PUMATRONIX

CLASSIFIER

CLASSIFIER

SOFTWARE FOR AUTOMATIC RECOGNITION OF VEHICLE CHARACTERISTICS

Integration

Pumatronix Equipamentos Eletrônicos Ltda.

Rua Bartolomeu Lourenço de Gusmão, 1970. Curitiba, Brasil

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

Send comments about this document to email suporte@pumatronix.com

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

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

Pumatronix grants permission to download and print this document, provided that the electronic or physical copy of this document contains the full text. Any changes to this content are strictly prohibited.

Change History

Date	Revision	Updated content
07/23/2024	1.0	Review of the layout and general formatting of the document; Content referring to version 1.16.0 of the product

Overview

This document aims to guide the developer in the application of the Classifier software library responsible for classifying vehicle types based on image analysis and applicable to software compatible with the library. This document details the configuration options for the software development kit (SDK) and the APIs available.

Summary

1. Unpacking the SDK

The Classifier SDK is distributed via a compressed file in 7zip format. Unzipping this file requires a password, provided along with the email containing download instructions for this SDK. If you have problems with unpacking the SDK, contact support by email contato@pumatronix.com.br or WhatsApp +55 (41) 9203-8327.

To unzip the SDK in a Linux environment, use the following command from a terminal (replace the fields <Classifier_PC_LINUX_64_vX.Y.Z.7z> and with the password provided by email and the correct package name).

```
7z x -p< PASSWORD_PROVIDED> <Classifier_PC_LINUX_64_vX.Y.Z.7z>
```

Hardkey Permissions

This configuration is only required for the Linux version of the SDK.

For the USB hardkey to work correctly on Linux, the udev access permissions must be changed. Add the following line:

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

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

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

If the distribution is Debian, add the lines:

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

at the end of the file:

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

For instructions on how to enable the hardkey on other Linux distributions, contact your contato@pumatronix.com.br ou WhatsApp +55 (41) 9203-8327.

Environment variables

Configuration of LD_LIBRARY_PATH

This configuration is only required for the Linux version of the SDK. For the Windows version, make a copy of the DLL to the folder where the executable is located or to the system32 folder

For the application to be able to find the Classifier libraries, the SDK lib directory path must be added to the PATH. In a Linux environment this can be done through the environment variable `LD_LIBRARY_PATH`, as shown below (replace with the absolute path of the SDK installation).

```
export LD_LIBRARY_PATH=/lib
```

2. SDK Structure

Classifier was not designed as a standalone application but rather as a library to be integrated into other applications. Therefore, the Classifier SDK is distributed as a set of shared libraries (.so and .dll) and their headers in C language.

The SDK also comes with an example application (source code and pre-compiled binary) that can be used as a basis for implementing an application that uses the Classifier. The Basic Example section of this manual also provides a step-by-step guide on how to implement an application.

Linux version file tree

```
include
      ptx
            classifier
                  ptx_classifier_api_local.h
            common
                  ptx_codes.h
                  ptx_defines.h
                  ptx_dict.h
                  ptx_image.h
            ptx_classifier.h
            ptx_common.h
lib
      libptx_classifierJava.so
      libptx_classifier.so
      libptx_commonJava.so
      libptx_common.so
sample
      bin
            libptx_classifierJava.so
            libptx_classifier.so
            libptx_commonJava.so
            libptx_common.so
            PtxClassifierSample
      src
            PtxClassifierSample.c
      wrapper
            java
                  br
                        com
                              pumatronix
                                    classifier
                                          PtxClassifier.java
                                          PtxClassifierSample.java
                                    common
                                          PtxCommon.java
                                          PtxDict.java
                                          PtxImage.java
```

```
      python
            PtxClassifier.py
            PtxClassifierSample.py
            PtxCommonLoader.py
            PtxCommon.py
            PtxDict.py
            PtxImage.py
```

Windows version file tree

```
include
      ptx
            classifier
                  ptx_classifier_api_local.h
            common
                  ptx_codes.h
                  ptx_defines.h
                  ptx_dict.h
                  ptx_image.h
            ptx_classifier.h
            ptx_common.h
lib
      ptx_classifier.dll
      ptx_classifierJava.dll
      ptx_classifierJava.lib
      ptx_classifier.lib
      ptx_common.dll
      ptx_commonJava.dll
      ptx_commonJava.lib
      ptx_common.lib
res
      manual_classifier.pdf
sample
      bin
            PtxClassifierSample.exe
      src
            PtxClassifierSample.c
wrapper
      delphi
            ptx
                  common
                        ptx_common.pas
      java
            br
                  com
                        pumatronix
                              classifier
                                    PtxClassifier.java
                                    PtxClassifierSample.java
                              common
                                    PtxCommon.java
                                    PtxDict.java
                                    PtxImage.java
      python
            PtxClassifier.py
            PtxClassifierSample.py
            PtxCommonLoader.py
            PtxCommon.py
```

```
            PtxDict.py
            PtxImage.py
```

3. Software architecture

Classifier has a simple architecture. All API calls are synchronous, meaning they block until processing is complete. The main function is ptx_classifier_classify, which receives a previously loaded image and a handle containing the configuration and, at the end of processing, fills in a results structure.

The library configuration consists of: type of scene (panoramic or closed), and the minimum probability for a detection to be considered a valid vehicle.

The results consist of a set of detected vehicles, each with the following associated data: vehicle class, classification reliability in the form of probability, and coordinates in the image of the rectangle containing the vehicle.

4. API usage example

For a complete example, see the file `sample/src/PtxClassifierSample.c` from the SDK.

To compile the example application with gcc and run it, run the following sequence of commands from a Linux terminal:

```
gcc sample/src/PtxClassifierSample.c -I include/ -L lib/ -l ptx_classifier -l ptx_common -o
PtxClassifierSample LD_LIBRARY_PATH=lib/ ./PtxClassifierSample
```

Once the application is launched, it should appear in the terminal output:

```
[PUMA] JidoshaClassifier - Client Sample

[PUMA] JidoshaClassifier library Version: X.Y.Z
[PUMA] JidoshaClassifier library SHA1: 0123456789abcdef0123456789abcdef01234567
[PUMA] ptx_common library SHA1: 123456789abcdef0123456789abcdef012345678
[PUMA] LicenseInfo - Serial: 123456789 - maxThreads: 6 - maxConnections: 0
[PUMA] LicenseInfo - State: 0 - TTL: -1 - Customer: Pumatronix
[PUMA] ./PtxClassifierSample
```

5. User APIs

The Classifier library exports an API for automatic vehicle feature recognition. By default, the languages supported by the API that come with the SDK are C, C++, and Java. Wrappers in Python, C# and Delphi can be provided on demand. If you have any questions or support for other languages, send an email to contato@pumatronix.com.br or WhatsApp +55 (41) 9203-8327. To standardize common structures used by its products, Pumatronix created the ptx_common library, which implements data structures, error codes and other commonly used definitions in C. The ptx_common library is required to use Classifier and is included in the SDK. Documentation for your API can be found in the ptx_common API section C/C++.

API Classifier C/C++

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

The API contains the types, definitions and functions for processing images. It is defined in the file `ptx_classifier_api_local.h`.

ptx_classifier_common.h

```
typedef enum PtxClassifierConfigs {
      PTX_CLASSIFIER_CONFIG_SCENE_TYPE                          = 0,
      PTX_CLASSIFIER_CONFIG_VEHICLE_TYPE_MIN_PROB               = 1,
      PTX_CLASSIFIER_CONFIG_MODEL_TYPE                          = 2,
      PTX_CLASSIFIER_CONFIG_ENABLE_VEHICLE_CHARACTERISTICS      = 3,
      PTX_CLASSIFIER_CONFIG_ENUM_MAX
} PtxClassifierConfigs;

typedef enum PtxClassifierSceneType {
      PTX_CLASSIFIER_SCENE_TYPE_CLOSEUP                         = 0, // Default scene
      PTX_CLASSIFIER_SCENE_TYPE_PANORAMIC                       = 1,
      PTX_CLASSIFIER_SCENE_TYPE_PANORAMIC_NIGHT                 = 2,
      PTX_CLASSIFIER_SCENE_TYPE_WIDERANGE                       = 3,
      PTX_CLASSIFIER_SCENE_TYPE_ENUM_MAX
} PtxClassifierSceneType;

typedef enum PtxClassifierModelType {
      PTX_CLASSIFIER_MODEL_TYPE_VEHICLES                        = 0,
      PTX_CLASSIFIER_MODEL_TYPE_BMC                             = 1,
      PTX_CLASSIFIER_MODEL_TYPE_WIND_SHIELD                     = 2, // Model not supported
yet, in development
      PTX_CLASSIFIER_MODEL_TYPE_VEHICLES_EXTENDED               = 3,
      PTX_CLASSIFIER_MODEL_TYPE_ENUM_MAX
} PtxClassifierModelType;

typedef enum PtxClassifierObjExtras {
      // Upper left
      PTX_CLASSIFIER_OBJ_UL_PT_X            = 0,
      PTX_CLASSIFIER_OBJ_UL_PT_Y            = 1,

      // Lower right
      PTX_CLASSIFIER_OBJ_LR_PT_X            = 2,
      PTX_CLASSIFIER_OBJ_LR_PT_Y            = 3,

      // Type
      PTX_CLASSIFIER_OBJ_TYPE_CLASS         = 4,
} PtxClassifierObjExtras;

//============================================================================
// Results
//============================================================================

// Prediction types
typedef enum PtxClassifierResultType {
      PTX_CLASSIFIER_GET_INT_RESULTS_SIZE                       = 0,
```

```
	PTX_CLASSIFIER_GET_PTXDICT_AT_RESULT                    = 1,

	PTX_CLASSIFIER_GET_INT_VEHICLE_TYPE_CLASS               = 2,
	PTX_CLASSIFIER_GET_FLOAT_VEHICLE_TYPE_PROB              = 3,
	PTX_CLASSIFIER_GET_INT_AT_VEHICLE_X_POINT               = 4,
	PTX_CLASSIFIER_GET_INT_VEHICLE_X_POINTS_SIZE            = 5,
	PTX_CLASSIFIER_GET_INT_AT_VEHICLE_Y_POINT               = 6,
	PTX_CLASSIFIER_GET_INT_VEHICLE_Y_POINTS_SIZE            = 7,

	PTX_CLASSIFIER_GET_INT_PROCESSING_TIME                  = 8,

	PTX_CLASSIFIER_GET_PTXSTRING_VEHICLE_COLOR_NAME         = 9,
	PTX_CLASSIFIER_GET_FLOAT_VEHICLE_COLOR_PROB             = 10,
	PTX_CLASSIFIER_GET_PTXSTRING_VEHICLE_BRAND_NAME         = 11,
	PTX_CLASSIFIER_GET_FLOAT_VEHICLE_BRAND_PROB             = 12,
	PTX_CLASSIFIER_GET_PTXSTRING_VEHICLE_MODEL_NAME         = 13,
	PTX_CLASSIFIER_GET_FLOAT_VEHICLE_MODEL_PROB             = 14,

	// Custom output
	PTX_CLASSIFIER_GET_PTXSTRING_OBJECT_TYPE_CLASS          = 15,
	PTX_CLASSIFIER_GET_FLOAT_OBJECT_TYPE_PROB               = 16,
	PTX_CLASSIFIER_GET_INT_AT_OBJECT_X_POINT                = 17,
	PTX_CLASSIFIER_GET_INT_OBJECT_X_POINTS_SIZE             = 18,
	PTX_CLASSIFIER_GET_INT_AT_OBJECT_Y_POINT                = 19,
	PTX_CLASSIFIER_GET_INT_OBJECT_Y_POINTS_SIZE             = 20,
	PTX_CLASSIFIER_GET_INT_OBJECT_ATTRIBUTES_SIZE           = 21,
	PTX_CLASSIFIER_GET_PTXDICT_AT_OBJECT_ATTRIBUTE          = 22,

	PTX_CLASSIFIER_GET_ENUM_MAX
} PtxClassifierResultType;

typedef enum PtxClassifierVehicleType {
	PTX_CLASSIFIER_VEHICLE_TYPE_UNKNOWN                     = 0,
	PTX_CLASSIFIER_VEHICLE_TYPE_CAR                         = 1,
	PTX_CLASSIFIER_VEHICLE_TYPE_MOTORCYCLE                  = 2,
	PTX_CLASSIFIER_VEHICLE_TYPE_TRUCK                       = 3,
	PTX_CLASSIFIER_VEHICLE_TYPE_BUS                         = 4,
	PTX_CLASSIFIER_VEHICLE_TYPE_PICKUP                      = 5,
	PTX_CLASSIFIER_VEHICLE_TYPE_SUV                         = 6,
	PTX_CLASSIFIER_VEHICLE_TYPE_VAN                         = 7,
	PTX_CLASSIFIER_VEHICLE_TYPE_TOW                         = 8,

	PTX_CLASSIFIER_VEHICLE_TYPE_ENUM_MAX
} PtxClassifierVehicleType;
```

ptx_classifier_api_local.h

```
#include "ptx/common/ptx_defines.h"
#include "ptx/common/ptx_codes.h"
#include "ptx/common/ptx_image.h"
#include "ptx/common/ptx_dict.h"
#include "ptx/classifier/ptx_classifier_common.h"

//==============================================================================
// PtxClassifierHandle
//==============================================================================
typedef struct PtxClassifierHandle PtxClassifierHandle;
```

```
//=============================================================================
// FUNCTIONS
//=============================================================================

/* Classifier functions */
PTXEXPORT int PTXAPI ptx_classifier_init();
PTXEXPORT int PTXAPI ptx_classifier_destroy();
PTXEXPORT int PTXAPI ptx_classifier_get_version(int* major, int* minor, int* release);
PTXEXPORT const char* PTXAPI ptx_classifier_get_SHA1();

/* Classifier handle */
PTXEXPORT PtxClassifierHandle* PTXAPI ptx_classifier_create_handle();
PTXEXPORT int PTXAPI ptx_classifier_free_handle(PtxClassifierHandle* handle);
PTXEXPORT int PTXAPI ptx_classifier_set_config(PtxClassifierHandle* handle, PtxDict*
config);

/* Classifier processing */
PTXEXPORT int PTXAPI ptx_classifier_classify(PtxClassifierHandle* handle, PtxImage* img,
PtxDict* results);

/* License */
PTXEXPORT int PTXAPI ptx_classifier_get_license_info(PtxProductLicenseInfo* license);
```

Types

enum PtxClassifierConfigs	
Description	Defines the available library settings. Settings are changed via the ptx_classifier_set_config function.
Members	`PTX_CLASSIFIER_CONFIG_SCENE_TYPE`: type of scene configuration to use for classification. `PTX_CLASSIFIER_CONFIG_VEHICLE_TYPE_MIN_PROB`: configuration of the minimum probability of return allowed by the Classifier. Classifications with a probability lower than this will be discarded internally by the library and will not be returned to the user. `PTX_CLASSIFIER_CONFIG_MODEL_TYPE`: configuration of the type of model to be used. `PTX_CLASSIFIER_CONFIG_ENABLE_VEHICLE_CHARACTERISTICS`: configuration to enable/disable reading of characteristics.

enum PtxClassifierSceneType	
Description	Defines the scenes that can be configured in the Classifier, allowing the library to function better depending on the installation.
Members	`PTX_CLASSIFIER_SCENE_TYPE_CLOSEUP`: Type of scene in which the vehicle occupies more than 80% of the image. `PTX_CLASSIFIER_SCENE_TYPE_PANORAMIC`: Type of daytime scene where vehicles occupy less than 60% of the image. `PTX_CLASSIFIER_SCENE_TYPE_PANORAMIC_NIGHT`: Type of night scene where vehicles occupy less than 60% of the image. `PTX_CLASSIFIER_SCENE_TYPE_WIDERANGE`: Type of scene where vehicles do not have legible license plates 3 lanes or more.

For vehicles occupying between 60% and 80% of the image, it is interesting to try each type of scene and check which one performs best.

enum PtxClassifierModelType	
Description	Defines the type of model that can be used in Classifier.
Members	`PTX_CLASSIFIER_MODEL_TYPE_VEHICLES`: Type of model that works on vehicle detection. `PTX_CLASSIFIER_MODEL_TYPE_VEHICLES_EXTENDED`: Type of model that works on vehicle detection, enabling the extension of classes for panoramic and daytime scenes (pickup, SUV, van and trailer). `PTX_CLASSIFIER_MODEL_TYPE_BMC`: Model type that receives vehicle type detection and classifies make, model and color. `PTX_CLASSIFIER_MODEL_TYPE_WIND_SHIELD`: Type of model that performs windshield detection (Under development, may not be available in your version).

enum PtxClassifierObjExtras	
Description	Defines the extra structures that can be placed in the image to be used by the Classifier.
Members	`PTX_CLASSIFIER_OBJ_UL_PT_X`: Sets the top left x point of an object to be defined in the image (`PtxImage`). `PTX_CLASSIFIER_OBJ_UL_PT_Y`: Sets the top left y point of an object to be defined in the image (`PtxImage`). `PTX_CLASSIFIER_OBJ_LR_PT_X`: Sets the bottom right x point of an object to be defined in the image (`PtxImage`). `PTX_CLASSIFIER_OBJ_LR_PT_Y`: Sets the bottom right y point of an object to be defined in the image (`PtxImage`). `PTX_CLASSIFIER_OBJ_TYPE_CLASS`: Defines the class type of an object to be defined in the image(`PtxImage`).

enum PtxClassifierResultType	
Description	Keys to request the values of vehicle properties from the Classifier (type `PtxDict`).
Members	`PTX_CLASSIFIER_GET_INT_RESULTS_SIZE`: Key to request the value `int` the number of results returned. `PTX_CLASSIFIER_GET_PTXDICT_AT_RESULT`: Key to request the value `PtxDict` of a given position containing a single result. `PTX_CLASSIFIER_GET_INT_VEHICLE_TYPE_CLASS`: Key to request the value `int` representing the type of vehicle found. `PTX_CLASSIFIER_GET_FLOAT_VEHICLE_TYPE_PROB`: Key to request the value `float` which represents reliability in probability form for the type of vehicle. `PTX_CLASSIFIER_GET_INT_VEHICLE_X_POINTS_SIZE`: Key to request the value `int` which represents how many x values are used to describe the contour of the object. `PTX_CLASSIFIER_GET_INT_AT_VEHICLE_X_POINT`: Key to request the value `int` which corresponds to a single value x belonging to the contour of the object. `PTX_CLASSIFIER_GET_INT_VEHICLE_Y_POINTS_SIZE`: Key to request the value `int` which represents how many y values are used to describe the contour of the object. `PTX_CLASSIFIER_GET_INT_AT_VEHICLE_Y_POINT`: Key to request the value `int` which corresponds to a single y value belonging to the contour of the object. `PTX_CLASSIFIER_GET_INT_PROCESSING_TIME`: Key to request the value `int` which corresponds to the Classifier processing time. `PTX_CLASSIFIER_GET_PTXSTRING_VEHICLE_COLOR_NAME`: Key to request the value `string` that corresponds to the color name of the vehicle. `PTX_CLASSIFIER_GET_FLOAT_VEHICLE_COLOR_PROB`: Key to request the value `float` which corresponds to the probability of the vehicle color. `PTX_CLASSIFIER_GET_PTXSTRING_VEHICLE_BRAND_NAME`: Key to request the value `string` that matches the vehicle brand name. `PTX_CLASSIFIER_GET_FLOAT_VEHICLE_BRAND_PROB`: Key to request the value `float` which corresponds to the probability of the vehicle brand. `PTX_CLASSIFIER_GET_PTXSTRING_VEHICLE_MODEL_NAME`: Key to request the value `string` that corresponds to the vehicle model.

	`PTX_CLASSIFIER_GET_FLOAT_VEHICLE_MODEL_PROB`: Key to request the value `float` which corresponds to the probability of the vehicle model. `PTX_CLASSIFIER_GET_PTXSTRING_OBJECT_TYPE_CLASS`: Key to request the value `string` that matches the class name of the object. `PTX_CLASSIFIER_GET_FLOAT_OBJECT_TYPE_PROB`: Key to request the value `float` which corresponds to the probability of the object class. `PTX_CLASSIFIER_GET_INT_AT_OBJECT_X_POINT`: Key to request the value `int` which corresponds to a single value x belonging to the contour of the object. `PTX_CLASSIFIER_GET_INT_OBJECT_X_POINTS_SIZE`: Key to request the value `int` which represents how many x values are used to describe the contour of the object. `PTX_CLASSIFIER_GET_INT_AT_OBJECT_Y_POINT`: Key to request the value `int` which corresponds to a single y value belonging to the contour of the object. `PTX_CLASSIFIER_GET_INT_OBJECT_Y_POINTS_SIZE`: Key to request the value `int` which represents how many y values are used to describe the contour of the object. `PTX_CLASSIFIER_GET_INT_OBJECT_ATTRIBUTES_SIZE`: Key to request the value `int` which corresponds to the number of attributes of the object. `PTX_CLASSIFIER_GET_PTXDICT_AT_OBJECT_ATTRIBUTE`: Key to request the value `PtxDict` of a given position containing a single attribute.

enum PtxClassifierVehicleType	
Description	Fields representing the possible returns of the key `PTX_CLASSIFIER_GET_INT_VEHICLE_TYPE_CLASS`, that is, the possible vehicle types returned by the library.
Members	***`PTX_CLASSIFIER_VEHICLE_TYPE_UNKNOWN:`*** Type of vehicle that could not be categorized ***`PTX_CLASSIFIER_VEHICLE_TYPE_CAR:`*** Car type vehicle ***`PTX_CLASSIFIER_VEHICLE_TYPE_MOTORCYCLE:`*** Motorcycle type vehicle ***`PTX_CLASSIFIER_VEHICLE_TYPE_TRUCK:`*** Truck type vehicle ***`PTX_CLASSIFIER_VEHICLE_TYPE_BUS:`*** Bus type vehicle ***`PTX_CLASSIFIER_VEHICLE_TYPE_PICKUP:`*** Pickup type vehicle ***`PTX_CLASSIFIER_VEHICLE_TYPE_SUV:`*** SUV type vehicle ***`PTX_CLASSIFIER_VEHICLE_TYPE_VAN:`*** Van type vehicle ***`PTX_CLASSIFIER_VEHICLE_TYPE_TOW:`*** Trailer type vehicle

struct PtxClassifierHandle	
Description	Structure that contains the internal settings and states of the library.

Methods

ptx_classifier_init	
Function Prototype	`int ptx_classifier_init();`
Description	Function used to initialize the library after it is loaded.
Parameters	None
Return	An integer with function return code.

ptx_classifier_destroy	
Function Prototype	`int ptx_classifier_destroy();`
Description	Function used to download the library.
Parameters	None
Return	An integer with function return code.

ptx_classifier_get_version	
Function Prototype	`int ptx_classifier_get_version(int* major, int* minor, int* release)`
Description	Function used to query the library version. Classifier follows a major.minor.release semantic versioning system. The major number is increased when the version has an API break in relation to the previous version. If there is no API break and new functionality is included, the minor number is increased. If there is no API break or new functionality included, the release number is increased (this is the case for bug fixes and small changes).
Parameters	`int *major, int *minor, int *release`: pointers to integer variables where the numbers that make up the version will be written.
Return	An integer with function return code.

ptx_classifier_get_SHA1	
Function Prototype	`const char* ptx_classifier_get_SHA1();`
Description	Function used to check the SHA1 hash of the library build. This string is used by Pumatronix for build tracking.
Parameters	None
Return	Returns a pointer to the beginning of a string ending in `\0` containing the SHA1 hash of the build

ptx_classifier_create_handle	
Function Prototype	`PtxClassifierHandle* ptx_classifier_create_handle();`
Description	Function used to allocate memory for library configuration. In case of multithreaded use, each thread must call `ptx_classifier_create_handle` and use your own `PtxClassifierHandle`. The same `PtxClassifierHandle` can be used as many times as necessary, as long as only one thread is used at a time. Library settings, made using the function *ptx_classifier_set_config*, are stored in the `PtxClassifierHandle`. Therefore, each `PtxClassifierHandle` may have a different configuration. This can be useful, for example, when you want to use the Classifier library to process images from different cameras, and you want to apply different settings to images from different cameras.
Parameters	None
Return	Returns a pointer to a struct type *PtxClassifierHandle* which will be used in subsequent function calls.

ptx_classifier_free_handle	
Function Prototype	`int ptx_classifier_free_handle(PtxClassifierHandle* handle);`
Description	Function used to free the memory allocated to the object of type *PtxClassifierHandle*.
Parameters	`PtxClassifierHandle* handle`: Pointer to the type's handle *PtxClassifierHandle*.
Return	An integer with function return code.

ptx_classifier_set_config	
Function Prototype	`int ptx_classifier_set_config(PtxClassifierHandle* handle, PtxDict* config);`
Description	Function used to apply the configuration made through a field `PtxDict`.

Parameters	`PtxClassifierHandle* handle`: Pointer for cable type *PtxClassifierHandle* `PtxDict* config`: Pointer for cable type *PtxDict*
Return	An integer with function return code.

ptx_classifier_classify	
Function Prototype	`int ptx_classifier_classify(PtxClassifierHandle* handle, PtxImage* img, PtxDict* results);`
Description	Function used to process and extract information from the image and return it through the parameter `results`. **This function can be quite time-consuming, taking hundreds of milliseconds to complete or even longer, depending on the CPU used.**
Parameters	`PtxClassifierHandle* handle`: Pointer to the type's handle *PtxClassifierHandle* `PtxImage* img`: Pointer to type image *PtxImage* `PtxDict* results`: Pointer to results of type *PtxDict*
Return	An integer with function return code.

ptx_classifier_get_license_info	
Function Prototype	`int ptx_classifier_get_license_info(PtxProductLicenseInfo* license);`
Description	Function used to request license information regarding the Classifier product.
Parameters	`PtxProductLicenseInfo* license:` Pointer to object of type *PtxProductLicenseInfo*
Return	An integer with function return code.

API ptx_common C/C++

The ptx_common library implements data structures, error codes and other definitions commonly used by Pumatronix products in C. Your API is defined in the file `ptx_common.h`, which itself includes other headers.

ptx_codes.h

```
//============================================================================
// RETURN CODES
//============================================================================
enum PtxCommonReturnCode {

     /* API CALL RETURN CODES */
     /* SUCCESS */
     PTX_SUCCESS                                = 0,

     /* BASIC ERRORS */
     PTX_FILE_NOT_FOUND                         = 1,
     PTX_INVALID_IMAGE                          = 2,
     PTX_INVALID_IMAGE_TYPE                     = 3,
     PTX_INVALID_PARAMETER                      = 4,
     PTX_COUNTRY_NOT_SUPPORTED                  = 5,
     PTX_API_CALL_NOT_SUPPORTED                 = 6,
     PTX_INVALID_ROI                            = 7,
     PTX_INVALID_HANDLE                         = 8,
     PTX_API_CALL_HAS_NO_EFFECT                 = 9,
     PTX_INVALID_IMAGE_SIZE                     = 10,
```

```
	PTX_MODEL_UNAVALIABLE                          = 11,

	/* LICENSE ERRORS */
	PTX_LICENSE_INVALID                            = 16,
	PTX_LICENSE_EXPIRED                            = 17,
	PTX_LICENSE_MAX_THREADS_EXCEEDED               = 18,
	PTX_LICENSE_UNTRUSTED_RTC                      = 19,
	PTX_LICENSE_MAX_CONNS_EXCEEDED                 = 20,
	PTX_LICENSE_UNAUTHORIZED_PRODUCT               = 21,

	/* NETWORK ERRORS */
	PTX_CONNECT_FAILED                             = 100,
	PTX_SOCKET_DISCONNECT                          = 101,
	PTX_SOCKET_QUEUE_TIMEOUT                       = 202,
	PTX_SOCKET_QUEUE_FULL                          = 103,
	PTX_SOCKET_IO_ERROR                            = 104,
	PTX_SOCKET_WRITE_FAILED                        = 105,
	PTX_SOCKET_READ_TIMEOUT                        = 106,
	PTX_INVALID_RESPONSE                           = 107,
	PTX_HANDLE_QUEUE_FULL                          = 108,
	PTX_INVALID_REQUEST                            = 109,
	PTX_INVALID_MESSAGE                            = 110,
	PTX_INVALID_STREAM_FRAME                       = 209,
	PTX_FRAME_QUEUE_FULL                           = 211,
	PTX_LAST_FRAME_UNAVAILABLE                     = 212,
	PTX_SERVER_CONN_LIMIT_REACHED                  = 213,
	PTX_SERVER_VERSION_NOT_SUPPORTED               = 214,

	/* MJPEG ERRORS */
	PTX_MJPEG_ERROR_BASE                           = 1000,
	PTX_MJPEG_HTTP_HEADER_OVERFLOW                 = 1001,
	PTX_MJPEG_HTTP_RESPONSE_NOT_OK                 = 1002,
	PTX_MJPEG_HTTP_CONTENT_TYPE_ERROR              = 1003,
	PTX_MJPEG_HTTP_CONTENT_LENGTH_ERROR            = 1004,
	PTX_MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND        = 1005,
	PTX_MJPEG_CONNECTION_CLOSED = 1006,
	PTX_MJPEG_CONNECT_FAILED                       = 1007,

	/* HARDWARE ERRORS - returned by the process */
	PTX_HW_FPGA_INIT_FAILED                        = 2000,
	PTX_HW_FPGA_LOCK_FAILED                        = 2001,

	/* OTHERS */
	PTX_UNKNOWN_ERROR                              = 99999
};
typedef struct PtxProductLicenseInfo
{
	uint64_t serial;
	char customer[64];
	int maxThreads;
	int maxConnections;
	int state;
	int ttl;
} PtxProductLicenseInfo;
PTXEXPORT const char* PTXAPI ptx_common_get_SHA1();
);
```

Types

enum PtxCommonReturnCode	
Description	Sets library error codes ptx_common.
Members	The various members of this enumeration are returned by functions in different types of situations: success, invalid image, invalid parameter, license errors, network errors, etc.

enum PtxProductLicenseInfo	
Description	Struct used to store information about the license used by the library
Members	• `uint64_t serial`: license serial number • `char customer[64]`: name of the customer who purchased the license • `int maxThreads`: maximum number of processing threads enabled • `int maxConnections`: maximum number of parallel connections enabled • `int state`: license status – see *PtxCommonReturnCode* • `int ttl`: time-to-live in hours for RTC type licenses. This field has the value -1 if the license is not expirable

Methods

ptx_common_get_SHA1	
Function Prototype	`const char* PTXAPI ptx_common_get_SHA1();`
Description	Function used to check the SHA1 hash of the library build ptx_common. This string is used by Pumatronix for build tracking.
Parameters	None
Return	Returns a pointer to the beginning of a string ending in `\0` containing the SHA1 hash of the build.

ptx_dict.h

This file defines a dictionary data structure, used to store key-value relationships. The key is always of type int. Each Pumatronix product that uses the ptx_common library lists in its header which keys it uses.

```
//============================================================================
// PtxDict
//============================================================================
typedef struct PtxDict PtxDict;

/* lifecycle */
PTXEXPORT PtxDict* PTXAPI ptxdict_create();
PTXEXPORT int PTXAPI ptxdict_free(PtxDict* ptx_dict);

/* Setters */
PTXEXPORT int PTXAPI ptxdict_set_int(PtxDict* ptx_dict, int key, int value);
PTXEXPORT int PTXAPI ptxdict_set_float(PtxDict* ptx_dict, int key, float value);

/* Getters */
PTXEXPORT int PTXAPI ptxdict_get_int(PtxDict* ptx_dict, int key, int* value);
PTXEXPORT int PTXAPI ptxdict_get_float(PtxDict* ptx_dict, int key, float* value);
PTXEXPORT int PTXAPI ptxdict_get_ptxstring(PtxDict* ptx_dict, int key, PtxString* value);
PTXEXPORT int PTXAPI ptxdict_get_int_at(PtxDict* ptx_dict, int key, int index, int* value);
PTXEXPORT int PTXAPI ptxdict_get_ptxdict_at(PtxDict* ptx_dict, int key, int index, PtxDict*
value);
```

Types

struct PtxDict	
Description	Struct opaque file used to store the dictionary.
Members	None, as it is an opaque structure.

Methods

ptxdict_create	
Function Prototype	`PtxDict* PTXAPI ptxdict_create();`
Description	Function used to allocate memory for a dictionary.
Parameters	None
Return	Returns a pointer to a struct type [PtxDict](#struct-ptxdict] that will be used in subsequent function calls.

ptxdict_free	
Function Prototype	`int ptxdict_free(PtxDict* ptx_dict);`
Description	Function used to free the memory allocated to the object of type *PtxDict*.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict* whose memory will be freed.
Return	An integer with function return code.

ptxdict_set_int	
Function Prototype	`int ptxdict_set_int(PtxDict* ptx_dict, int key, int value);`
Description	Function used to store a key-value pair.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict* where the key-value pair will be stored. `int key`: type key int. `int value`: value of type int that will be stored as associated with the `key`.
Return	An integer with function return code.

ptxdict_set_float	
Function Prototype	`int ptxdict_set_float(PtxDict* ptx_dict, int key, float value);`
Description	Function used to store a key-value pair.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict* where the key-value pair will be stored. `int key`: type key int. `float value`: value of type int that will be stored as associated with the `key`
Return	An integer with function return code.

ptxdict_get_int	
Function Prototype	`int ptxdict_get_int(PtxDict* ptx_dict, int key, int* value);`

Description	Function used to request a value associated with a key.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict*. `int key`: key of type int whose value you want to request. `int* value`: value of type int that will be stored as associated with the `key`, if it exists.
Return	An integer with function return code.

ptxdict_get_float	
Function Prototype	`int ptxdict_get_float(PtxDict* ptx_dict, int key, float* value);`
Description	Function used to request a value associated with a key.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict*. `int key`: key of type int whose value you want to request. `float value`: pointer to int where the value associated with the `key` will be written, if it exists.
Return	An integer with function return code.

ptxdict_get_ptxstring	
Function Prototype	`int ptxdict_get_ptxstring(PtxDict* ptx_dict, int key, PtxString* value);`
Description	Function used to request a string associated with a key.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict*. `int key`: key of type int whose value you want to request. `PtxString* value`: pointer to object of type *PtxString* where the value associated with the `key` will be written, if it exists.
Return	An integer with function return code.

ptxdict_get_int_at	
Function Prototype	`int ptxdict_get_int_at(PtxDict* ptx_dict, int key, int index, int* value);`
Description	Function used to request the value at a given index of a vector associated with a key.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict*. `int key`: key of type int whose value you want to request. `int index`: desired index of the vector associated with `key.` `int* value`: pointer to float where the value will be written, if any.
Return	An integer with function return code.

ptxdict_get_ptxdict_at	
Function Prototype	`int ptxdict_get_ptxdict_at(PtxDict* ptx_dict, int key, int index, PtxDict* value);`
Description	Function used to request the value at a given index of a vector associated with a key.
Parameters	`PtxDict* dict`: Pointer to object of type *PtxDict*. `int key`: key of type int whose value you want to request. `int index`: desired index of the vector associated with `key.`

	`PtxDict* value`: pointer to *PtxDict* where the value will be written, if it exists.
Return	An integer with function return code.

ptx_image.h

This file defines a structure for loading images. Supports both structured images (jpg and bmp) and RAW images.

```
//============================================================================
// Types
//============================================================================

typedef struct PtxImage PtxImage;

/* Raw image pixel format */
typedef enum PtxImagePixelFmt {
     PTX_IMG_FMT_XRGB_8888 = 0,
     PTX_IMG_FMT_RGB_888 = 1,
     PTX_IMG_FMT_LUMA = 2,
     PTX_IMG_FMT_YUV420 = 3,
     PTX_IMG_FMT_YUV_NV12 = 4
} PtxImagePixelFmt;

//============================================================================
// Functions
//============================================================================

/* lifecycle */
PTXEXPORT PtxImage* PTXAPI ptximage_create();
PTXEXPORT int PTXAPI ptximage_free(PtxImage* img);

/* load */
PTXEXPORT int PTXAPI ptximage_load_from_file(PtxImage* img, const char* filename);
PTXEXPORT int PTXAPI ptximage_load_from_memory(PtxImage* img, const uint8_t* buffer, int
bufferSize);
PTXEXPORT int PTXAPI ptximage_load_from_raw_format(PtxImage* img, const uint8_t* buffer,
int width, int height, int stride, PtxImagePixelFmt fmt)

/* setters */
PTXEXPORT int PTXAPI ptximage_append_extra(PtxImage* img, PtxDict* extra);
PTXEXPORT int PTXAPI ptximage_clear_extras(PtxImage* img);
```

Types

struct PtxImage	
Description	Struct opaque file used to load images.
Members	None, as it is an opaque structure.

enum PtxImagePixelFmt	
Description	Enumeration of RAW image types that can be loaded.
Members	None, as it is an opaque structure.

Methods

ptximage_create	
Function Prototype	`PtxImage* ptximage_create();`
Description	Function used to allocate memory for an image structure.
Parameters	None
Return	Returns a pointer to a struct type [PtxImage](#struct-ptximage] that will be used in subsequent function calls.

ptximage_free	
Function Prototype	`int ptximage_free(PtxImage* img);`
Description	Function used to free memory allocated to an object of type *PtxImage*.
Parameters	`PtxImage* img`: Pointer to object of type *PtxImage* whose memory will be freed.
Return	An integer with function return code.

ptximage_load_from_file	
Function Prototype	`int ptximage_load_from_file(PtxImage* img, const char* filename);`
Description	Function used to load a structured image (jpg or bmp) from a file.
Parameters	`PtxImage* img`: Pointer to object of type *PtxImage* where the image will be stored in memory. `const char* filename`: File containing a structured image to be loaded.
Return	An integer with function return code.

ptximage_load_from_memory	
Function Prototype	`int ptximage_load_from_memory(PtxImage* img, const uint8_t* buffer, int bufferSize);`
Description	Function used to load a structured image (jpg or bmp) from a buffer in memory.
Parameters	`PtxImage* img`: Pointer to object of type *PtxImage* where the image will be stored in memory. `const uint8* buffer`: Buffer containing the structured image to be loaded. `int bufferSize`: `Buffer` size in bytes.
Return	An integer with function return code.

ptximage_load_from_raw_format	
Function Prototype	`int ptximage_load_from_raw_format(PtxImage* img, const uint8_t* buffer, int width, int height, int stride, PtxImagePixelFmt fmt)`
Description	Function used to load a RAW image from a memory buffer. The supported image types are defined by *PtxImagePixelFmt*.
Parameters	`PtxImage* img`: Pointer to object of type *PtxImage* where the image will be stored in memory. `const uint8* buffer`: Buffer containing the RAW image to be loaded. `int width`: Image width in pixels.

	`int height`: Image height in pixels. `int stride`: Number of bytes between a line and the next line of the image. `PtxImagePixelFmt fmt`: RAW image type, as per *PtxImagePixelFmt*.
Return	An integer with function return code.

ptximage_append_extra	
Function Prototype	`int ptximage_append_extra(PtxImage* img, PtxDict* extra);`
Description	Function to add an extra *PtxDict* to the image structure.
Parameters	`PtxImage* img`: Pointer to object of type *PtxImage*. `PtxDict* extra`: Pointer to the *PtxDict* extra to be added to the image.
Return	An integer with function return code.

ptximage_clear_extras	
Function Prototype	`int ptximage_clear_extras(PtxImage* img);`
Description	Function used to clear added extras.
Parameters	`PtxImage* img`: Pointer to object of type *PtxImage*.
Return	An integer with function return code.

ptx_string.h

```
Este arquivo define uma estrutura para ler textos.
//============================================================================
// PtxString
//============================================================================
typedef struct PtxString PtxString;

/* lifecycle */
PTXEXPORT PtxString* PTXAPI ptxstring_create();
PTXEXPORT int PTXAPI ptxstring_free(PtxString* ptx_string);

/* Setters */
PTXEXPORT int PTXAPI ptxstring_set(PtxString* ptx_string, const char* str);

/* Getters */
PTXEXPORT const char* PTXAPI ptxstring_get(PtxString* ptx_string);
```

Types

struct PtxString	
Description	Struct opaque used to hold texts.
Members	None, as it is an opaque structure.

Methods

ptxstring_create	
Function Prototype	`PtxString* ptxstring_create();`

Description	Function used to allocate memory for a text structure.
Parameters	None
Return	Returns a pointer to a *PtxString* struct that will be used in subsequent function calls.

ptxstring_free	
Function Prototype	`int ptxstring_free(PtxString* ptx_string);`
Description	Function used to free memory allocated to an object of type *PtxString*.
Parameters	`PtxString* ptx_string`: Pointer to object of type *PtxString* whose memory will be freed.
Return	An integer with function return code.

ptxstring_set	
Function Prototype	`int ptxstring_set(PtxString* ptx_string, const char* str);`
Description	Function used to copy the characters of a const char* into the structure *PtxString*.
Parameters	`PtxString* ptx_string`: Pointer to object of type *PtxString*. `const char* str`: Pointer to text in format const char*.
Return	An integer with function return code.

ptxstring_get	
Function Prototype	`const char* ptxstring_get(PtxString* ptx_string);`
Description	Function used to retrieve the characters of a const char* from within the structure *PtxString*.
Parameters	`PtxString* ptx_string`: Pointer to object of type *PtxString*.
Return	Pointer to text in format const char*.


PUMATRONIX
www.pumatronix.com