode`: numeric code of the country for which license plates should be recognized. See **enum JidoshaLightCountryCode**

`float minProbPerChar`: value between `0.0f` and `1.0f` used to define the minimum probability that each plate character should have to be considered valid **(recommended: 0.85)**

`int maxLowProbabilityChars`: maximum number of characters with probability less than `minProbPerChar` such that the plate is still considered valid - a plate considered invalid will be returned as an empty string, i.e. '\0'

`char lowProbabilityChar`: substitution character used in place of characters with probability lower than `minProbPerChar`

`float avgPlateAngle`: average angle between license plates and the horizontal image axis

`float avgPlateSlant`: average angle between license plates and the vertical image axis

`int maxCharHeight`: maximum acceptable character height, in pixels

`int minCharHeight`: minimum acceptable character height, in pixels

`int maxCharWidth`: maximum acceptable character width, in pixels

`int minCharWidth`: minimum acceptable character width, in pixels

`int avgCharHeight`: average character height, in pixels (default value: 20)

`int avgCharWidth`: average character height, em pixels (default value: 7)

`int xRoi[4]` and `int yRoi[4]`: x and y coordinates of four points of the Region of Interest (ROI), in any order. A ROI is a quadrilateral inside the image, inside which one expects to find license plates. The use of a ROI decreases processing time and may improve recognition rates, since it excludes unimportant image regions where a license plate has a low probability of appearing. Defining all coordinates equal to zero will have the ROI be ignored and the entire image will be processed. Values larger than the image dimensions, or negative values, will result in the error return code `JIDOSHA_LIGHT_ERROR_INVALID_ROI`. Changing the ROI results in the library recalculating the ROI mask, which will impact processing time for the first image after the change.

> Warning: ROI's points coordinates span from (0,0) at the top-left corner of the image to (width-1, height-1) at the bottom-right corner. Thus, in a

800x600 imagem, the accepted range of values are (0,0) to (799,599). The 4 points must not be collinear at once.

How to measure `avgPlateAngle` and `avgPlateSlant`

`struct JidoshaLightRecognitionInfo`

Description

The purpose of this structure is to return information about JidoshaLight's processing time, which makes it easy to diagnose performance issues. All times are in milliseconds.

Members

`double totalTime`: total image processing time (sum of remaining times).

`double localizationTime`: time used during license plate localization in the image.

`double segmentationTime`: time used to extract characters from the license plate.

`double classificationTime`: time used to classify the license plate characters.

`double loadDecodeTime`: time used reading and decoding the image file or image structure in memory.

`int libVersion[3]`: library version that processed the image.

`char libSHA1[41]`: hash identifier of the library that processed the image.

`struct JidoshaLightRecognition`

Description

The purpose of this strcuture is to store the license plate recognition result, including: the plate characters, the probability of each character being correct (that is, the recognition reliability), and the plate coordinates in the image.

Members

`int frameId`: reserved for future use, this field is currently always zero.

`char plate[8]`: null-terminated 7-character license plate string, or an empty string if a license plate could not be found.

`float probabilities[7]`: value between 0.0 and 1.0 indicating the reliability of each character's recognition.

`int xText` and `int yText`: top-left coordinates of the rectangle containing all license plate characters, if it was found.

`int widthText`: width of the license plate rectangle.

`int heightText`: height of the license plate rectangle.

`int xChar[7]` and `int yChar[7]`: top-left coordinates of each of the recognized characters.

`int widthChar[7]`: width of the rectangle of each recognized character.

`int heightChar[7]`: height of the rectangle of each recognized character.

`int textColor`: license plate text color, 0 - black, 1 - white.

`int isMotorcycle`: indicates whether the license plate is that of a motorcycle, 0 - non-motorcycle, 1 - motorcycle.

`int countryCode`: numeric code (in ISO 3166-1 format) of the country of the recognized license plate. Possible values are defined in **enum JidoshaLightCountryCode**.

`JidoshaLightRecognitionInfo info`: structure containing information about processing time.

`struct JidoshaLightLicenseInfo`

Description

Structure used to store information about the license used by the JidoshaLight library.

Members

`uint64_t serial` : serial number of the license

`char customer[64]` : name of the client that acquired the license

`int maxThreads` : maximum number of enabled processing threads

`int maxConnections` : maximum number of enabled concurrent connections

`int state` : license status (see **Function return codes**)

`int ttl` : time-to-live in hours of RTC (Real Time Clock)-type licenses. If the license has no expiry time, this field has value **-1**

`struct JidoshaLightRecognitionList`

Description

Opaque type used by the library to return a list of objects of type **JidoshaLightRecognition**. Functions that manipulate the list insert new elements at the end. To reset a list, you only have to destroy the current one and create a new one.

To avoid memory leaks, the user must always destroy lists that are no longer in use.

This type is not thread-safe.

Members

None

Related methods

- **jidoshaLight_ANPR_createList**
- **jidoshaLight_ANPR_duplicateList**
- **jidoshaLight_ANPR_destroyList**
- **jidoshaLight_ANPR_getListSize**
- **jidoshaLight_ANPR_getListElement**

`struct JidoshaLightImage`

Description

Opaque type used by the library to load an image to be processed. When created, a `JidoshaLightImage` object can be used to load any number of images, even if they are in different formats. However, subsequent calls will overwrite previously loaded contents.

The image decoding process may be delayed until the image needs to be processed if the **LazyDecode** mode is enabled. In that case, the **load** functions only store the raw image buffer and don't process anything. This behavior is useful for client-server appliations, since the decoding computational cost

is delegated to the server.

To avoid memory leaks, the user must always destroy images that are no longer in use.

This type is not thread-safe.

Members

None

Related methods

- **jidoshaLight_ANPR_createImage**
- **jidoshaLight_ANPR_duplicateImage**
- **jidoshaLight_ANPR_destroyImage**
- **jidoshaLight_ANPR_setImageLazyDecode**
- **jidoshaLight_ANPR_loadImageFromFile**
- **jidoshaLight_ANPR_loadImageFromMemory**
- **jidoshaLight_ANPR_loadImageFromRawImgFmt**

7.1.1.2. Methods

`jidoshaLight_ANPR_createList`

Function prototype

```
JidoshaLightRecognitionList* jidoshaLight_ANPR_createList();
```

Description

Function used to create an empty **JidoshaLightRecognitionList**

Parameters

None

Return

A valid pointer to type `JidoshaLightRecognitionList`, or `NULL` in case of failure.

`jidoshaLight_ANPR_duplicateList`

Function prototype

```
JidoshaLightRecognitionList* jidoshaLight_ANPR_duplicateList(JidoshaLightRecognitionList* list);
```

Description

Function used to duplicate a **JidoshaLightRecognitionList**

Parameters

`JidoshaLightRecognitionList* list`: pointer to a `JidoshaLightRecognitionList` object

Return

A valid pointer to type `JidoshaLightRecognitionList`, or `NULL` in case of failure.

`jidoshaLight_ANPR_destroyList`

Function prototype

```
int jidoshaLight_ANPR_destroyList(JidoshaLightRecognitionList* list);
```

Description

Function used to destroy objects created by functions **jidoshaLight_ANPR_createList** and **jidoshaLight_ANPR_duplicateList**.

Parameters

`JidoshaLightRecognitionList* list`: a valid pointer to a `JidoshaLightRecognitionList` object

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_getListSize

Function prototype

```
int jidoshaLight_ANPR_getListSize(JidoshaLightRecognitionList* list);
```

Description

Function used to read the number of elements stored inside the list.

Parameters

`JidoshaLightRecognitionList* list`: a valid pointer to a `JidoshaLightRecognitionList` object

Return

Number of elements in the list (greater than or equal to zero) or -1 in case of failure (invalid `*list*`).

---

jidoshaLight_ANPR_getListElement

Function prototype

```
const JidoshaLightRecognition* jidoshaLight_ANPR_getListElement(JidoshaLightRecognitionList* list, int pos);
```

Description

Function used to retrieve the pointer to the element in position `pos` in the list. The contents of the returned pointer cannot be changed by the user (const).

Parameters

`JidoshaLightRecognitionList* list`: a valid pointer to a `JidoshaLightRecognitionList` object

`int pos`: position of the element to be retrieve, in the range `[0,ListSize)`

Return

A valid pointer to an immutable object of type **JidoshaLightRecognition** or `NULL` in case of failure (invalid `*list` or out-of-range `pos`).

---

jidoshaLight_ANPR_createImage

Function prototype

```
JidoshaLightImage* jidoshaLight_ANPR_createImage();
```

Description

Function used to create a **JidoshaLightImage**

Parameters

None

Return

A valid pointer to type `JidoshaLightImage` or `NULL` in case of failure.

---

jidoshaLight_ANPR_duplicateList

Function prototype

```
JidoshaLightImage* jidoshaLight_ANPR_duplicateList(JidoshaLightImage* img);
```

Description

Function used to duplicate a **JidoshaLightImage**. The duplicated image inherits the state of the original image.

Parameters

`JidoshaLightImage* img`: pointer to a `JidoshaLightImage` object

Return

A valid pointer to type `JidoshaLightImage` or `NULL` in case of failure.

---

jidoshaLight_ANPR_destroyImage

Function prototype

```
int jidoshaLight_ANPR_destroyImage(JidoshaLightImage* img);
```

Description

Function used to destroy objects created by functions **jidoshaLight_ANPR_createImage** and **jidoshaLight_ANPR_duplicateImage**.

Parameters

`JidoshaLightImage* img`: a valid pointer to a `JidoshaLightImage` object

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_setImageLazyDecode

Function prototype

```
int jidoshaLight_ANPR_setImageLazyDecode(JidoshaLightImage* img, int enable);
```

Description

Function used to enable the `LazyDecode` mode (see Description in **JidoshaLightImage**). The change has immediate effect and invalidates any previously loaded image.

Parameters

`JidoshaLightImage* img`: valid pointer to a `JidoshaLightImage` object

`int enable` : 0 disabled (default), 1 enabled

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_loadImageFromFile

Function prototype

```
int jidoshaLight_ANPR_loadImageFromFile (
  JidoshaLightImage* img,
  const char* filename
);
```

Description

Function used to load a `JidoshaLightImage` from a file.

Supported file formats: JPEG, BMP, PNG, and TIFF.

Parameters

`JidoshaLightImage* img`: a valid pointer to a `JidoshaLightImage` object

`const char* filename` : absolute path of the file to be loaded

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_loadImageFromMemory

Function prototype

```
int jidoshaLight_ANPR_loadImageFromMemory (
    JidoshaLightImage* img,
    const uint8_t* buffer,
    int bufferSize
);
```

Description

Function used to load a `JidoshaLightImage` from a file already loaded in memory.

Supported file formats: JPEG, BMP, PNG, and TIFF.

Parameters

`JidoshaLightImage* img`: a valid pointer to a `JidoshaLightImage` object

`const uint8_t* buffer`: pointer to the buffer containing the loaded image

`int bufferSize`: buffer size in bytes

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_loadImageFromRawImgFmt

Function prototype

```
int jidoshaLight_ANPR_loadImageFromRawImgFmt (
    JidoshaLightImage* img,
    const uint8_t* buffer,
    int width,
    int height,
    int stride,
    JidoshaLightRawImgFmt fmt
);
```

Description

Function used to load a `JidoshaLightImage` from a buffer containing an image in RAW format.

See supported formats in **enum JidoshaLightRawImgFmt**.

Parameters

`JidoshaLightImage* img`: a valid pointer to a `JidoshaLightImage` object

`const uint8_t* buffer`: pointer to the buffer containing the image in RAW format

`int width`: image width in pixels

`int height`: image height in pixels

`int stride`: size in bytes of one image row

`JidoshaLightRawImgFmt fmt`: image format

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_fromFile

Function prototype

```
int jidoshaLight_ANPR_fromFile (
    const char* filename,
    JidoshaLightConfig* config,
    JidoshaLightRecognition* rec
);
```

Description

Recognizes a license plate from an image file whose path is supplied by `const char* filename`.

Uses the configuration defined in `JidoshaLightConfig* config` and returns the recognition result in `JidoshaLightRecognition* rec`. If a license plate could not be found in the image, the field `rec->plate` will be empty.

If an error occurs during processing, struct `JidoshaLightRecognition* rec` will be empty and a value different from `JIDOSHA_LIGHT_SUCCESS` will be returned by the function. The possible return values are defined in `enum JidoshaLightReturnCode`.

See supported formats in **jidoshaLight_ANPR_loadImageFromFile**.

Parameters

`const char* filename`: path to the image file.

`JidoshaLightConfig* config`: pointer to the `struct JidoshaLightConfig` containing the library configuration. A `NULL` pointer for this parameter will cause the default library configuration to be used.

`JidoshaLightRecognition* rec`: pointer to the `struct JidoshaLightRecognition` where the license plate recognition result will be stored.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_fromMemory

Function prototype

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

Description

Recognizes a license plate from a buffer containing an image file previously loaded in memory.

Uses the configuration defined in `JidoshaLightConfig* config` and returns the recognition result in `JidoshaLightRecognition* rec`. If a license plate could not be found in the image, the field `rec->plate` will be empty.

If an error occurs during processing, struct `JidoshaLightRecognition* rec` will be empty and a value different from `JIDOSHA_LIGHT_SUCCESS` will be returned by the function. The possible return values are defined in `enum JidoshaLightReturnCode`.

See supported formats in **jidoshaLight_ANPR_loadImageFromMemory**.

Parameters

`const unsigned char* buffer`: byte array containing the image.

`int bufferSize`: array size in bytes.

`JidoshaLightConfig* config`: pointer to the `struct JidoshaLightConfig` containing the library configuration. A `NULL` pointer for this parameter will cause the default library configuration to be used.

`JidoshaLightRecognition* rec`: pointer to the `struct JidoshaLightRecognition` where the license plate recognition result will be stored.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_fromLuma

Function prototype

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

Description

Recognizes a license plate from a buffer containing an image in 8-bit grayscale RAW format.

Uses the configuration defined in `JidoshaLightConfig* config` and returns the recognition result in `JidoshaLightRecognition* rec`. If a license plate could not be found in the image, the field `rec->plate` will be empty.

If an error occurs during processing, struct `JidoshaLightRecognition* rec` will be empty and a value different from `JIDOSHA_LIGHT_SUCCESS` will be returned by the function. The possible return values are defined in `enum JidoshaLightReturnCode`.

Parameters

`unsigned char* luma`: byte array containing the image in 8-bit grayscale RAW format.

`int width`: image width.

`int height`: image height.

`JidoshaLightConfig* config`: pointer to the `struct JidoshaLightConfig` containing the library configuration. A `NULL` pointer for this parameter will cause the default library configuration to be used.

`JidoshaLightRecognition* rec`: pointer to the `struct JidoshaLightRecognition` where the license plate recognition result will be stored.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_fromRawImgFmt

Function prototype

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

Description

Recognizes a license plate from a buffer containing an image in one of the RAW formats defined in `enum JidoshaLightRawImgFmt`.

Uses the configuration defined in `JidoshaLightConfig* config` and returns the recognition result in `JidoshaLightRecognition* rec`. If a license plate could not be found in the image, the field `rec->plate` will be empty.

If an error occurs during processing, struct `JidoshaLightRecognition* rec` will be empty and a value different from `JIDOSHA_LIGHT_SUCCESS` will be returned by the function. The possible return values are defined in `enum JidoshaLightReturnCode`.

See supported formats in **jidoshaLight_ANPR_loadImageFromRawImgFmt**.

Parameters

`const unsigned char* buffer`: byte array containing the image in RAW format.

`int width`: image width.

`int height`: image height.

`int stride`: size in bytes of one image row.

`JidoshaLightRawImgFmt fmt`: image format.

`JidoshaLightConfig* config`: pointer to the `struct JidoshaLightConfig` containing the library configuration. A `NULL` pointer for this parameter will cause the default library configuration to be used.

`JidoshaLightRecognition* rec`: pointer to the `struct JidoshaLightRecognition` where the license plate recognition result will be stored.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_fromImage

Function prototype

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

Description

Recognizes a license plate from a previously loaded **JidoshaLightImage**.

Uses the configuration defined in `JidoshaLightConfig* config` and returns the recognition result in `JidoshaLightRecognition* rec`. If a license plate could not be found in the image, the field `rec->plate` will be empty.

If an error occurs during processing, struct `JidoshaLightRecognition* rec` will be empty and a value different from `JIDOSHA_LIGHT_SUCCESS` will be returned by the function. The possible return values are defined in `enum JidoshaLightReturnCode`.

Parameters

`JidoshaLightImage* img`: pointer to a valid **JidoshaLightImage**

`JidoshaLightConfig* config`: pointer to the `struct JidoshaLightConfig` containing the library configuration. A `NULL` pointer for this parameter will cause the default library configuration to be used.

`JidoshaLightRecognition* rec`: pointer to the `struct JidoshaLightRecognition` where the license plate recognition result will be stored.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_ANPR_multi_fromImage

Function prototype

```
int jidoshaLight_ANPR_multi_fromImage (
    JidoshaLightImage* img,
    JidoshaLightConfig* config,
    int maxPlates,
    JidoshaLightRecognitionList* list
);
```

Description

Recognizes multiple license plates from a previously loaded **JidoshaLightImage**.

Uses the configuration defined in `JidoshaLightConfig* config` and adds `maxPlates` recognitions to the end of `JidoshaLightRecognitionList* list`. If the number of license plates found is less than the specified number, empty `JidoshaLightRecognition` elements will be added to the list until the list has `maxPlates` elements.

If an error occurs, empty `JidoshaLightRecognition` elements will be added to the list and a return code different than `JIDOSHA_LIGHT_SUCCESS` will be returned by the function (see **enum JidoshaLightReturnCode**).

Parameters

`JidoshaLightImage* img`: pointer to a valid **JidoshaLightImage**

`JidoshaLightConfig* config`: pointer to the `struct JidoshaLightConfig` containing the library configuration. A `NULL` pointer for this parameter will cause the default library configuration to be used.

`int maxPlates`: maximum number of plates to be recognized (1 or more)

`JidoshaLightRecognitionList* list`: pointer to a **JidoshaLightRecognitionList** object to which `maxPlates` new **JidoshaLightRecognition** objects will be added.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jidoshaLight_getVersion

Function prototype

```
int jidoshaLight_getVersion(int* major, int* minor, int* release);
```

Description

Used to verify the library version, in major.minor.release format.

Parameters

`int major, minor, release`: pointers to int variables where the version numbers will be written.

Return

Always returns `JIDOSHA_LIGHT_SUCCESS`.

---

jidoshaLight_getBuildSHA1

Function prototype

```
const char* jidoshaLight_getBuildSHA1();
```

Description

Used to verify the SHA1 hash of the library build.

Parameters

None

Return

Returns a pointer to a null-terminated string containing the build SHA1.

---

jidoshaLight_getBuildFlags

Function prototype

```
const char* jidoshaLight_getBuildFlags();
```

Description

Used to verify the library's build options.

Parameters

None

Return

Returns a pointer to a null-terminated string containing the build options.

`jidoshaLight_isRemoteApi`

Function prototype

```
JL_API int jidoshaLight_isRemoteApi();
```

Description

Checks whether the application library implements local or remote processing.

Parameters

None

Return

Returns `0` if the API implements local processing, or a different value if the processing is remote.

`jidoshaLight_getLicenseInfo`

Function prototype

```
int jidoshaLight_getLicenseInfo(JidoshaLightLicenseInfo* info)
```

Description

Function used to read the license information used by the JidoshaLight library.

Parameters

`JidoshaLightLicenseInfo* info`: pointers to a **struct JidoshaLightLicenseInfo**

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success.

`jidoshaLight_getReturnCodeString`

Function prototype

```
const char* jidoshaLight_getReturnCodeString(int rc)
```

Description

Function used to convert a library return code to a C string.

Parameters

`int rc`: some code defined in **enum JidoshaLightReturnCode** or **enum JidoshaLightReturnCodeNetwork**

Return

String representing the error code.

Function return codes

Description

The function return codes are related to the recognition process (`enum JidoshaLightReturnCode`) or to the remote communication process (`enum JidoshaLightReturnCodeNetwork`).

Códigos

- `JIDOSHA_LIGHT_ERROR_FILE_NOT_FOUND`: returned by functions **jidoshaLight_ANPR_fromFile** and **jidoshaLight_ANPR_loadImageFromFile** when the specified file path does not exist.
- `JIDOSHA_LIGHT_ERROR_INVALID_IMAGE`: returned by the image processing and loading functions. Occurs when the image is corrupted.
- `JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_TYPE`: returned by functions **jidoshaLight_ANPR_fromFile**, **jidoshaLight_ANPR_fromMemory**, and functions that load **JidoshaLightImage**. Occurs when one tries to process an image in a non-supported format.

- `JIDOSHA_LIGHT_ERROR_INVALID_IMAGE_SIZE`: returned by functions **jidoshaLight_ANPR_fromFile**, **jidoshaLight_ANPR_fromMemory**, and functions that load **JidoshaLightImage**. Occurs when one tries to process an image whose size exceeds the limits supported by the library (ARM Zynq: 1280x960px, others: 2500x2500px).
- `JIDOSHA_LIGHT_ERROR_INVALID_PROPERTY`: returned by all functions that have parameters. Occurs when an argument is invalid. In the case of functions that take pointers, this code is returned when the argument is **NULL** (except in those cases where **NULL** is a valid value for the parameter).
- `JIDOSHA_LIGHT_ERROR_COUNTRY_NOT_SUPPORTED`: returned by the **ANPR** functions when the country code supplied in the configuration struct is not supported by the library.
- `JIDOSHA_LIGHT_ERROR_API_CALL_NOT_SUPPORTED`: returned when an API function is not available for a specific platform.
- `JIDOSHA_LIGHT_ERROR_INVALID_ROI`: returned when an invalid region of interest is supplied. See the description of **struct JidoshaLightConfig** 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 effects when executed. This can happen when a call must be preceded by another.
- `JIDOSHA_LIGHT_ERROR_LICENSE_INVALID`: returned by the **ANPR** functions when the hardkey is not present or has some issue. Contact Pumatronix for more information.
- `JIDOSHA_LIGHT_ERROR_LICENSE_EXPIRED`: returned by the **ANPR** functions when a demonstration-type hardkey has expired. Contact Pumatronix for more information.
- `JIDOSHA_LIGHT_ERROR_LICENSE_MAX_THREADS_EXCEEDED`: returned by the **ANPR** functions when the maximum number of concurrent threads exceeds the limit allowed by the license.
- `JIDOSHA_LIGHT_ERROR_LICENSE_UNTRUSTED_RTC`: returned by the **ANPR** functions when a license with expiry date has no reliable date and time reference.
- `JIDOSHA_LIGHT_ERROR_OTHER`: returned when an unexpected error occurs. Contact Pumatronix for technical 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 closed unexpectedly.
- `JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_TIMEOUT`: returned when a request was discarded by the server due to timeout.
- `JIDOSHA_LIGHT_ERROR_SERVER_QUEUE_FULL`: returned when a request was discarded by the server due to the queue being full.
- `JIDOSHA_LIGHT_ERROR_SOCKET_IO_ERROR`: returned when a network IO error occurs during a remote session with the server.
- `JIDOSHA_LIGHT_ERROR_SOCKET_WRITE_FAILED`: returned when an error occurs when sending messages between client and remote server.
- `JIDOSHA_LIGHT_ERROR_SOCKET_READ_TIMEOUT`: returned when an error occurs when receiving messages between client and remote server.
- `JIDOSHA_LIGHT_ERROR_SOCKET_INVALID_RESPONSE`: returned when an invalid response was received.
- `JIDOSHA_LIGHT_ERROR_HANDLE_QUEUE_FULL`: returned when the pending request queue has reached the maximum size for a specific asynchronous handle.
- `JIDOSHA_LIGHT_ERROR_SERVER_CONN_LIMIT_REACHED`: returned when trying to connect to a server that alread has the maximum number of open sessions.
- `JIDOSHA_LIGHT_ERROR_SERVER_VERSION_NOT_SUPPORTED`: server version is not compatible with client's library version.
- `JIDOSHA_LIGHT_ERROR_SERVER_NOT_READY`: server is starting and is unable to accept connections yet. Try again later.

7.1.2. JidoshaLight C/C++ (Synchronous Remote)

The Synchronous Remote API extends the local API, allowing the user to configure a remote server to process images remotely instead of locally. It must be used with the `libjidoshaLightRemote.so` library.

The `jidoshaLight_ANPR*` callls defined in the Local API are still valid, but the processing will happen remotely when the application is linked with `libjidoshaLightRemote.so`.

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

7.1.2.1. Methods

`jidoshaLight_setRemoteSyncServerIp`

Function prototype

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

Description

Globally configures the IP address and TCP port used to connect to a remote license plate recognition server. The session is established and closed at each recognition call.

Parameters

`const char* ip`: string containing the server's IP address.

`int port`: the server's TCP port.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

7.1.3. JidoshaLight C/C++ API (Asynchronous Remote)

The Asynchronous Remote API extends the Local API, allowing the user to configure a remote server to process images remotely instead of locally. It must be used 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
);
```

7.1.3.1. Types

`struct JidoshaLightHandle`

Description

The purpose of this structure is to store the client object of a license plate recognition server.

Members

Nenhum

`typedef void JCallback`

Description

The purpose of this type is to define the user callback format for receiving events from the server.

Members

`struct JidoshaLightRecognition rec`: structure where the recognition result will be stored

`int rc`: request return code (see **Function return codes**)

`uint8_t* buffer`: pointer to the image where the recognition was made (this pointer is only valid during the callback execution)

`unsigned int bufferSize`: image size

`void* arg`: pointer to user-supplied opaque structure provided in the handle creation

---

`struct JidoshaLightClientConfig`

Description

The purpose of this structure is to define the parameters of the client-server connection.

Members

`int queueSize`: maximum number of pending requests for this handle.

`const char* ip`: string containing the server's IP address.

`int port`: the server's TCP port.

`JCallback callback`: function that will be called to process results generated by the server.

`void* arg`: pointer to user-supplied opaque structure used for handling server events. This pointer is repassed as a parameter to the user callback.

`struct JidoshaLightServerInfo`

Description

Struct used to store license and version information of a JidoshaLight server.

Members

`JidoshaLightLicenseInfo license` : structure containing information about the server license - see **struct JidoshaLightLicenseInfo**

`int major` : major component of the library version used by the server

`int minor` : minor component of the library version used by the server

`int release` : release component of the library version used by the server

7.1.3.2. Methods

`jl_async_create_handle`

Function prototype

```
JidoshaLightHandle* jl_async_create_handle(
    JidoshaLightClientConfig* config
);
```

Description

Creates the handle of an asynchronous client for connection with a license plate recognition server.

Parameters

`JidoshaLightClientConfig* config`: configuration for this handle.

Return

Returns a pointer to a handle of type `JidoshaLightHandle`, or `NULL` in case of failure.

---

jl_async_destroy_handle

Function prototype

```
int jl_async_destroy_handle(
    JidoshaLightHandle* handle
);
```

Description

Deletes the handle of an asynchronous client, closing the connection to the license plate recognition server.

Parameters

`JidoshaLightHandle* handle`: pointer to a handle created by **jl_async_create_handle**.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jl_async_connect

Function prototype

```
int jl_async_connect(
    JidoshaLightHandle* handle
);
```

Description

Establishes a session with the license plate recognition server for a given handle. This function blocks until the connection is established or a timeout occurs.

Parameters

`JidoshaLightHandle* handle`: pointer to a handle created by **jl_async_create_handle**.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jl_async_connect

Function prototype

```
int jl_async_connect_info(
    JidoshaLightHandle* handle,
    JidoshaLightServerInfo* info
);
```

Description

Same functionality as function **jl_async_connect**, but receives an additional parameter used to store information about the server's license and version.

Parameters

`JidoshaLightHandle* handle`: pointer to a handle created by **jl_async_create_handle**.

`JidoshaLightServerInfo* info`: pointer to a **struct JidoshaLightServerInfo** that will be filled by the function.

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

---

jl_async_get_localqueue_size

Function prototype

```
int jl_async_get_localqueue_size(
    JidoshaLightHandle* handle
);
```

Description

Returns the size of the client-side pending requests queue for a given handle.

Parameters

`JidoshaLightHandle* handle`: pointer to a handle created by **jl_async_create_handle**.

Return

Returns the number of pending requests in the client-side queue.

---

jl_async_ANPR_fromFile

Function prototype

```
int jl_async_ANPR_fromFile(
    JidoshaLightHandle* handle,
    const char* filename,
    JidoshaLightConfig* config
);
```

Description

See description of the **jidoshaLight_ANPR_fromFile** method.

Parameters

`JidoshaLightHandle* handle`: pointer to a handle created by **jl_async_create_handle**.

`const char* filename`: see description of method **jidoshaLight_ANPR_fromFile**.

`JidoshaLightConfig* config`: see description of method **jidoshaLight_ANPR_fromFile**.

Return

See description of method **jidoshaLight_ANPR_fromFile**.

---

jl_async_ANPR_fromMemory

Function prototype

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

Description

See description of method **jidoshaLight_ANPR_fromMemory**.

Parameters

`JidoshaLightHandle* handle`: pointer to a handle created by **jl_async_create_handle**.

`const unsigned char* buffer`: see description of method **jidoshaLight_ANPR_fromMemory**.

`unsigned int bufferSize`: see description of method **jidoshaLight_ANPR_fromMemory**.

`JidoshaLightConfig* config`: see description of method **jidoshaLight_ANPR_fromMemory**.

Return

See description of method **jidoshaLight_ANPR_fromMemory**.

---

jl_async_ANPR_fromLuma

Function prototype

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

Description

See description of method **jidoshaLight_ANPR_fromLuma**.

Parameters

`JidoshaLightHandle* handle`: Pointer to a handle created by **jl_async_create_handle**.

`unsigned char* luma`: See description of method **jidoshaLight_ANPR_fromLuma**.

`int width`: See description of method **jidoshaLight_ANPR_fromLuma**.

`int height`: See description of method **jidoshaLight_ANPR_fromLuma**.

`JidoshaLightConfig* config`: See description of method **jidoshaLight_ANPR_fromLuma**.

Return

See description of method **jidoshaLight_ANPR_fromLuma**.

---

jl_async_ANPR_fromRawImgFmt

Function prototype

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

Description

See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

Parameters

`JidoshaLightHandle* handle`: Pointer to a handle created by **jl_async_create_handle**.

`const unsigned char* buffer`: See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

`int width`: See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

`int height`: See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

`int stride`: See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

`JidoshaLightRawImgFmt fmt`: See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

`JidoshaLightConfig* config`: See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

`JidoshaLightRecognition* rec`: See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

Return

See description of method **jidoshaLight_ANPR_fromRawImgFmt**.

---

jl_async_ANPR_fromImage

Function prototype

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

Description

See description of method **jidoshaLight_ANPR_fromImage**.

Parameters

`JidoshaLightHandle* handle`: Pointer to a handle created by **jl_async_create_handle**

`JidoshaLightImage* img`: See description of method **jidoshaLight_ANPR_fromImage**

`JidoshaLightConfig* config`: See description of method **jidoshaLight_ANPR_fromImage**

Return

See description of method **jidoshaLight_ANPR_fromImage**

---

jl_async_ANPR_multi_fromImage

Function prototype

```
int jl_async_ANPR_multi_fromImage (
   JidoshaLightHandle* handle,
   JidoshaLightImage* img,
   JidoshaLightConfig* config,
   int maxPlates
);
```

Description

Recognizes multiple plates from a previously loaded **JidoshaLightImage**, causing multiple calls to the callback function. A set of recognitions belonging to the same image may be identified by the `int frameId` field in **struct JidoshaLightRecognition**.

See description of method **jidoshaLight_ANPR_multi_fromImage** for more details.

Parameters

`JidoshaLightHandle* handle`: Pointer to a handle created by **jl_async_create_handle**

`JidoshaLightImage* img`: See description of method **jidoshaLight_ANPR_multi_fromImg**

`JidoshaLightConfig* config`: See description of method **jidoshaLight_ANPR_multi_fromImg**

`int maxPlates`: See description of method **jidoshaLight_ANPR_multi_fromImg**

Return

See description of method **jidoshaLight_ANPR_multi_fromImg**

7.1.4. JidoshaLight C/C++ API (Server)

The Server API extends the Local API, allowing the user the create and configure a license plate recognition server for use with remote APIs. It must be used 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
);
```

7.1.4.1. Types

`struct JidoshaLightServer`

Description

The purpose of this structure is to store the license plate recognition server object.

Members

None

`struct JidoshaLightServerConfig`

Description

The purpose of this structure is to configure the library when working as a license plate recognition server.

Members

`int port`: TCP port number used for message exchange.

`int conns`: number of simultaneous connections accepted by the server.

`int threads`: number of parallel processing threads started by the server.

`int threadQueueSize`: maximum size of the request queue for each processing thread.

`int queueTimeout`: maximum waiting time for a request in the request queue, in milliseconds (ms). The value `0` indicates that there is no timeout.

7.1.4.2. Methods

`jidoshaLightServer_create`

Function prototype

```
JidoshaLightServer* jidoshaLightServer_create(
    JidoshaLightServerConfig* serverConfig
);
```

Description

Creates a license plate recognition server instance. Uses the configuration struct pointed by `JidoshaLightServerConfig* serverConfig` and returns a pointer to handler of type `JidoshaLightServer`.

Parameters

`serverConfig`: pointer to `struct JidoshaLightServerConfig` with the server configuration

Return

Returns a pointer to the handle of type `JidoshaLightServer`, or `NULL` in case of failure.

`jidoshaLightServer_destroy`

Function prototype

```
int jidoshaLightServer_destroy(
    JidoshaLightServer* handler
);
```

Description

Deletes the server instance identified by its handler.

Parameters

`handler`: pointer to the server instance

Return

Return code `JIDOSHA_LIGHT_SUCCESS` in case of success, or another code otherwise. (see **Function return codes**).

7.2. JidoshaLight Java API

There are differences between the Linux and the Android™ versions of JidoshaLight's Java API.

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

7.2.1. JidoshaLight Java 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_ARGENTINA              = 32;
    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_MEXICO                 = 484;
    public static final int COUNTRY_CODE_PARAGUAY               = 600;
    public static final int COUNTRY_CODE_PERU                   = 604;
    public static final int COUNTRY_CODE_URUGUAY                = 858;
    public static final int COUNTRY_CODE_NETHERLANDS            = 528;
    public static final int COUNTRY_CODE_FRANCE                 = 250;

    /* 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");
        }

    }
```

7.2.1.1. Types

class JidoshaLightImage

Description

Has the same functionalities as the **struct JidoshaLightImage** function of the C API.

Public methods

```
    public JidoshaLightImage();
```

Builds a new object of type `JidoshaLightImage`. If the native handle allocation fails, a `RuntimeException` is thrown.

To avoid memory leaks, all `JidoshaLightImage` objects must be explicitly destroyed by the user by calling the `destroy()` function.

```
    public JidoshaLightImage duplicate();
```

Duplicates an object of type `JidoshaLightImage` that was previously created and loaded in memory. The new object must be destroyed by the user by calling the `destroy()` function.

```
    public int destroy();
```

Frees the memory allocated by the object.

```
    public int setLazyDecode(boolean enable);
```

See **struct JidoshaLightImage** of the C API.

```
    public int loadFromFile(String filename);
```

See **struct JidoshaLightImage** of the C API.

```
    public int loadFromMemory(byte[] buffer);
```

See **struct JidoshaLightImage** of the C API.

```
    public int loadFromRawImgFmt(byte[] buffer, int width, int height, int stride, int fmt);
```

See **struct JidoshaLightImage** of the C API.

`class JidoshaLight.Config`

Description

Has the same functionalities as the **struct JidoshaLightConfig** function of the C API.

Members

`int vehicleType`: type of plate that the library should search for. Possible values for this field are:

- JidoshaLight.VEHICLE_TYPE_CAR: see JIDOSHA_LIGHT_VEHICLE_TYPE_CAR.
- JidoshaLight.VEHICLE_TYPE_MOTO: see JIDOSHA_LIGHT_VEHICLE_TYPE_MOTO.
- JidoshaLight.VEHICLE_TYPE_BOTH: see JIDOSHA_LIGHT_VEHICLE_TYPE_BOTH.

`int processingMode`: processing strategy to be used. Possible values for this field are:

- JidoshaLight.MODE_DISABLE: see JIDOSHA_LIGHT_MODE_DISABLE.
- JidoshaLight.MODE_FAST: see JIDOSHA_LIGHT_MODE_FAST.
- JidoshaLight.MODE_NORMAL: see JIDOSHA_LIGHT_MODE_NORMAL.
- JidoshaLight.MODE_SLOW: see JIDOSHA_LIGHT_MODE_SLOW.
- JidoshaLight.MODE_ULTRA_SLOW: see JIDOSHA_LIGHT_MODE_ULTRA_SLOW.

`int timeout`: see C API.

`int countryCode`: see C API.

`float minProbPerChar`: see C API.

`int maxLowProbabilityChars`: see C API.

`byte lowProbabilityChar`: see C API.

`float avgPlateAngle`: see C API.

`float avgPlateSlant`: see C API.

`int maxCharHeight`: see C API.

`int minCharHeight`: see C API.

`int maxCharWidth`: see C API.

`int minCharWidth`: see C API.

`int avgCharHeight`: see C API.

`int avgCharWidth`: see C API.

`int xRoi[]` and `int yRoi[]`: x and y coordinates of four points of the Region of Interest (ROI). See C API for more information.

`class JidoshaLight.Recognition`

Description

Concatenates the functionalities of types **struct JidoshaLightRecognition** and **struct JidoshaLightRecognitionInfo** of the C API.

Members

`String plate`: string containing the characters of the recognized license plate, or an empty string if a license plate could not be found.

`float probabilities[]`: see C API.

`int frameId`: see C API.

`int xText` and `int yText`: see C API.

`int widthText`: see C API.

`int heightText`: see C API.

`int xChar[]` and `int yChar[]`: see C API.

`int widthChar[]`: see C API.

`int heightChar[]`: see C API.

`int textColor`: see C API.

`int isMotorcycle`: see C API.

`double totalTime`: see C API.

`double localizationTime`: see C API.

`double segmentationTime`: see C API.

`double classificationTime`: see C API.

`double loadDecodeTime`: see C API.

`int libVersion[]`: see C API.

`String libSHA1[]`: see C API.

`class JidoshaLight.LicenseInfo`

Description

Type used by the **getLicenseInfo** function to return information about the library license.

Members

`String serial` : serial number of the license in decimal base

`String customer` : name of the client that acquired the license

`int maxThreads` : maximum number of enabled processing threads

`int maxConnections` : maximum number of enabled concurrent connections

`int state` : license status (see **Function return codes**)

`int ttl` : time-to-live in hours of RTC (Real Time Clock)-type licenses. If the license has no expiry time, this field has value **-1**

`class JidoshaLight.Version`

Description

Type used by the `getVersion` function to return the library version.

Members

`int major`: major value of the version.

`int minor`: minor value of the version.

`int release`: release value of the version.

7.2.1.2. Methods

`JidoshaLight.ANPR_fromImage [LINUX and ANDROID]`

Function prototype

```
public static native int ANPR_fromImage(
    JidoshaLightImage img,
    Config config,
    Recognition rec
);
```

Description

Has the same behavior as the **jidoshaLight_ANPR_fromImage** function of the C API.

Parameters

`img`: object of type **JidoshaLightImage** containing the image to be recognized.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

`rec`: object of type **JidoshaLight.Recognition** where the recognition result will be stored.

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

`JidoshaLight.ANPR_multi_fromImage [LINUX and ANDROID]`

Function prototype

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

Description

Has the same behavior as the **jidoshaLight_ANPR_multi_fromImage** of the C API.

Parameters

`img`: object of type **JidoshaLightImage** containing the image to be recognized.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

`maxPlates` : maximum number of plates to be recognized (1 to 8).

`recList`: list of objects of type **JidoshaLight.Recognition** where the recognition results will be stored. Has size equal to maxPlates if the function returns successfully.

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

`JidoshaLight.ANPR_fromFile [LINUX]`

Function prototype

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

Description

Has the same behavior as the **jidoshaLight_ANPR_fromFile** function of the C API.

Parameters

`filename`: string containing the path to the image file to be recognized.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

`rec`: object of type **JidoshaLight.Recognition** where the recognition result will be stored.

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

`JidoshaLight.ANPR_fromMemory [LINUX]`

Function prototype

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

Description

Has the same behavior as the **jidoshaLight_ANPR_fromMemory** function of the C API.

Parameters

`buffer`: byte array containing the image.

`bufferSize`: array size in bytes.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

`rec`: object of type **JidoshaLight.Recognition** where the recognition result will be stored.

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

JidoshaLight.ANPR_fromLuma [LINUX]

Function prototype

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

Description

Has the same behavior as the **jidoshaLight_ANPR_fromLuma** function of the C API.

Parameters

`luma`: byte array containing the image in 8-bit grayscale RAW format.

`width`: image width.

`height`: image height.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

`rec`: object of type **JidoshaLight.Recognition** where the recognition result will be stored.

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

JidoshaLight.ANPR_fromBitmap [ANDROID]

Function prototype

```
public static native int ANPR_fromBitmap (
    Bitmap bitmap,
    Config config,
    Recognition rec
);
```

Description

Recognizes a license plate from a `bitmap` object using the configuration present in `config`. The recognition result is returned in `rec`. If an error occurs

during procesing, the `rec` object will contain an empty string for the license plate and a value other than `JidoshaLight.SUCCESS` will be returned by the function. The possible return values are defined in class `JidoshaLight` and have the prefix `ERROR_`.

Parameters

`bitmap`: object of type `android.graphics.Bitmap` containing the image to be recognized in `ARGB8888` format.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

`rec`: object of type **JidoshaLight.Recognition** where the recognition result will be stored.

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

`JidoshaLight.ANPR_fromUri [ANDROID]`

Function prototype

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

Description

Recognizes a plate from an image file `Uri`. Internally, this method calls function `ANPR_fromBitmap`. The recognition result is returned in `rec`. If an error occurs during processing, the `rec` object will contain an empty string for the license plate and a value other than `JidoshaLight.SUCCESS` will be returned by the function. The possible return values are defined in class `JidoshaLight` and have the prefix `ERROR_`.

Parameters

`context`: object of type `android.content.Context` containing the Activity context.

`uri`: object of type `android.net.Uri` containing the `uri` of the image to be recognized.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

`rec`: object of type **JidoshaLight.Recognition** where the recognition result will be stored.

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

`JidoshaLight.getAndroidFingerprint [ANDROID]`

Function prototype

```
static native String getAndroidFingerprint(Activity androidActivity);
```

Description

Returns the unique identifier generated by the library installation.

Parameters

`androidActivity`: object of type `android.app.Activity` containing the reference to the application's main Activity.

Return

String containing the unique identifier needed to generate the license file. This string must not be changed.

`JidoshaLight.getLicenseFromServer [ANDROID]`

Protótipo da Função

```
static native int getLicenseFromServer(Activity activity, String savePath, String user, String key);
```

Descrição

Request a license file from Pumatronix license server.

Parâmetros

`activity`: object of type `android.app.Activity` containing the reference to the application's main Activity.

`savePath`: path to save the received file

`user`: user to be used for the request; `null` otherwise;

`key`: key to be used for the request; `null` otherwise;

Retorno

- JidoshaLight.LICENSE_REQUEST_OK
- JidoshaLight.LICENSE_REQUEST_BAD_REQUEST
- JidoshaLight.LICENSE_REQUEST_NOT_FOUND
- JidoshaLight.LICENSE_REQUEST_UNAUTHORIZED
- JidoshaLight.LICENSE_REQUEST_FORBIDDEN
- JidoshaLight.LICENSE_REQUEST_PAYMENT_REQUIRED
- JidoshaLight.LICENSE_REQUEST_INTERNAL_SERVER_ERROR
- JidoshaLight.LICENSE_REQUEST_SERVICE_UNAVAILABLE
- JidoshaLight.LICENSE_REQUEST_ORIGIN_IS_UNREACHABLE

See Android sample for more information.

`JidoshaLight.setLicenseFromData [ANDROID]`

Function prototype

```
static native int setLicenseFromData(Activity androidActivity, byte[] data, int dataSize);
```

Description

This method is used to configure the library's license file from the contents of the `byte[] data` buffer.

Parameters

`androidActivity`: object of type `android.app.Activity` containing the reference to the application's main Activity.

`data`: buffer with the license file contents.

`dataSize`: size of the license file - `data.len`

Return

Return code `JidoshaLight.SUCCESS` in case of success, or another code otherwise. (see **7.2.1.3. Function return codes**).

`JidoshaLight.getVersion`

Function prototype

```
    public static native int getVersion(Version version);
```

Description

Returns the library version in major.minor.release format.

Parameters

`version` : object of type **JidoshaLight.Version** with the version number

Return

Always returns `JidoshaLight.SUCCESS`.

JidoshaLight.getBuildSHA1

Function prototype

```
String getBuildSHA1();
```

Description

Has the same behavior as the **jidoshaLight_getBuildSHA1** function of the C API.

Parameters

None

Return

Returns a String containing the build's SHA1 hash.

JidoshaLight.getBuildFlags

Function prototype

```
String getBuildFlags();
```

Description

Has the same behavior as the **jidoshaLight_getBuildFlags** function of the C API.

Parameters

None

Return

Returns a String containing the library's build flags.

JidoshaLight.getLicenseInfo

Function prototype

```
public static native int getLicenseInfo(LicenseInfo info);
```

Description

Function used to read information about the license used the JidoshaLight library.

Parameters

`info` : object of type **JidoshaLight.LicenseInfo**

Return

Returns `JIDOSHA_LIGHT_SUCCESS` in case of success.

7.2.1.3. Function return codes

Description

The codes returned the JidoshaLight library functions are defined as `public static final int` attributes inside the `JidoshaLight` class.

Codes returned in Linux and ANDROID versions of the SDK

- `JidoshaLight.ERROR_FILE_NOT_FOUND`: returned by the `ANPR_fromFile` and `ANPR_fromUri` functions when the specified file path does not exist
- `JidoshaLight.ERROR_INVALID_IMAGE`: returned by the `ANPR` functions. Occurs when the image is corrupted.
- `JidoshaLight.ERROR_INVALID_IMAGE_TYPE`: returned by the `ANPR` functions. Occurs when one tries to process an image in an 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 parameters. Occurs when the argument is invalid
- `JidoshaLight.ERROR_COUNTRY_NOT_SUPPORTED`: returned by the `ANPR` functions when the country code supplied in the configuration structure is not supported by the library

- `JidoshaLight.ERROR_API_CALL_NOT_SUPPORTED`: returned when an API function is not supported for a specific platform
- `JidoshaLight.ERROR_INVALID_ROI`: returned when an invalid region of interest is supplied. See the description of `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. This can happen when a call must be preceded by another.
- `JidoshaLight.ERROR_LICENSE_INVALID`: returned by the `ANPR` functions when the supplied license is invalid (for license of the hardkey type, this means the hardkey is not connected or has some issue). Contact Pumatronix for more information
- `JidoshaLight.ERROR_LICENSE_EXPIRED`: returned by the `ANPR` functions when a demonstration-type license has expired. Contact Pumatronix for more information
- `JidoshaLight.ERROR_LICENSE_MAX_THREADS_EXCEEDED`: returned by the `ANPR` functions when the maximum number of concurrent threads exceeds the limit allowed by the license
- `JidoshaLight.ERROR_LICENSE_UNTRUSTED_RTC`: returned by the `ANPR` functions when a license with expiry date has no reliable date and time reference
- `JidoshaLight.ERROR_OTHER`: returned when an unexpected error occurs. Contact Pumatronix for technical support

7.2.2. JidoshaLight Java API (Asynchronous Remote)

Note: All Asynchronous Remote API functions 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();

}
```

7.2.2.1. Types

`class JidoshaLightRemote.Config`

Description

The purpose of this structure is to store the client object of a license plate recognition server.

Members

`queueSize`: maximum number of pending requests for this handle.

`ip`: string containing the server's IP address.

`port`: the server's TCP port.

`interface JidoshaLightRemote.CallBacks`

Description

The purpose of this interface is to define the user callback format for receiving events from the server.

Members

- `on_lpr_result_cb(JidoshaLight.Recognition rec, int code, byte[] buffer)`

> - `rec` : object containing the recognition result
> - `code` : request return code
> - `buffer`: buffer containing the image to be recognized

`class JidoshaLightRemote.ServerInfo`

Description

Type used to store license and version information of a JidoshaLight server.

Members

`JidoshaLight.LicenseInfo license` : object containing information about the server license

`JidoshaLight.Version version` : library version used by the server

`interface JidoshaLightRemote.CallBacks`

7.2.2.2. Methods

`JidoshaLightRemote.create_handle`

Function prototype

```
long create_handle (Config config, CallBacks callbacks);
```

Description

Creates a handle for an asynchronous client for connection with a license plate recognition server. A call to `destroy_handle` must be made to free allocated resources.

Parameters

A `JidoshaLightRemote.Config` object containing the server configuration parameters and an object that implements the `JidoshaLightRemote.CallBacks` interface.

Return

In case of success, returns the memory address of the created handle. Returns `0` otherwise.

`JidoshaLightRemote.destroy_handle`

Function prototype

```
int destroy_handle (long handle);
```

Description

Frees the resources allocated to the handle. To avoid incorrect reuse of an already-released handle, it is recommended that, after this function call, the handle is set to `0`, i.e. `mHandle = 0;`.

Parameters

A `long` containing the memory address of a valid `JidoshaLightRemote` handle.

Return

`JidoshaLight.SUCCESS` in case of success.

JidoshaLightRemote.connect

Function prototype

```
int connect (long handle);
```

Description

Establishes a session with the license plate recognition server for a given handle.

Parameters

`long handle`: long containing the memory address of a valid `JidoshaLightRemote` handle.

Return

`JidoshaLight.SUCCESS` in case of success. Otherwise, see error codes.

JidoshaLightRemote.connect_info

Function prototype

```
int connect_info(long handle, ServerInfo info);
```

Description

Has the same behavior as the **connect** function but has an additional parameter to receive information about the server's license and version.

Parameters

`long handle`: long containing the memory address of a valid `JidoshaLightRemote` handle.

`ServerInfo* info`: object of type **JidoshaLightRemote.ServerInfo**

Return

`JidoshaLight.SUCCESS` in case of success. Otherwise, see error codes.

JidoshaLightRemote.get_localqueue_size

Function prototype

```
int get_localqueue_size (long handle);
```

Description

Returns the size of the client-side pending requests queue for a given handle.

Parameters

A `long` containing the memory address of a valid `JidoshaLightRemote` handle.

Return

Returns the number of pending requests in the local queue (client-side).

JidoshaLightRemote.ANPR_fromMemory

Function prototype

```
int ANPR_fromMemory (
    long handle,
    byte[] buffer,
    JidoshaLight.Config config
);
```

Description

Remote version of the `JidoshaLight.ANPR_fromMemory` function.

Parameters

`handle`: long containing the memory address of a valid `JidoshaLightRemote` handle.

`buffer`: byte array containing the image to be recognized in JPEG, PNG, or BMP format.

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

Return

`JidoshaLight.SUCCESS` in case of success. Otherwise, see error codes.

JidoshaLightRemote.ANPR_fromRawImgFmt

Function prototype

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

Description

Sends a license plate recognition request from an image in RAW format.

Parameters

`handle`: long containing the memory address of a valid `JidoshaLightRemote` handle.

`buffer`: byte array containing the image to be recognized in one of the supported RAW formats (see definitions in class `JidoshaLight`).

`width`: image width

`height`: image height

`stride`: size in bytes of one image row

`fmt`: image format (see definitions in class `JidoshaLight`).

`config`: object of type **JidoshaLight.Config** containing the configuration to be used by the library. Passing a `null` in this parameter implies the use of the default library configuration.

Return

`JidoshaLight.SUCCESS` in case of success. Otherwise, see error codes.

JidoshaLightRemote.getVersion

Function prototype

```
public static native int getVersion(Version version);
```

Description

Returns the library version in major.minor.release format.

Parameters

`version` : object of type `Version` with the version number

Return

Always returns `JidoshaLight.SUCCESS`.

JidoshaLightRemote.getBuildSHA1

Function prototype

```
String getBuildSHA1();
```

Description

Has the same behavior as the **jidoshaLight_getBuildSHA1** function of the C API.

Parameters

None

Return

Returns a String containing the build's SHA1 hash.

JidoshaLightRemote.getBuildFlags

Function prototype

```
String getBuildFlags();
```

Description

Has the same behavior as the **jidoshaLight_getBuildFlags** function of the C API.

Parameters

None

Return

Returns a String containing the library's build flags.

7.2.3. JidoshaLight Java API (Server)

Note: All Server API functions 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();

}
```

7.2.3.1. Types

`class JidoshaLightServer.Config`

Description

Configuration type for a license plate recognition server.

Members

`port`: server connection port.

`conns`: number of simultaneous connections accepted by the server (maximum value limited by the license)

`threads`: maximum number of processing threads the server can use (maximum value limited by the license). The threads are shared among connections

`threadQueueSize`: maximum size of the request queue for each processing thread

`queueTimeout`: maximum waiting time for a request in the request queue, in milliseconds (ms)

7.2.3.2. Methods

`JidoshaLightServer.create_handle`

Function prototype

```
long create_handle (Config config);
```

Description

Creates and initializes a handle to a license plate recognition server. A call to `destroy_handle` must be made to release the allocated resources.

Parameters

A `JidoshaLightServer.Config` object containing the configuration parameters of the server.

Return

In case of success, returns the memory address of the created handle. Otherwise, returns `0`.

JidoshaLightServer.destroy_handle

Function prototype

```
void destroy_handle (long handle);
```

Description

Frees the resources allocated to the handle. To avoid incorrect reuse of an already-released handle, it is recommended that, after this function call, the handle is set to `0`, i.e. `mHandle = 0;`.

Parameters

A `long` containing the memory address of a valid `JidoshaLightServer` handle.

Return

`0` if the handle cannot be created. Otherwise, returns a value different than zero.

JidoshaLightServer.getVersion

Function prototype

```
public static native int getVersion(Version version);
```

Description

Returns the library version in major.minor.release format.

Parameters

`version` : object of `Version` type with the number version

Return

Always returns `JidoshaLight.SUCCESS`.

JidoshaLightServer.getBuildSHA1

Function prototype

```
String getBuildSHA1();
```

Description

Has the same behavior as the **jidoshaLight_getBuildSHA1** function of the C API.

Parameters

None

Return

Returns a String containing the build's SHA1 hash.

JidoshaLightServer.getBuildFlags

Function prototype

```
String getBuildFlags();
```

Description

Has the same behavior as the **jidoshaLight_getBuildFlags** function of the C API.

Parameters

None

Return

Returns a String containing the library's build flags.

7.2.4. JidoshaLight Java API (IO/Mjpeg)

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

> **Note:** All functions of the IO/Mjpeg API 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);

}
```

7.2.4.1. Types

`class Mjpeg.Config`

Description

Configuration type for an Mjpeg stream.

Members

`url`: String containing the Mjpeg stream URL in the `http://<IP>[:PORT]/[PATH]` format.

`timeout`: maximum interval between frames in milliseconds. Delays larger than `timeout` are handled as connection loss. (Recommended values: 1000 to 5000).

`bufferSize`: maximum number of frames that can be queued. This parameter must be larger than **0** and preferably equal to **1**. Values larger than **1** should be considered for cases when the `frame_cb` callback function can take more time than the time interval between stream frames.

`interface Mjpeg.Callbacks`

Description

Interface that defines the callbacks generated by the Mjpeg stream.

> **Note 1:** the callback execution cannot take too long (see parameter `bufferSize`).
>
> **Note 2:** under no circumstances should `destroy_handle(long handle)` be called from inside a callback.

Members

`void frame_cb(byte[] frame)`: callback that is called whenever a new frame is available. The frame is in JPEG format.

`void error_cb(int code)`: callback that is called whenever an error occurs in the Mjpeg stream (see error definition below). Whenever there is a disconnection error, the stream will try to reestablish connection automatically. To interrupt this process, the user should destroy the handle.

7.2.4.2. Methods

`Mjpeg.create_handle`

Function prototype

```
long create_handle (
    Callbacks callbacks,
    Config config
);
```

Description

Creates a handle to be used with function of the `Mjpeg` class. A call to `destroy_handle` must be made to free the allocated resources.

Parameters

An object that implements the `interface Mjpeg.Callbacks` interface and a `Mjpeg.Config` object containing the stream configuration parameters.

Return

In case of success, returns the memory address of the created handle. Otherwise, returns `0`.

`Mjpeg.destroy_handle`

Function prototype

```
void destroy_handle (long handle);
```

Description

Frees the resources allocated to the handle. To avoid incorrect reuse of an already-released handle, it is recommended that, after this function call, the handle is set to `0`, i.e. `mHandle = 0;`.

Parameters

A `long` containing the memory address of a valid `Mjpeg` handle.

Return

`0` if the handle cannot be created. Otherwise, returns a value different than zero.

`Mjpeg.connect`

Function prototype

```
void connect (long handle);
```

Description

Attempts to estasblish a connectino with the URL defined during creationg of the `Mjpeg` handle.

Parameters

A `long` containing the memory address of a valid `Mjpeg` handle.

Return

`JidoshaLight.SUCCESS` in case of success.

`Mjpeg.JL_MJPEG_CONNECT_FAILED` if the connection cannot be established immediately. In this case, the handle will not try to reconnect automatically, and it is up to the user to call `connect` again at an approriate time.

`Mjpeg.get_frame`

Function prototype

```
byte[] get_frame(long handle)
```

```
```

Description

Returns the most recent frame from the `Mjpeg` reception queue. Consecutive calls to this function may return the same frame, if no new frame has been received since the previous call.

Parameters

A `long` containing the memory address of a valid `Mjpeg` handle.

Return

A byte array containing the last frame received in JPEG format. If no frame has been received until the call, the function returns an array of size 0 (`byteArray.length == 0`) and the `error_cb` callback is called with code `JL_LAST_FRAME_UNAVAILABLE`.

7.3. Migration Guide - Jidosha C/C++ API 1

The migration process of a PC application that uses the legacy JIDOSHA API 1 to an application using JidoshaLight is simple and quick. The `lePlaca` function must be replaced by the `jidoshaLight_ANPR_fromFile` function. `struct JidoshaConfig` must be replaced by `struct JidoshaLightConfig` and `struct Reconhecimento` by `struct JidoshaLightRecognition`. The user should mind the new configuration fields in JidoshaLight, which must be filled with the correct values.

The following example shows how to obtain the same behavior of JIDOSHA with JidoshaLight.

JIDOSHA

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

int main(int argc, char* argv[])
{
    Reconhecimento rec;
    JidoshaConfig config;
    config.tipoPlaca = JIDOSHA_TIPO_PLACA_AMBOS;
    config.timeout = 1000;
    lePlaca(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;
    JidoshaLightConfig 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;
}
```

Observations:

- `struct JidoshaLightConfig config` is initialized with zero `{0}`, guaranteeing that the fields `int xRoi[4]` and `int yRoi[4]` are zero and disabling the use of a 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 includes a more detailed sample application.

8. JIDOSHA user APIs

To ease the use and migration to the JidoshaLight library, legacy JIDOSHA APIs are also available through the `libjidoshaCore.so` and `jidoshaCore.dll` libraries. It is possible to switch the JIDOSHA library by these new files and achieve the same behavior. The files with the JIDOSHA interface can be found in the jidoshapc folder inside the Windows or Linux SDK.

The API (Application Programming Interface) of the JIDOSHA library is written in C, which allows it to be used from practically any programming environment. The SDK includes wrapper libraries to simplify use of JIDOSHA from .NET (C# and VB.NET), Java and Delphi. These wrappers simply wrap the calls to the library functions, performing any necessary conversion of parameters and results.

The entire C API can be included with a single header file, `jidoshaCore.h`, whose contents are display below. A detailed description is also shown.

The library can be used in two ways: through API 1 or API 2. The design principle behind API 1 is ease of use. It is possible to read license plates by calling a single function (`lePlaca` or `lePlacaFromMemory` in C language).

API 2, on the other hand, was created to allow more flexibility in configuration and image loading. For instance, it is possible to configure the minimum number of characters that have to be read with good reliability for a license plate to be considered valid. It is possible to add new parameters to API 2 without affecting existing users of the library (that is, those users can update their JIDOSHA DLL/.so to a more recent version without recompiling). Also, API 2 allows the use of RAW images, either grayscale or RGB/BGR. Compatibility with other image formats might be added if necessary.

We recommend API 1 for users who want to integrate JIDOSHA to their application as quickly as possible, and API 2 for those who want more control over the library.

jidoshaCore.h

```
#define JIDOSHA_TIPO_PLACA_CARRO 1 /* recognize only non-motorcycle plates */
    #define JIDOSHA_TIPO_PLACA_MOTO  2 /* recognize only motorcycle plates */
    #define JIDOSHA_TIPO_PLACA_AMBOS 3 /* recognize all plates */

    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_OTHER = 999,
    };

    /* OCR parameters */
    typedef struct JidoshaConfig
    {
        int tipoPlaca;  /* indicates the plate type the OCR should recognize
                           use JIDOSHA_TIPO_PLACA_CARRO,
                               JIDOSHA_TIPO_PLACA_MOTO,
                           or  JIDOSHA_TIPO_PLACA_AMBOS */
        int timeout;    /* timeout in milliseconds */
    } JidoshaConfig;

    /* OCR result */
    typedef struct Reconhecimento
    {
        char placa[8]; /* null-terminated, 7 character plate, or an empty string
                          if the plate was not found */
        double probabilities[7]; /* values from 0.0 to 1.0 indicating the
                                    reliability of each recognized character */
        int xText;      /* left coordinate of the plate text rectangle */
        int yText;      /* top coordinate of the plate text rectangle */
        int widthText;  /* width of the plate text rectangle */
        int heightText; /* height of the plate text rectangle */
        int textColor;  /* text color, 0 - dark, 1 - light */
        int isMotorcycle; /* 0 – non-motorcycle plate, 1 - motorcycle plate */

    } Reconhecimento;

    /* Run OCR on a buffer containing a coded image (JPG, BMP etc)
       returns an empty plate if no hardkey was found or if it is unauthorized */
    int lePlacaFromMemory(const unsigned char* stream, int n, JidoshaConfig* config, Reconhecimento* rec);

    /* Run OCR on an image file
       returns an empty plate if no hardkey was found or if it is unauthorized */
    int lePlaca(const char* filename, JidoshaConfig* config, Reconhecimento* rec);

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

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

    /* Hardkey state
       state == 0   -> unauthorized
       state == 1   -> authorized
       retorno == 0 -> hardkey found
       retorno == 1 -> hardkey not found */
    JIDOSHACORE_API int getHardkeyState(int* state);

    /* Remaining time of demonstration hardkey
       days==-1 and hours==-1: not a demo hardkey (infinite duration)
     */
    int getHardkeyRemainingTime(int* days, int* hours);


    /* API 2 ***************************************************************/
```

```
/* Default API configuration:
    int tipoPlaca               = 3 (JIDOSHA_TIPO_PLACA_AMBOS)
    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        = -30.00
    double avgPlateAngle        = 0.00
    double maxPlateAngle        = 30.00
    double minProbPerCharacter = 0.80
    char lowProbabilityChar     = '*'
*/

/* Recognitions linked list */
typedef struct ResultList
{
    struct ResultList* next;
    struct Reconhecimento* reconhecimento;
} ResultList;

/* Free memory from a result list */
void jidoshaFreeResultList(ResultList* list);

typedef void JidoshaHandle; /* handle for API2 */
typedef void JidoshaImage;  /* image handle for API2 */

/* Initialize API2's handle
   use one handle per thread for multithreaded programs */
JIDOSHACORE_API JidoshaHandle* jidoshaInit();

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

/* Set the value of an integer property */
JIDOSHACORE_API int jidoshaSetIntProperty(JidoshaHandle* handle, const char* name, int value);
/* Read the value of an integer property */
JIDOSHACORE_API int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int* value);

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

/* Set the value of a char property */
JIDOSHACORE_API int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char value);
/* Read the value of a char property */
JIDOSHACORE_API int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char* value);

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

/* Run OCR on 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);

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

/* Load a jpg, bmp of RAW image (grayscale or RGB/BGR)
   from a memory buffer */
JIDOSHACORE_API int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type, int width, int height, JidoshaImag

/* Free a previously loaded 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();
```

8.1.1. JIDOSHA C/C++ API 1

8.1.1.1. Types

`struct JidoshaConfig`

Description

This structure is used to configure the library's behavior when running license plate recognition.

Members

`int tipoPlaca`: indicates the type of plate which the OCR must search. It must be among the following values:

- `JIDOSHA_TIPO_PLACA_CARRO` : only car license plates are searched, where "car" means "non-motorcycle", that is, it includes cars, trucks, buses etc.
- `JIDOSHA_TIPO_PLACA_MOTO` : only motorcycle license plates will be searched.
- `JIDOSHA_TIPO_PLACA_AMBOS`: both motorcycle and non-motorcycle license plates will be searched.

`int timeout`: indicates the maximum time that the plate recognition should take, in milliseconds. A value of zero indicates no timeout. A non-zero timeout is useful in keeping a low average processing time. This value should be determined base on the image resolution and the CPU used.

`struct Reconhecimento`

Description

This structure is used to store results from license plate recognition, including: the plate characters, the reliability of each character, and the coordinates of the plate in the image.

Members

`char placa[8]`: null-terminated, 7 character plate string, or an empty string if the plate was not found.

`double probabilities[7]`: values from 0.0 to 1.0 indicating the reliability of each recognized character.

`int xText` and `int yText`: top-left coordinates of the plate text rectangle.

`int widthText`: width of the plate text rectangle.

`int heightText`: height of the plate text rectangle.

`int textColor`: text color, 0 - dark, 1 - light.

`int isMotorcycle`: 0 - non-motorcycle plate, 1 - motorcycle plate.

8.1.1.2. Methods

`lePlaca`

Function prototype

```
int lePlaca(const char* filename, JidoshaConfig* config, Reconhecimento* rec);
```

Description

Recognizes the license plate and stores it in a `Reconhecimento` object. The image should be passed as a filepath pointing to an image file. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

The image file should be in BMP, JPEG, or PNG format.

Parameters

`filename`: path to the image file.

`config`: pointer to the `JidoshaConfig` struct with the configuration for the library.

`rec`: pointer to the struct `Reconhecimento` that stores the processing result.

Return

Error code: `0` in case of success, otherwise a non-zero number.

---

lePlacaFromMemory

Function prototype

```
int lePlacaFromMemory(const unsigned char* stream, int n, JidoshaConfig* config, Reconhecimento*rec);
```

Description

Recognizes the license plate and stores it in a `Reconhecimento` object. The image should be passed as a byte array, where the number of bytes should be indicated by parameter `n`. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

The image file should be in BMP, JPEG, or PNG format.

Parameters

`stream`: byte array containing the image.

`n`: length of the byte array.

`config`: pointer to the `JidoshaConfig` struct with the library configuration.

`rec`: pointer to the struct `Reconhecimento` where the processing result will be stored.

Return

Error code: `0` in case of success, otherwise a non-zero number.

---

getVersion

Function prototype

```
int getVersion(int* major, int* minor, int* release);
```

Description

Used to get the library version, in major.minor.release format.

Parameters

`major`: pointer to an int variable where the major number will be written.

`minor`: pointer to an int variable where the minor number will be written.

`release`: pointer to an int variable where the release number will be written.

Return

Always returns `0`.

---

getHardkeySerial

Function prototype

```
int getHardkeySerial(unsigned long* serial);
```

Description

Used to get the hardkey serial number.

Parameters

`serial`: pointer to unsigned long variable where the hardkey serial number will be written.

Return

Returns `0` in case of success, `1` if the hardkey was not found.

`getHardkeyState`

Function prototype

```
int getHardkeyState(int* state);
```

Description

Used to get the hardkey state. If `state` is equal to 0, the hardkey is unauthorized; if state is equal to 1, the hardkey is authorized.

Parameters

`state`: pointer to an int variable where the hardkey state will be written.

Return

Returns `0` in case of success, `1` if the hardkey was not found.

`getHardkeyRemainingTime`

Function prototype

```
int getHardkeyRemainingTime(int* days, int* hours);
```

Description

Used to get the remaining time for a demonstration-type license. If days and hours both equal `-1` there is no time limit.

Parameters

`days`: pointer to an int variable that will store the number of days left.

`hours`: pointer to an int variable that will store the number of hours left.

Return

Returns `0` in case of success, `1` if the hardkey was not found.

8.1.2. JIDOSHA C/C++ API 2

8.1.2.1. Types

`struct ResultList`

Description

This structure is used to store the linked list with processing results from the `jidoshaFindFirst` and `jidoshaFindNext` functions.

Members

`struct ResultList* next`: pointer to the next node of the list. `NULL` if the current node is the last.

`struct Reconhecimento* reconhecimento`: pointer to the struct containing one license plate recognition result.

`typedef void JidoshaHandle`

Description

Type used to represent memory allocated for the library configuration.

`typedef void JidoshaImage`

Description

Type used to represent memory allocated for an image.

8.1.2.2. Methods

`jidoshaFreeResultList`

Function prototype

```
void jidoshaFreeResultList(ResultList* list);
```

Description

Frees the memory allocated for the results linked list.

Parameters

`list`: pointer to a `ResultList` struct.

Return

None.

`jidoshaInit`

Function prototype

```
JidoshaHandle* jidoshaInit();
```

Description

Allocates memory for the library configuration. In the multithread case, each thread must call `jidoshaInit` and use its own `JidoshaHandle`.

Parameters

None.

Return

Pointer to a `JidoshaHandle` that must be used when calling other API2 functions.

`jidoshaDestroy`

Function prototype

```
int jidoshaDestroy(JidoshaHandle* handle);
```

Description

Frees the memory allocated by the `jidoshaInit` function.

Parameters

`handle`: pointer to a `JidoshaHandle` previously allocated by `jidoshaInit`.

Return

JIDOSHA_SUCCESS.

`jidoshaSetIntProperty`

Function prototype

```
int jidoshaSetIntProperty(JidoshaHandle* handle, const char* name, int value);
```

Description

Changes the value of an int variable of the library configuration.

Parameters

`handle`: pointer to a `JidoshaHandle`.

`name`: string containing the name of the property to be set.

`value`: value that shall be set to the property.

Return

JIDOSHA_SUCCESS if the value is changed, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type int.

`jidoshaGetIntProperty`

Function prototype

```
int jidoshaGetIntProperty(JidoshaHandle* handle, const char* name, int* value);
```

Description

Reads the value of an int variable from the library configuration.

Parameters

`handle`: pointer to a `JidoshaHandle`.

`name`: string containing the name of the property to be read.

`value`: pointer to an int variable where the value of the property will be stored.

Return

JIDOSHA_SUCCESS if the value is read, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type int.

`jidoshaSetDoubleProperty`

Function prototype

```
int jidoshaSetDoubleProperty(JidoshaHandle* handle, const char* name, double value);
```

Description

Changes the value of a double variable from the library configuration.

Parameters

`handle`: pointer to a `JidoshaHandle`.

`name`: string containing the name of the property to be set.

`value`: value that shall be set to the property.

Return

JIDOSHA_SUCCESS if the value is changed, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type double.

---

jidoshaGetDoubleProperty

Function prototype

```
int jidoshaGetDoubleProperty(JidoshaHandle* handle, const char* name, double* value);
```

Description

Reads the value of a double variable from the library configuration.

Parameters

`handle`: pointer to a `JidoshaHandle`.

`name`: string containing the name of the property to be read.

`value`: pointer to a double variable where the value of the property will be stored.

Return

JIDOSHA_SUCCESS if the value is read, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type double.

---

jidoshaSetCharProperty

Function prototype

```
int jidoshaSetCharProperty(JidoshaHandle* handle, const char* name, char value);
```

Description

Changes the value of a char variable from the library configuration.

Parameters

`handle`: pointer to a `JidoshaHandle`.

`name`: string containing the name of the property to be set.

`value`: value that shall be set to the property.

Return

JIDOSHA_SUCCESS if the value is changed, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type char.

---

jidoshaGetCharProperty

Function prototype

```
int jidoshaGetCharProperty(JidoshaHandle* handle, const char* name, char* value);
```

Description

Reads the value of a char variable from the library configuration.

Parameters

`handle`: pointer to a `JidoshaHandle`.

`name`: string containing the name of the property to be read.

`value`: pointer to a char variable where the value of the property will be stored.

Return

JIDOSHA_SUCCESS if the value is read, JIDOSHA_ERROR_INVALID_PROPERTY if the property does not exist or is not of type char.

---

jidoshaFindFirst

Function prototype

```
int jidoshaFindFirst(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);
```

Description

Recognizes the license plate and stores it in a `Reconhecimento` object inside the first node of the `ResultList`. The image should be loaded with `jidoshaLoadImage` or `jidoshaLoadImageFromMemory` first. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

This function should be called only with an empty `ResultList` object.

Parameters

`handle`: pointer to a `JidoshaHandle` that has the library configuration.

`image`: pointer to a `JidoshaImage` that has the image to be processed.

`list`: pointer to a `ResultList` where the result will be stored.

Return

JIDOSHA_SUCCESS if the function is able to process the image, otherwise it returns another `jidoshaError`.

---

jidoshaFindNext

Function prototype

```
int jidoshaFindNext(JidoshaHandle* handle, JidoshaImage* image, ResultList* list);
```

Description

The purpose of this function is to allow the user to recognize multiple license plates in a single image. Each call will recognize one license plate and store it in a `Reconhecimento` object inside the last node of the `ResultList`. The image should be loaded with `jidoshaLoadImage` or `jidoshaLoadImageFromMemory` first. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

This function should be called only with a `ResultList` already processed with either `jidoshaFindFirst` or `jidoshaFindNext`.

Parameters

`handle`: pointer to a `JidoshaHandle` that has the library configuration.

`image`: pointer to a `JidoshaImage` that has the image to be processed.

`list`: pointer to a `ResultList` where the result will be stored.

Return

JIDOSHA_SUCCESS if the function is able to process the image, otherwise it returns another `jidoshaError`.

---

jidoshaLoadImage

Function prototype

```
int jidoshaLoadImage(const char* filename, JidoshaImage** img);
```

Description

Loads an image from a file and stores a reference as a `JidoshaImage`.

The image file should be in BMP, JPEG, or PNG format.

Parameters

`filename`: path to the image file.

`img`: pointer to the `JidoshaImage` where the loaded image will be stored.

Return

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 is any problem while loading the image.

`jidoshaLoadImageFromMemory`

Function prototype

```
int jidoshaLoadImageFromMemory(const unsigned char* buf, int n, int type, int width, int height, JidoshaImage** img);
```

Description

Loads an image from an array of bytes and stores a reference as a `JidoshaImage`.

The image must be in a structured format (BMP, JPEG, PNG, etc.) or RAW format (8-bit grayscale, RGB, or BGR).

Parameters

`buf`: byte array containing the image.

`n`: size of the byte array.

`type`: type of the image

- Structured type=0, GRAY8=1, RGB=2, BGR=3

`width`: image width, ignored if type==0

`height`: image height, ignored if type==0

`img`: pointer-to-pointer to a `JidoshaImage` where the loaded image will be stored.

Return

JIDOSHA_SUCCESS if the image is correctly loaded, 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 is any problem while loading the image.

`jidoshaFreeImage`

Function prototype

```
int jidoshaFreeImage(JidoshaImage** img);
```

Description

Frees the memory allocated for the image.

Parameters

`img`: pointer-to-pointer to a `JidoshaImage` that will be freed.

Return

JIDOSHA_SUCCESS.

`jidoshaBuildInfo`

Function prototype

```
const char* jidoshaBuildInfo();
```

Description

Check the library build info. Useful to check if the version being used is the right one.

Parameters

None.

Return

Constant string made by 12 or 13 characters that represents the BuildInfo, plus an end of string (`\0`).

`jidoshaNumThreads`

Function prototype

```
int jidoshaNumThreads();
```

Description

Check the number of authorized threads in the hardkey.

Parameters

None.

Return

Integer that represents the number of threads that are authorized to execute the library's license plate recognition functions simultaneously. Returns `1` if no hardkey is found.

8.1.2.3. API 2 - Configuration

This section details the configuration parameters for API 2. It applies to the C, Java, .NET, Delphi, and Python APIs.

Parameter tipoPlaca

Use to restrict the type of vehicle license plate to be recognized. It is used mainly to reduce processing time. In particular, when `tipoPlaca=JIDOSHA_TIPO_PLACA_CARRO`, a faster license plate localization method may be used by the library.

> Name: tipoPlaca
>
> Type: int
>
> Default value: `JIDOSHA_TIPO_PLACA_AMBOS`
>
> Other values:
>
> - `JIDOSHA_TIPO_PLACA_CARRO` == 1
> - `JIDOSHA_TIPO_PLACA_MOTO` == 2
> - `JIDOSHA_TIPO_PLACA_AMBOS` == 3

Parameter timeout

When `timeout` milliseconds have elapsed since the start of license plate recognition call, processing will be interrupted and the best license plate found so far will be returned. If `timeout` is zero, there is no timeout. A `timeout` different than zero is recommended when the application requires that images are processed quickly (low latency) or when the CPU load is too high.

> Name: timeout
>
> Type: int
>
> Default value: 0

Parameter minNumChars

Indicates the minimum number of characters that the license plate should have. If the library version in use multiple plate syntaxes enabled (for

instance, plates from multiple countries), this parameter is ignored, and `numAllowedBadChars` should be used instead.

> Name: minNumChars
>
> Type: int
>
> Default value: 7

Parameter numAllowedBadChars

Indicates the maximum number of missing characters that a license plate can have. This parameter is used when one whishes that partially recognized license plates are returned.

> Name: numAllowedBadChars
>
> Type: int
>
> Default value: 0

Parameter maxNumChars

Indicates the maximum number of characters that a license plate can have. This parameter is ignored.

> Name: maxNumChars
>
> Type: int
>
> Default value: 7

Parameter minCharWidth

Minimum width, in pixels, that a license plate character can have.

> Name: minCharWidth
>
> Type: int
>
> Default value: 1

Parameter avgCharWidth

Average expected character width, in pixels. This parameter is obsolete and no longer used.

> Name: avgCharWidth
>
> Type: int
>
> Default value: 1

Parameter maxCharWidth

Maximum width, in pixels, that a license plate character can have.

> Name: maxCharWidth
>
> Type: int
>
> Default value: 7

Parameter minCharHeight

Minimum height, in pixels, that a license plate character can have.

> Name: minCharHeight
>
> Type: int

Default value: 9

Parameter avgCharHeight

Average expected character height, in pixels. This parameter can be used when plates are very large. When `avgCharHeight > 30`, the image will be downscaled internally before being processed. The minimum and maximum size limits are adjusted accordingly.

Name: avgCharHeight

Type: int

Default value: 20

Parameter maxCharHeight

Maximum character height, in pixels.

Name: maxCharHeight

Type: int

Default value: 60

Parameter ocrModel

Defines the Optical Character Recognition (OCR) model to be used. This parameter exists to easily enable switching the OCR to that of older versions of the library, without needing to recompile the user application or switching the library. Do not use values different than the default, except when recommended by Pumatronix technical support.

Name: ocrModel

Type: int

Default value: 1

Parameter checkSyntax

When `checkSyntax=1` the libarry performs an additional processing step to check that the recognized characters have the expected syntax (letter or number), which reduces the false positive rate (texts that are not plates being recognized as such).

Note: even when `checkSyntax=0`, the library will never return a recognition result with syntax different from the defined one. For example, if the syntax is `LLLNNNN`, the returned plate will always have 3 letters followed by 4 numbers. However, a non-plate text, such as "ESCOLAR", may be confused with a legitimate plate, which would result in a recognition like `ESC0148`. The syntax of that result is correct, though it does not belong to a license plate. Using `checkSyntax=1` helps to filter false recognitions like that.

Name: checkSyntax

Type: int

Default value: 1

Parameter minPlateAngle

Minimum allowed horizontal plate angle, in degrees. For more details, see the section on image perspective configuration.

Name: minPlateAngle

Type: double

Default value: -30.0

Parameter maxPlateAngle

Maximum allowed horizontal plate angle, in degrees. For more details, see the section on image perspective configuration.

Name: maxPlateAngle

Type: double

Default value: 30.0

Parameter minProbPerCharacter

Minimum probability (reliability) for each recognized character. This parameter is extremely important for the correct functioning of JIDOSHA, and changing the default configuration is not recommended. However, in specific cases it may be appropriate to fine-tune it.

If `minProbPerCharacter` is lower than the default value, the number of non-recognized plates will decrease, but the number of incorrectly recognized plates will increase as well.

If `minProbPerCharacter` is higher than the default value, the number of non-recognized plates may increase, but the number of errors will decrease.

Name: minProbPerCharacter

Type: double

Default value: 0.8

Parameter excellentProb

The purpose of this parameter is to reduce the average processing time. If all recognized characters have probability less than or equal to `excellentProb`, the recognition result will be considered excellent and returned immediately to the caller, without additional processing. Otherwise, processing will continue until one of the following conditions is met: an excellent recognition is found; the call has timeouted; or there are no mode processing steps to perform.

Higher values of `excellentProb` result in higher recognition rates and lower error rates (character confusion), but with longer processing times.

Lower values of `excellentProb` result in lower recognition rates and higher error rates (character confusion), but with shorter processing times.

Name: excellentProb

Type: double

Default value: 0.95

Parameter lowProbabilityChar

Substitution character used when a license plate character is recognized with probability less than `minProbPerCharacter`. It will have effect only if parameter `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`.

Name: lowProbabilityChar

Type: char

Default value: '*'

Parameter country

ISO 3166-1 code representing the country to be processed.

For example, if code 32 were used, the plates from Argentina would be processed. Note that the processing availability of a particular country is dependent on the acquired license.

Name: country

Type: int

Default value: 76

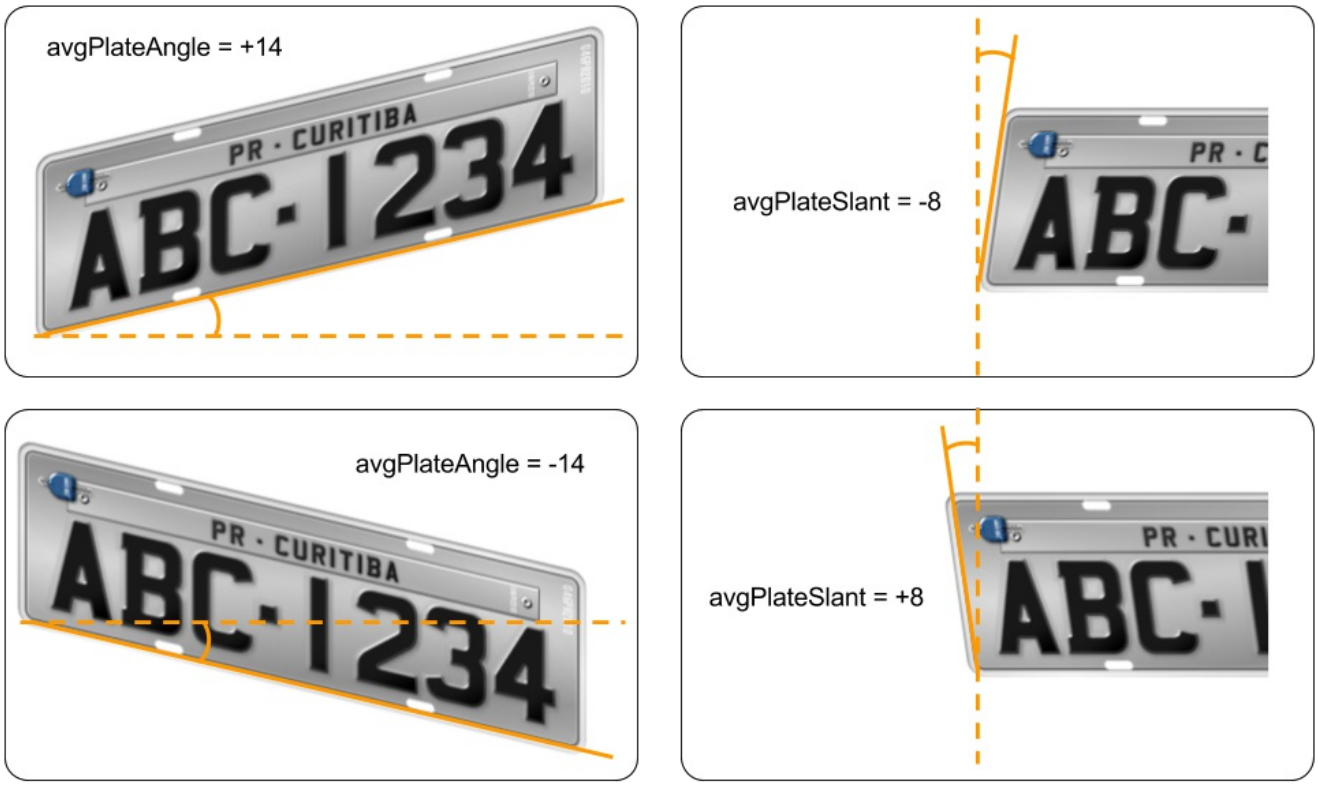
8.1.2.4. API 2 - Image perspective configuration

In general, we recommend that the camera installation for license plate recognition is such that the plates are aligned to the horizontal and vertical image axes. However, in some situations that is not possible, so the plates end up at an angle relative to the image axes, which may be harmful to recognition. In those cases it is possible to inform the plate perspective to the library. The library will then perform perspective correction, in order to maximize the recognition rate.

In the case of an equipment with several cameras, one API 2 `handle` per camera must be created (through the `jidoshaInit` function) and the perspective parameters configured individually for each `handle`.

The `avgPlateAngle`, `avgPlateSlant`, and `adjustPerspective` parameters are used to inform the plate perspective in the image (horizontal and vertical inclinations) and correct it. The horizontal inclination (`avgPlateAngle`) and the vertical inclination (`avgPlateSlant`) must be measured with typical images for each camera installation.

Besides the above manual perspective correction, the library also has algorithms for automatic perspective correction. See parameters `autoSlope` and `autoSlant` for more details.



How to measure `avgPlateAngle` and `avgPlateSlant`

Parameter avgPlateAngle

Expected horizontal inclination angle of the license plate, in degrees. It is used to perform perspective correction in the image. It will only be enabled if `adjustPerspective` is different than zero. The angle must be measured according to the convention in the image shown above.

> Name: avgPlateAngle
>
> Type: double
>
> Default value: 0.0

Parameter avgPlateSlant

Expected vertical inclination angle of the license plate, in degrees. It is used to perform perspective correction in the image. It will only be enabled if `adjustPerspective` is different than zero. The angle must be measured according to the convention in the image shown above.

> Name: avgPlateSlant
>
> Type: double
>
> Default value: 0.0

Parameter adjustPerspective

`adjustPerspective=1` enables manual perspective correction configured by `avgPlateAngle` and `avgPlateSlant`.

`adjustPerspective=0` disables manual perspective correction (`avgPlateAngle` and `avgPlateSlant` are ignored).

> Name: adjustPerspective
>
> Type: int
>
> Default value: 0

Parameter autoSlope

`autoSlope=1` enables automatic correction for the license plate's horizontal angle. If used with manual perspective correction (`avgPlateAngle` when `adjustPerspective=1`), the manual correction will be applied before the automatic correction algorithm.

`autoSlope=0` disables automatic horizontal angle correction.

> Name: autoSlope
>
> Type: int
>
> Default value: 1

Parameter autoSlant

`autoSlant=1` enables automatic correction for the license plate's vertical angle. If used with manual perspective correction (`avgPlateSlant` when `adjustPerspective=1`), the manual correction will be applied before the automatic correction algorithm.

`autoSlant=0` disables automatic vertical angle correction.

> Name: autoSlant
>
> Type: int
>
> Default value: 1

8.2.1. JIDOSHA C# / VB.NET API

The library's .NET API contains three overloaded functions, which make it easy to apply license plate recognition to an image from any of three sources: an array of bytes containing a coded image (JPG, BMP etc.), and object of type `Image`, or a filepath. All of these functions need an object of type `JidoshaConfig`, which is used to configure the behavior of the library.

.NET API 2 has the same functions from C/C++ API 2.

8.2.2. API 1

8.2.2.1. Methods

`reconhecePlaca 1`

Function prototype

```
Reconhecimento reconhecePlaca(byte[] array, JidoshaConfig config)
```

Description

Returns a `Reconhecimento` object which represents the result of the license plate recognition process. The image (JPG, BMP etc.) should be passed as a byte array.

Return

`Reconhecimento` object containing the license plate text string, an array of doubles containing the reliability of each character, the coordinates of the text rectangle, the text color (dark or light), and whether the license plate is that of a motorcycle. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

`reconhecePlaca 2`

Function prototype

```
Reconhecimento reconhecePlaca(Image image, JidoshaConfig config)
```

Description

Returns a `Reconhecimento` object which represents the result of the license plate recognition process. The image should be passed as an `Image` object.

Return

`Reconhecimento` object containing the license plate text string, an array of doubles containing the reliability of each character, the coordinates of the text rectangle, the text color (dark or light), and whether the license plate is that of a motorcycle. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

`reconhecePlaca 3`

Function prototype

```
Reconhecimento reconhecePlaca(string filename, JidoshaConfig config)
```

Description

Returns a `Reconhecimento` object which represents the result of the license plate recognition process. The image should be passed as a filepath pointing to the image file.

Return

`Reconhecimento` object containing the license plate text string, an array of doubles containing the reliability of each character, the coordinates of the text rectangle, the text color (dark or light), and whether the license plate is that of a motorcycle. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

`getVersionString`

Function prototype

```
String getVersionString()
```

Description

Used to get the library version, in the major.minor.release format.

Return

Returns a string formatted with the library version.

`getHardkeySerial`

Function prototype

```
int getHardkeySerial()
```

Description

Used to get the hardkey serial number.

Return

Returns the hardkey serial number.

`getHardkeyState`

Function prototype

```
int getHardkeyState()
```

Description

Used to get the hardkey state. If the return value is equal to `0`, the hardkey is unauhorized; if the return value is equal to `1`, the hardkey is authorized.

Return

Returns the hardkey state, according to the description above.

8.2.2. JIDOSHA C# / VB.NET API Examples

C# Example

```
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 ? "autorizado" : "não autorizado");

                if (args.Length < 1)
                {
                    Console.WriteLine("uso: jidoshaNETSample imagem");
                    Console.WriteLine("Aperte Enter para sair");
                    Console.ReadLine();
                    return;
                }

                // Carrega a imagem
                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.tipoPlaca = TipoPlaca.AMBOS;
                Reconhecimento r = Jidosha.reconhecePlaca(filename, cfg);
                System.Console.WriteLine("reconhecePlaca: {0}", r.placa);
                r = Jidosha.reconhecePlaca(array, cfg);
                System.Console.WriteLine("reconhecePlacaFromMemory: {0}", r.placa);

                // Sample API2

                // Inicializa
                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("Altura media: {0}", maxCharHeight);
                Console.WriteLine("Probabilidade minima: {0}", minProb);

                // Carrega uma imagem
                IntPtr JidoshaImg = Jidosha.jidoshaLoadImage(array, 0, 0, 0);

                // Reconhece placa
                ResultList resultList = new ResultList();
                Jidosha.jidoshaFindFirst(JidoshaHandle, JidoshaImg, ref resultList);
                while (resultList.reconhecimento[resultList.reconhecimento.Count - 1].placa != "")
                {
                    Jidosha.jidoshaFindNext(JidoshaHandle, JidoshaImg, ref resultList);
                }

                // Imprime o resultado
                foreach (Reconhecimento rec in resultList.reconhecimento)
                {
                    Console.WriteLine("Placa: {0}", rec.placa);
                    Console.Write("Probs:");
                    foreach (double d in rec.probabilities)
                        Console.Write(" {0},", d);
                    Console.WriteLine("");
                }

                // Apaga a lista de reconhecimentos
                Jidosha.jidoshaFreeResultList(resultList);

                // Libera a imagem
                Jidosha.jidoshaFreeImage(JidoshaImg);

                // Libera o handle do jidosha
                Jidosha.jidoshaDestroy(JidoshaHandle);

                Console.WriteLine("Aperte Enter para sair");
                Console.ReadLine();
            }
        }
    }
```

VB.NET Example

```
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.tipoPlaca = TipoPlaca.AMBOS
            config.timeout = 1000
            Dim rec As Reconhecimento = Jidosha.reconhecePlaca(filename, config)
            Console.WriteLine("placa: " + rec.placa)
        End Sub

    End Module
```

8.3.1. JIDOSHA Delphi API

8.3.2. API 1

8.3.2.1. Methods

`reconhecePlaca`

Function prototype

```
function reconhecePlaca(filename: String; config: JidoshaConfig) : Reconhecimento;
```

Description

Returns a `Reconhecimento` object which represents the result of the license plate recognition process. The image should be passed as a filepath pointing to the image file.

Return

`Reconhecimento` object containing the license plate text string, an array of doubles containing the reliability of each character, the coordinates of the text rectangle, the text color (dark or light), and whether the license plate is that of a motorcycle. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

`reconhecePlacaFromMemory`

Function prototype

```
function reconhecePlacaFromMemory(byteArray: array of byte; config: JidoshaConfig) : Reconhecimento;
```

Description

Returns a `Reconhecimento` object which represents the result of the license plate recognition process. The image (JPG, BMP etc.) should be passed as a byte array.

Return

`Reconhecimento` object containing the license plate text string, an array of doubles containing the reliability of each character, the coordinates of the text rectangle, the text color (dark or light), and whether the license plate is that of a motorcycle. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

8.3.3. JIDOSHA Delphi API Example

Note: This example is for Delphi 2007. In more recent Delphi versions, it may be necessary to convert the filepath string to `AnsiString` before passing it to the C library. It may also be necesssary to convert the license plate string from `AnsiString` to `Unicode`.

```
program JidoshaDelphiSample;

    {$APPTYPE CONSOLE}

    uses
      SysUtils,
      jidoshaDelphi in 'jidoshaDelphi.pas';

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

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

        config.tipoPlaca := JIDOSHA_TIPO_PLACA_AMBOS;
        config.timeout := 1000;
        rec := reconhecePlaca(filename, config);
        Writeln('placa: ', rec.placa);
    end.
```

8.4.1. JIDOSHA Java API

8.4.2. API 1

8.4.2.1. Methods

`reconhecePlaca`

Function prototype

```
public static native Reconhecimento reconhecePlaca(String filename, JidoshaConfig config);
```

Description

Returns a `Reconhecimento` object which represents the result of the license plate recognition process. The image should be passed as a filepath pointing to the image file.

Return

`Reconhecimento` object containing the license plate text string, an array of doubles containing the reliability of each character, the coordinates of the text rectangle, the text color (dark or light), and whether the license plate is that of a motorcycle. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

`reconhecePlacaFromMemory`

Function prototype

```
public static native Reconhecimento reconhecePlacaFromMemory(byte[] buf, JidoshaConfig config);
```

Description

Returns a `Reconhecimento` object which represents the result of the license plate recognition process. The image (JPG, BMP etc.) should be passed as a byte array.

Return

`Reconhecimento` object containing the license plate text string, an array of doubles containing the reliability of each character, the coordinates of the text rectangle, the text color (dark or light), and whether the license plate is that of a motorcycle. If no plate is found, or if the hardkey is unauthorized or could not be found, the `Reconhecimento` object will contain an empty string as the plate.

8.4.3. JIDOSHA Java API Example

```
import br.com.gaussian.jidosha.Jidosha;
    import br.com.gaussian.jidosha.JidoshaConfig;
    import br.com.gaussian.jidosha.Reconhecimento;

    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]);
                Reconhecimento rec = Jidosha.reconhecePlaca(args[i], config);
                System.out.println("placa: " + rec.placa);
            }
        }
    }
```

8.5.1. Special legacy API builds

For several reasons, JIDOSHA had several different build types for the same version number, which in general are not compatible with each other. The build type can be verified with the function `jidoshaBuildInfo`. 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 compatible only with JIDOSHA's `std` build, which is the standard build type. Since v3.5.0 JidoshaLight is compatible also with the `charpos` build ("character positions"), as long as a certain registry key or environment variable exists, as detailed below. The only difference between builds `std` and `charpos` consists in four additional fields in the `Reconhecimento` struct in header `jidoshaCore.h`, which contain the license plate character coordinates when recognition is successful. Due to this API difference, builds `std` and `charpos` are incompatible with each other (an executable compiled for one of these builds cannot be used with a different build).

> Note: The compatibility mode for `charpos` build is only supported in C language API.

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

std build:

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

charpos build:

```
typedef struct Reconhecimento
{
    char placa[7+1];
    double probabilities[7];
    int xText;
    int yText;
    int widthText;
    int heightText;
    int xChar[7];
    int yChar[7];
    int widthChar[7];
    int heightChar[7];
    int textColor;
    int isMotorcycle;
} Reconhecimento;
```

To enable the `charpos` build compatibility mode in **Windows**, it is necessary to create a key in the Windows registry, in `HKLM\SOFTWARE\PUMATRONIX`, named `JL_LEGACY_API_TYPE`, with type `REG_SZ` and value `charpos`. Any other value will make JidoshaLight return to its standard mode (compatibility with build `std`). Instead of the registry, one can also use an environment variable, with name `JL_LEGACY_API_TYPE` and value `charpos`.

The registry key can be created with the following prompt command (Administrator credentials needed):

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

To disable `charpos` build 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 activate the `charpos` build compatibility mode in **Linux**, it is necesssary to create an environment variable, with name `JL_LEGACY_API_TYPE` and value `charpos`. Any other value will make JidoshaLight return to its standard mode (compatibility with build `std`)

Notes:

- If the `charpos` build compatibility mode is enabled (`JL_LEGACY_API_TYPE=charpos`), but by mistake the user code is using struct `Reconhecimento` from the `std` build, invalid memory access or even silent data corruption may occur.
- We recommend migrating legacy code to JidoshaLight's API as soon as possible.

9. Known limitations

JidoshaLight has the following known limitations:

1. When the library is loaded dynamically (`LoadLibrary` in Windows, `dlopen` in Linux), the library must not be unloaded (`FreeLibrary` and `dlclose`, respectively).
2. In Linux, the library's process must not be forked.
3. When JidoshaLight is used concurrently and in the same process as Pumatronix's container or rail OCR libraries, JidoshaLight must be loaded last. This limitation will be removed in a future version of the libraries.

API Index

C/C++

- enum JidoshaLightCountryCode
- enum JidoshaLightMode
- enum JidoshaLightRawImgFmt
- enum JidoshaLightVehicleType
- getHardkeyRemainingTime
- getHardkeySerial
- getHardkeySerial
- getHardkeyState
- getHardkeyState
- getVersion
- getVersionString
- jidoshaBuildInfo
- jidoshaDestroy
- jidoshaFindFirst
- jidoshaFindNext
- jidoshaFreeImage
- jidoshaFreeResultList
- jidoshaGetCharProperty
- jidoshaGetDoubleProperty
- jidoshaGetIntProperty
- jidoshaInit
- jidoshaLight_ANPR_createImage
- jidoshaLight_ANPR_createList
- jidoshaLight_ANPR_destroyImage
- jidoshaLight_ANPR_destroyList
- jidoshaLight_ANPR_duplicateList
- jidoshaLight_ANPR_duplicateList
- jidoshaLight_ANPR_fromFile
- jidoshaLight_ANPR_fromImage
- jidoshaLight_ANPR_fromLuma
- jidoshaLight_ANPR_fromMemory
- jidoshaLight_ANPR_fromRawImgFmt
- jidoshaLight_ANPR_getListElement
- jidoshaLight_ANPR_getListSize
- jidoshaLight_ANPR_loadImageFromFile
- jidoshaLight_ANPR_loadImageFromMemory
- jidoshaLight_ANPR_loadImageFromRawImgFmt
- jidoshaLight_ANPR_multi_fromImage
- jidoshaLight_ANPR_setImageLazyDecode
- jidoshaLight_getBuildFlags
- jidoshaLight_getBuildSHA1
- jidoshaLight_getLicenseInfo
- jidoshaLight_getReturnCodeString
- jidoshaLight_getVersion
- jidoshaLight_isRemoteApi
- jidoshaLight_setRemoteSyncServerIp
- jidoshaLightServer_create
- jidoshaLightServer_destroy
- jidoshaLoadImage
- jidoshaLoadImageFromMemory
- jidoshaNumThreads
- jidoshaSetCharProperty
- jidoshaSetDoubleProperty
- jidoshaSetIntProperty
- jl_async_ANPR_fromFile
- jl_async_ANPR_fromImage

- jl_async_ANPR_fromLuma
- jl_async_ANPR_fromMemory
- jl_async_ANPR_fromRawImgFmt
- jl_async_ANPR_multi_fromImage
- jl_async_connect
- jl_async_connect
- jl_async_create_handle
- jl_async_destroy_handle
- jl_async_get_localqueue_size
- lePlaca
- lePlacaFromMemory
- reconhecePlaca
- reconhecePlaca
- reconhecePlaca 1
- reconhecePlaca 2
- reconhecePlaca 3
- reconhecePlacaFromMemory
- reconhecePlacaFromMemory
- struct JidoshaConfig
- struct JidoshaLightClientConfig
- struct JidoshaLightConfig
- struct JidoshaLightHandle
- struct JidoshaLightImage
- struct JidoshaLightLicenseInfo
- struct JidoshaLightRecognition
- struct JidoshaLightRecognitionInfo
- struct JidoshaLightRecognitionList
- struct JidoshaLightServer
- struct JidoshaLightServerConfig
- struct JidoshaLightServerInfo
- struct Reconhecimento
- struct ResultList
- typedef void JCallback
- typedef void JidoshaHandle
- typedef void JidoshaImage

JAVA

- class JidoshaLight.Config
- class JidoshaLight.LicenseInfo
- class JidoshaLight.Recognition
- class JidoshaLight.Version
- class JidoshaLightImage
- class JidoshaLightRemote.Config
- class JidoshaLightRemote.ServerInfo
- class JidoshaLightServer.Config
- class Mjpeg.Config
- interface JidoshaLightRemote.CallBacks
- interface JidoshaLightRemote.CallBacks
- interface Mjpeg.Callbacks
- JidoshaLight.ANPR_fromBitmap [ANDROID]
- JidoshaLight.ANPR_fromFile [LINUX]
- JidoshaLight.ANPR_fromImage [LINUX and ANDROID]
- JidoshaLight.ANPR_fromLuma [LINUX]
- JidoshaLight.ANPR_fromMemory [LINUX]
- JidoshaLight.ANPR_fromUri [ANDROID]
- JidoshaLight.ANPR_multi_fromImage [LINUX and ANDROID]
- JidoshaLight.getAndroidFingerprint [ANDROID]
- JidoshaLight.getBuildFlags
- JidoshaLight.getBuildSHA1
- JidoshaLight.getLicenseFromServer [ANDROID]
- JidoshaLight.getLicenseInfo
- JidoshaLight.getVersion
- JidoshaLight.setLicenseFromData [ANDROID]

- JidoshaLightRemote.ANPR_fromMemory
- JidoshaLightRemote.ANPR_fromRawImgFmt
- JidoshaLightRemote.connect
- JidoshaLightRemote.connect_info
- JidoshaLightRemote.create_handle
- JidoshaLightRemote.destroy_handle
- JidoshaLightRemote.get_localqueue_size
- JidoshaLightRemote.getBuildFlags
- JidoshaLightRemote.getBuildSHA1
- JidoshaLightRemote.getVersion
- JidoshaLightServer.create_handle
- JidoshaLightServer.destroy_handle
- JidoshaLightServer.getBuildFlags
- JidoshaLightServer.getBuildSHA1
- JidoshaLightServer.getVersion
- Mjpeg.connect
- Mjpeg.create_handle
- Mjpeg.destroy_handle
- Mjpeg.get_frame