Intertraff ANPR Engine SDK Documentation

Introduction **4**

SDK licensing **4**
- “Parking” version (slow version) 4
- “FreeFlow” version (high speed traffic) 4
- Demo version 4
- SDK license transfer 4

Architecture overview **5**

Supported programming languages **5**

Hardware requirements **5**

Supported image / video formats **6**

Supported video sources **6**

Supported countries **6**

Dataset resolution **7**

Crop region **9**

Synchronous and Asynchronous **10**

Benchmarks **11**

Sample applications **12**
- Overview 12
- VideoSample example 13
- VideoAsinc example 14
- PlateRecognize example 15
- Multicameras example 15

Getting started **16**
- How to install the SDK 16
- Adding the SDK to your project 16

Reference **16**
- Enumerators 16
  - NNANPR_Device 16
  - NNANPR_PlateType 16
  - NNANPR_ImageFormat 16
  - NNANPR_GeographicalArea 16
- Structures 17
  - NNANPR_Object 17
  - NNANPR_RecognizerResult 17

Functions 17
NANPR_CreatePlateFinder 17
NNANPR_SetupPlateFinder 18
NANPR_SetPlateFinderCroppingRegion 18
NNANPR_GetPlateFinderSpecificNames 19
NNANPR_FreePlateFinder 19
NNANPR_CreatePlateRecognizer 19
NNANPR_SetupPlateRecognizer 19
NNANPR_GetPlateRecognizerSpecificNames 20
NNANPR_FreePlateRecognizer 20
NNANPR_CreateImage 20
NNANPR_FreeImage 20
NNANPR_CloneImage 21
NNANPR_LoadImage 21
NNANPR_SaveImage 21
NNANPR_DrawRect 21
NNANPR_GetImageData 22
NNANPR_CreateImageWindow 22
NNANPR_FreeImageWindow 22
NNANPR_DrawImage 23
NNANPR_ImageWindowStartDrawCrop 23
NNANPR_ImageWindowStopDrawCrop 23
NNANPR_ImageWindowShowCrop 24
NNANPR_WaitImageWindow 24
NNANPR_CreateVideoCapture 24
NNANPR_GetSpecialVideoCaptureList 25
NNANPR_CreateSpecialVideoCapture 25
NNANPR_FreeVideoCapture 26
NNANPR_CaptureFrame 26
NNANPR_StartCaptureFrameAsync 26
NNANPR_StopCaptureFrameAsync 26
NNANPR_GetVideoFPS 27
NNANPR_FindPlates 27
NNANPR_FindPlatesAsync 27
NNANPR_ReconizePlate 28
NNANPR_ReconizePlateAsync 28
NNANPR_CreateRecognitionAccumulator 29
NNANPR_SetupRecognitionAccumulator 29
NNANPR_FreeRecognitionAccumulator 30
NNANPR_ReconizePlateAsyncWithAccumulator 30
NNANPR_GetErrorString 31

Introduction

The purpose of this SDK is to facilitate the creation of different ANPR (Automatic Number Plate Recognition) systems or to add the ANPR functionality to other projects for Windows 10 OS. The SDK contains a set of programming examples written in different programming languages which can be used as a starting point.

At any time the SDK installation folder can be accessed by selecting the `Intertraff / ANPR Engine SDK Folder` menu option of the Windows Start menu.

SDK licensing

After the software is installed on the target PC, the customer requires a license ID and a password to activate it. At the time of activation the target PC should have reliable Internet connection. To activate the SDK just start the VideoAsync example application or `Activator-Deactivator.exe` program (in the `bin` subfolder of the SDK installation folder) and enter provided credentials to the appeared activation dialog.

There are three SDK license options:

“Parking” version (slow version)

This version cannot process image or video files. Just live video sources. And there will be 3 seconds blackout time between successive plate recognition results.

“FreeFlow” version (high speed traffic)

Full featured version without any restrictions.

Demo version

It is the full featured version with time limitation. The license expires at the end of every month but the customer can get a new one.

SDK license transfer

To move the license on another PC the customer needs to choose the `Intertraff / License Transfer` menu option of the Windows Start menu. Then he will be asked if he wants to “Deactivate license ID xxx”. After deactivation he will be able to install the SDK on another PC using the same ID and password he’s got.

This operation can be done only for a certain number of times. Then you need to contact Intertraff to be able to re-transfer the license on another machine.

Architecture overview

ANPR engine internally consists of two distinct parts: Plate Finder and Plate Recognizer. The Plate Finder searches for all car license plates on submitted images or video frames and returns bounding boxes of all found objects together with its confidence levels (probabilities that the found object is a real license plate). The Plate Recognizer takes the particular bounding box on the image and returns the license plate text string. For some countries, it also returns the country and region within other useful information concerning the recognized license plate (see `NNANPR_RecognizerResult` structure in the Reference section for more information).

There is an option to assign the particular device to execute the Plate Finding and Plate Recognizer operations. These can be: CPU, Intel GPU, Intel Movidius Myriad Processing Unit, NVidia GPU or external online server (only for Plate Recognition operation).

There are different varieties of the Plate Finder and the Plate recognizer even inside one geographical area (see `NNANPR_GetPlateFinderSpecificNames` and `NNANPR_GetPlateRecognizerSpecificNames` functions). As a rule the bigger the number in the variety name the more precise operation is and the more time it takes.

At API level the SDK provides a set of functions to create, manipulate, use and destroy the following objects through obscure pointers:

- image objects;
- image window objects;
- video capture objects;
- plate finder objects;
- plate recognizer objects;
- accumulator of recognition results objects;

Supported programming languages

Currently the SDK targets Visual Studio 2017 C++ / C# and Delphi 10 programming languages but other versions may work too.

Hardware requirements

The SDK shall be installed on Windows 10 64bit O/S and requires either Intel 64 bit CPU, Intel GPU and / or NVidia GPU that supports CUDA version 11 or higher. Nvidia GPU can also be used without CUDA but the performance may be 50% slower. The SDK can also be executed on Intel Movidius (Myriad) Processing Unit. The PC should be equipped with at least 4GB RAM.

Supported image / video formats

The SDK can read the following image formats:

- Windows bitmaps - *.bmp, *.dib
- JPEG files - *.jpeg, *.jpg, *.jpe
- JPEG 2000 files - *.jp2
- Portable Network Graphics - *.png
- Portable image format - *.pbm, *.pgm, *.ppm *.pxm, *.pnm
- TIFF files - *.tiff, *.tif

The list of supported video formats depends on the video codecs installed in the operating system. We recommend to install the K-Lite Codec Pack.

Supported video sources

The SDK can work with the following video sources: video files, IP-cameras (identified by URL) and Industrial cameras manufactured by the following brands: Basler, Flir (formerly Point Grey Research Inc.), IDS and Lucid Vision Labs (see `NNANPR_GetSpecialVideoCaptureList` function for details).
If you wish to operate our SDK with a supported industrial camera, please ensure that the Industrial Camera SDK has been installed. See the table that follows:

Camera Manufacturer	**Supported SDK**
Basler AG	Pylon Camera Software Suite v. 5.0.10.10613
FLIR Systems Inc	Spinnaker SDK, v. 2.0.0.147
IDS Imaging Development Systems GmbH	IDS Software Suite 4.94
Lucid Vision Labs Inc	Arena SDK – Win 32/64-bit, v. 1.0.24.7

The SDK parses input video sources into a set of subsequent frames. It is up to the user to submit all or just a subset of them to the Plate Finding / Plate Recognizer operations.

Supported countries

All supported countries are part of the following geographical areas:

- Europe (including Russia)
- The Middle East
- Africa
- Asia and Australasia

- USA and South America

You don’t need to select the particular country, just denote the area (where the country of interest is located) in the appropriate API functions. For some countries the SDK is also able to report particular country region.

Dataset resolution

For each Geographical Area supported, the SDK provides different versions of the Plate Finder dataset (see `NNANPR_GetPlateFinderSpecificNames` and `NANPR_GetPlateRecognizerSpecificNames` functions).
For example, for Europe the SDK contains the following Plate Finder datasets:

- EU 1280
- EU 832
- EU 608
- EU 416

Before submitting the original image to the Plate Finder, the SDK will resize it. If you select EU 416, this means that the original image will be resized to 416 x 416 pixels before being submitted to the Plate Finder. Therefore regardless of the original size of the image, the SDK will always resize it to 416 x 416 pixels.
If the plate is quite small in the original image, after resizing it to 416 x 416 pixel, this may appear really tiny. See the images which follow:

The resolution of the image below is 1288 x 964 pixels

When the image is resized to 416 x 416 (see the picture that follows), the plate of the car located at a farther distance from the camera becomes very small and so the software might not detect it.

To overcome this problem there are two alternatives: either to use a bigger Plate Finder dataset (such as the 608 that requires more computational power) or to crop the original image using the Crop Region function before submitting it to the Plate Finder dataset. If the resolution of the original image is quite big (i.e. 2 or 3 Megapixel), you might consider using both solutions together.
On the other hand the bigger the license plate in the image (in pixels) the better the detection and reading results after the image shrinkage.
All the steps described in this section are automatically done by the SDK. You don't need to write a single line of code to crop or resize the input image.

Crop region

The crop region feature may be useful when you wish to remove a part of the image where it will be very unlikely to find a license plate (part of the sky, pavement, bush etc).
If enabled the Plate Finder will search for license plates only inside this region (see `NANPR_SetPlateFinderCroppingRegion` function). Currently the crop region is just one rectangular area.

Synchronous and Asynchronous

There are two major flavours of the Plate Finding and the Plate Recognizer operations: synchronous and asynchronous. Synchronous operations do not return any result until this is ready. On the other hand asynchronous operations return control to the caller immediately and as soon as the result is ready, this is returned through the provided callback function.
Also the asynchronous Plate Recognizer operation can be enhanced by adding a so-called recognition results accumulator (see `NNANPR_ReconizePlateAsyncWithAccumulator` function). The accumulator can accumulate separately a set of different results of Recognition Operation over a sequence of images, merging together similar results (and enhancing their confidence). It fires recognition callback only when some predefined condition is met like accumulation time span or number of accumulated results for one license plate.

Benchmarks

The figures contained by the tables which follow, report the time taken by the Plate Finding and Plate Recognizing operation (particularly `NNANPR_FindPlates` and `NNANPR_ReconizePlate` functions) to process one or more license plates in images with a size of 825x550 pixels. Plate Finder version V4 is more accurate although it requires more time compared to its predecessor. Figures for particular PC may vary a little bit depending on the current overall system load.

Computational device for Plate Finding	**time [ms]***					
	EU 416	EU v4 416	EU 608	EU v4 608	EU 832	EU v4 832
CPU Intel Atom E3950 1.6 GHz	237	304	424	611	759	1135
GPU Intel Atom E3950 1.6 GHz	98	120	223	252	435	475
CPU Intel Core i5-3570 3.4 GHz	54	72	97	132	168	233
CPU Intel Core i5-9500 3.0 GHz	19	23	35	44	56	78
GPU Intel Core i5-9500 3.0 GHz	90	120	158	263	240	439
CPU Intel Core i9-9900K 3.6 GHz	19	25	38	53	71	90
GPU Intel Core i9-9900K 3.6 GHz	17	44	27	88	43	160
GPU GeForce GTX 1050 CUDA	18	18	24	29	35	47
GPU GeForce GTX 1050 without CUDA	51	85	112	164	191	248
GPU GeForce RTX 2080 CUDA	7	11	12	17	20	28
GPU GeForce RTX 2080 without CUDA	44	61	80	115	144	208

Computational device for Plate Recognizing	time [ms]*
	EU 416
CPU Intel Atom E3950 1.6 GHz	235
GPU Intel Atom E3950 1.6 GHz	99
CPU Intel Core i5-3570 3.4 GHz	70
CPU Intel Core i5-9500 3.0 GHz	18
GPU Intel Core i5-9500 3.0 GHz	93
CPU Intel Core i9-9900K 3.6 GHz	19
GPU Intel Core i9-9900K 3.6 GHz	16
GPU GeForce GTX 1050 CUDA	18
GPU GeForce GTX 1050 without CUDA	47
GPU GeForce RTX 2080 CUDA	7
GPU GeForce RTX 2080 without CUDA	45

** averaged over 10 different images;*

Sample applications

Overview

The SDK contains several programming examples for C++, C# and Delphi programming languages in corresponding subfolders of the `Examples` folder. All GUI examples have the same look and functionality so different programming languages are not distinguished here.

GUI examples can accept both the separate images and the video streams. The console *PlateRecognize* example can work only with a single image file. On the other hand the *Multicameras* console example works only with video sources.

The `Settings` dialog of the GUI examples allows to select geographical area, particular plate finder and plate recognizer varieties for the area and the hardware device on which these operations will be executed. `Min plate height` option if not zero allows to filter plate bounding boxes after the Plate Finding stage and before submitting them to the Plate Recognition stage (in fact it's the height of the bounding box). Also the recognition accumulator can be adjusted here if selected.

As for the Plate Finder and the Plate Recognizer varieties of the same geographical area. It is important to highlight that they are not about the particular country selection. They are about performance and precision of the operations. The Plate Finder or the Plate Recognizer variety name consists of the geographical area shortcut and some number. This number designates the size in pixels to which the input source image will be resized internally before applying the operation. For example the variety name “EU 416” means that this operation is for European area and it resizes any input image to 416 x 416 pixels sized image before executing the operation itself. The smaller this size the faster the operation but at the same time it is less accurate. Also you have to take this information into account for example when looking for small license plates on a big image. After significant downsizing of the image the plate may become completely unreadable. On the other hand images of big plates (in pixels) may give better results after proper shrinkage.

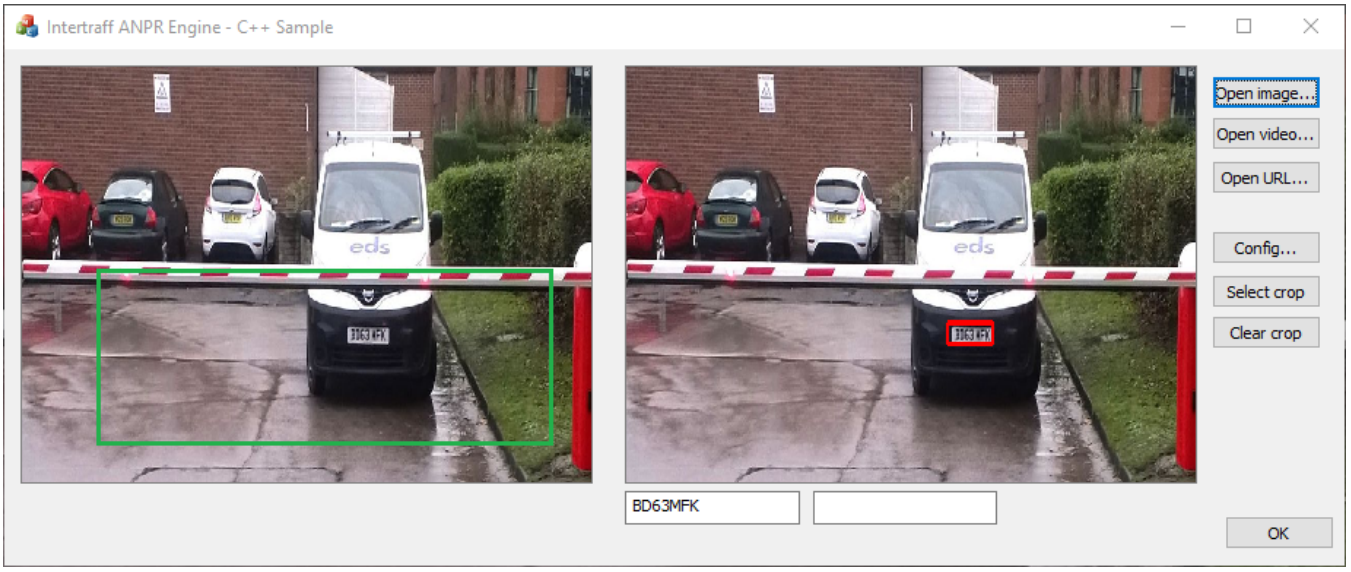
To evenly distribute the load and increase performance it is always preferable to select different devices for the Plate Finding and Plate Recognition operations.

There is one important thing concerning particularly the C# example: under the Visual Studio debugger the program may issue an exception like “Attempting managed execution inside OS Loader lock”. This issue is due to the program protection / licensing software. It doesn’t lead to any other problems during the program execution and can be safely ignored (just uncheck the `LoaderLock` exception in the `Debug -> Exceptions` dialog of the Visual Studio).

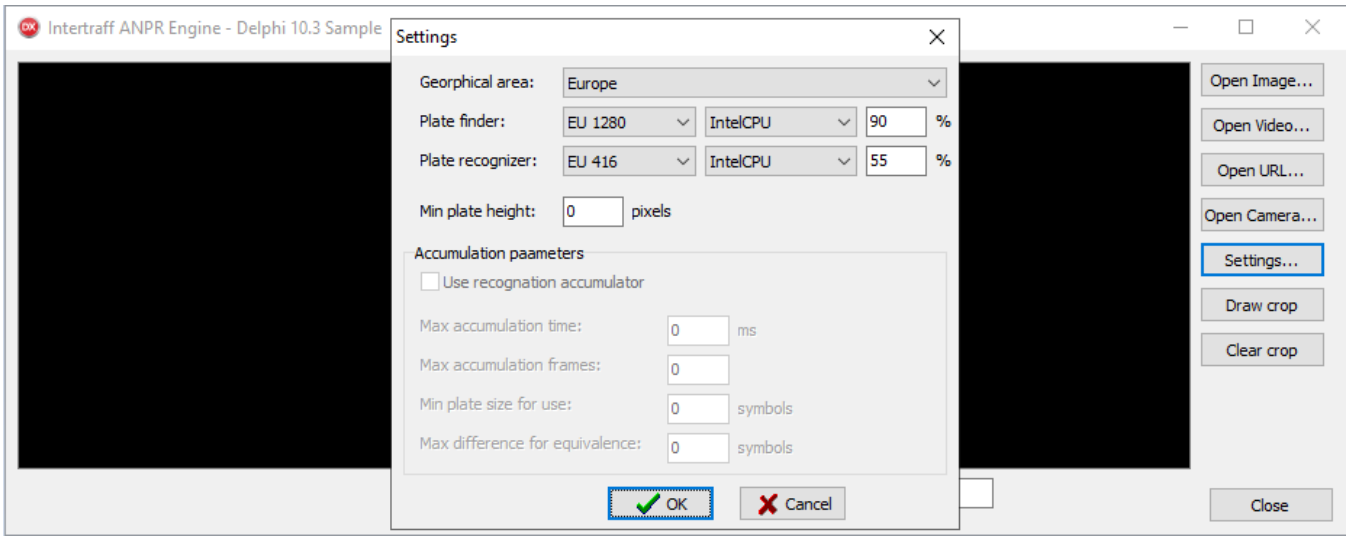
VideoSample example

This example makes use of synchronous versions of Plate Finding and Plate Recognition operations. So usage of the Recognition Accumulator is disabled here (in the `Config` dialog).

C++ example with single image recognition result.



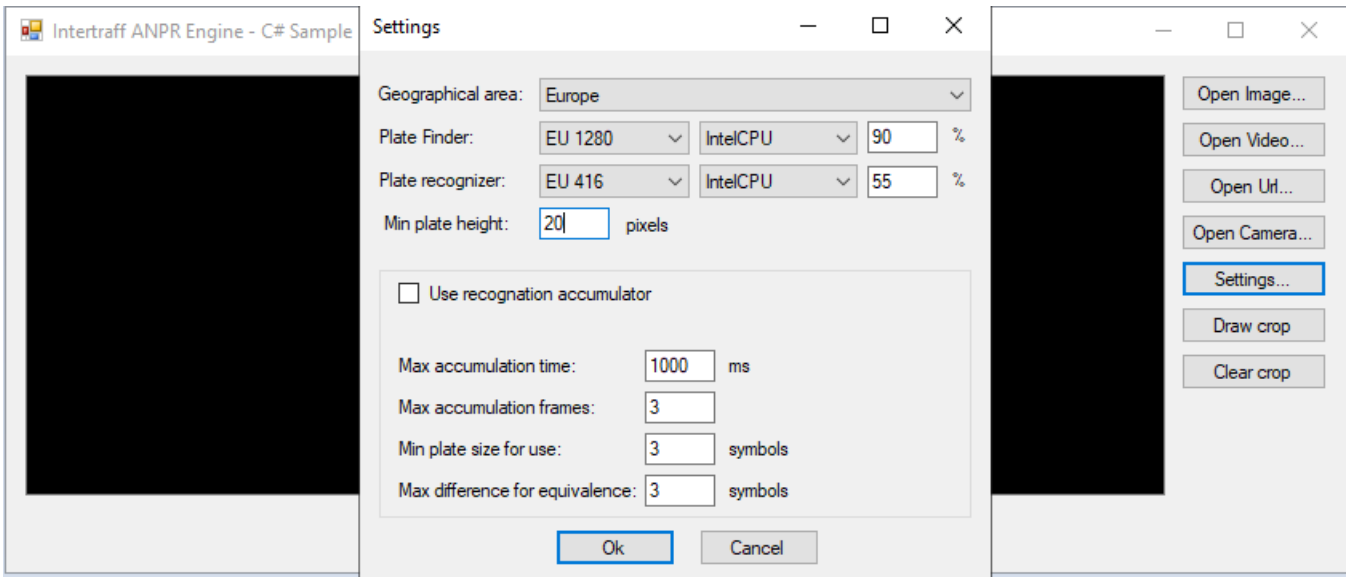
Here is the plate recognition inside the defined crop region. Defining the crop region may be very useful in some use case scenarios like barriers or when you need to control only one traffic lane on a multilane road. Every plate outside the crop region is neglected at the Plate Finding stage.



.. and here is the Delphi version screenshot with `Settings` dialog opened (`Use recognition accumulation` option is disabled here).

VideoAsinc example

This example is similar to the previous one but it makes use of asynchronous versions of Plate Finding and Plate Recognition operations and the Recognition Accumulator can be engaged here (see the `Settings` dialog).



here is the C# version with `Settings` dialog opened. Min plate height option is not zero here and all found license plates will be filtered before the plate recognition stage.

PlateRecognize example

This is the console based C++ example (but it can open the separate GUI window to illustrate the plate recognition result). It uses Europe plate finder and plate recognizer executed on Intel CPU.

Usage:

```
PlateRecognize.exe image_file_path
```

Multicameras example

This is also C++ console based example which illustrates how the SDK can work with several different live video sources (cameras).

Usage:

```
Multicameras.exe url1 [url2] [url3] [-PF {0,1,2,3,4}] [-PR {0,1,2,3,4}]
```

where:

`-PF` is a plate Finder device selector and `-PR` is a plate recognition device selector;
0 - Intel CPU(default), 1 - Intel GPU, 2 - Intel Myriad, 3-Server, 4 - NvidiaGPU.

Getting started

How to install the SDK

Just run the installer. Here you can select the installation folder and shortcuts creation.

Adding the SDK to your project

To add the SDK into your project just reference files from the `Include` subfolder of the installation folder. These are `nnAnpr.h` and `nnAnpr.lib` for C++ projects, `nnAnpr.pas` for Delphi projects and `nnAnpr.cs` for C# projects.

As a starting point you can review the source code of the example applications in the `Examples` subfolder of the SDK installation folder.

Reference

The following references comply with C++ syntax rules. But the SDK also supports a few other programming languages like C# and Delphi. All entity names and their semantics are the same in all languages. But some details like function signatures may be different. Please refer to the provided header files for particular details.

Enumerators

- NNANPR_Device

```
{ nndIntelCPU = 0, nndIntelGPU, nndMyriad, nndServer, nndNvidiaGPU }
```

- NNANPR_PlateType

```
{ nnptNotDefined = 0 , nnptTaxi, nnptDiplomatic }
```

- NNANPR_ImageFormat

```
{ nnifU8C3 }
```

- NNANPR_GeographicalArea

```
{ nngaEurope, nngaMiddleEast, nngaAfrica, nngaAsia_Australasia,
nngaUSA_SouthAmerica }
```

Structures

- NNANPR_Object

```
{
```

`int classId` – class identifier of the found object (license plate);
`RECT Position` – position of the found object;
`double Confidence` – confidence level of this object finding result;

```
}
```

- NNANPR_RecognizerResult

```
{
```

`wchar_t Characters[NNANPR_NumberPlateMaxSize]` – recognized plate number in UTF-16 encoding. In some countries the plate string consists of two parts with different alphabets (e.g. some plate types in Kingdom of Saudi Arabia have arabic part and duplicating it latin part). In this case the second part follows right after the first part in square brackets. For example: ٦٠٤٧اىح**[6047VAJ].** Characters outside the actual plate string are zeroed.

`float CharacterConfinedces[NNANPR_NumberPlateMaxSize]` – confidences for each found symbol;

`wchar_t Country[NNANPR_CountryMaxSize]` – country information of the found plate; empty if it's not detected;

`wchar_t CountryRegion[NNANPR_CountryRegionMaxSize]`– country region information of the found plate; empty if it's not detected;

`NNANPR_PlateType PlateType` – found plate type; `nnptNotDefined` if not detected;
`COLORREF Color` – colour of the found plate; 0xffffffff if not detected;
`RECT platePosition` – coordinates of the found plate; coordinate origin is at upper left corner of the image;

`NNANPR_RESULT ErrorCode` - error code of the operation;

```
}
```

Functions

- NANPR_CreatePlateFinder

```
NNANPR_RESULT NANPR_CreatePlateFinder(
```

```
      NNANPR_GeographicalArea ga,
      NNANPR_Device device,
      NNANPR_PlateFinder_ptr *pPlateFinder_ptr,
      const wchar_t *pSpecificName = nullptr)
```

The function creates a Plate Finder object and binds it to the particular device for execution.

parameters:

`device` – device used for processing by created Plate Finder object;
`pPlateFinder_ptr` – created Plate Finder object;
`pSpecificName` – specific name, null for default plate finder;

Device cannot be `nndServer` (i.e. plate finding operation can be done only locally).

- NNANPR_SetupPlateFinder

```
NNANPR_RESULT NNANPR_SetupPlateFinder(
      NNANPR_PlateFinder_ptr pPlateFinder_ptr,
      double Threshold)
```

The function sets the minimum acceptable confidence level for plate finding operation.

parameters:

`pPlateFinder_ptr` – Plate Finder object;
`Threshold` – minimum acceptable confidence level for plate finding operation in the range 0...1;

- NANPR_SetPlateFinderCroppingRegion

```
NNANPR_RESULT NANPR_SetPlateFinderCroppingRegion(
      NNANPR_PlateFinder_ptr pPlateFinder_ptr,
      RECT PixelCoords,
      RECT PercentCoords)
```

It sets the cropping region for the Plate Finder so that the Plate Finder will search for license plates only inside this region.

parameters:

`pPlateFinder_ptr` – Plate Finder object;
`PixelCoords` – cropping region in absolute pixel coordinates;
`PercentCoords` – cropping region in relative to the image sizes coordinates;

Coordinates origin is at the top left corner of the image.
Region value of `{0,0,0,0}` disables the crop region;
`PixelCoords` have higher priority.

- NNANPR_GetPlateFinderSpecificNames

```
NNANPR_RESULT NNANPR_GetPlateFinderSpecificNames(
      NNANPR_GeographicalArea ga,
      wchar_t *pFinderSpecificNamesBuffer,
      size_t BuffSize)
```

It gets the list of specific Plate Finder names for use in the `NNANPR_CreatePlateFinder` function. The name consists of the geographical area shortcut and the number. The bigger this number the more precise the plate finder and the more time / resources it takes to execute the plate finding operation.

- NNANPR_FreePlateFinder

```
NNANPR_RESULT NNANPR_FreePlateFinder(
      NNANPR_PlateFinder_ptr pPlateFinder_ptr)
```

It releases the Plate Finder object and frees occupied memory;

- NNANPR_CreatePlateRecognizer

```
NNANPR_RESULT NNANPR_CreatePlateRecognizer(
      NNANPR_GeographicalArea ga,
      NNANPR_Device device,
      NNANPR_PlateRecognizer_ptr *pPlateRecognizer_ptr,
      const wchar_t *pSpecificName = nullptr)
```

The function creates a Plate Recognizer object and binds it to the particular device for execution.

parameters:

`device` – device used for processing by created Plate Recognizer object;
`pPlateFinder_ptr` – created Plate Recognizer object;
`pSpecificName` – specific name, null for default plate finder;

- NNANPR_SetupPlateRecognizer

```
NNANPR_RESULT NNANPR_SetupPlateRecognizer(
      NNANPR_PlateRecognizer_ptr pPlateRecognizer_ptr,
      double Threshold)
```

The function sets the minimum acceptable confidence level for plate recognition operation.

parameters:

`pPlateRecognizer_ptr` - Plate Recognizer object;
`Threshold` - minimum acceptable confidence level for plate recognizing operation in the range 0...1;

- NNANPR_GetPlateRecognizerSpecificNames

```
NNANPR_RESULT NNANPR_GetPlateRecognizerSpecificNames(
      NNANPR_GeographicalArea ga,
      wchar_t *pPlateRecognizerNamesBuffer,
      size_t BuffSize)
```

It gets the list of specific Plate Recognizer objects for use in the `NNANPR_CreatePlateRecognizer` function. The name consists of the geographical area shortcut and the number. The bigger this number the more precise the plate recognizer and the more time / resources it takes to execute the plate recognizing operation.

- NNANPR_FreePlateRecognizer

```
NNANPR_RESULT NNANPR_FreePlateRecognizer(
      NNANPR_PlateRecognizer_ptr pPlateRecognizer_ptr)
```

It releases the Plate Recognizer object and frees occupied memory.

- NNANPR_CreateImage

```
NNANPR_RESULT NNANPR_CreateImage(
      NNANPR_Image_ptr *pImage_ptr)
```

It creates image object.

- NNANPR_FreeImage

```
NNANPR_RESULT NNANPR_FreeImage(
      NNANPR_Image_ptr pImage_ptr)
```

The function releases the image object and frees occupied memory.

- NNANPR_CloneImage

```
NNANPR_RESULT NNANPR_CloneImage(
      NNANPR_Image_ptr pSourceImage_ptr,
      NNANPR_Image_ptr pResultImage_ptr)
```

It makes deep copy of the image object. The destination image object must be created by the NNANPR_CreateImage function first.

- NNANPR_LoadImage

```
NNANPR_RESULT NNANPR_LoadImage(
      const wchar_t *pImagePath,
      NNANPR_Image_ptr pImage_ptr)
```

It loads an image from a file to the already created image object. The function supports a set of common image formats like JPEG, BMP, PNG, etc. The image format is determined by the file name extension.

- NNANPR_SaveImage

```
NNANPR_RESULT NNANPR_SaveImage(
      const wchar_t *pImagePath,
      NNANPR_Image_ptr pImage_ptr)
```

It saves the image referenced by the image object to a file. The function supports a set of common image formats like JPEG, BMP, PNG, etc. The image format is determined by the file content, not by the file name extension.

- NNANPR_DrawRect

```
NNANPR_RESULT NNANPR_DrawRect(
      NNANPR_Image_ptr pImage_ptr,
      RECT r,
```

```
      COLORREF rgb,
      int thickness)
```

The function draws a rectangle on the image referenced by the image object.

- NNANPR_GetImageData

```
NNANPR_RESULT NNANPR_GetImageData(
      NNANPR_Image_ptr pImage_ptr,
      NNANPR_ImageFormat imageFormat,
      DWORD *width,
      DWORD *height,
      void *OutputArray,
      size_t OutputArraySize)
```

The function gets image related data from the provided image object.

parameters:

`imageFormat` – expected image format;
`width` – returned image width;
`height` – returned image height;
`OutputArray` – provided buffer for image pixel data array. If it's null the function returns only the image sizes;
`OutputArraySize` – the size of the provided image pixel buffer;

- NNANPR_CreateImageWindow

```
NNANPR_RESULT NNANPR_CreateImageWindow(
      HWND parentHwnd,
      NNANPR_ImageWindow_ptr *pImageWindow_ptr)
```

It creates a window object to draw image objects on it.

parameters:

`parentHwnd` – handle of the parent window. If it's null - the function creates dialog window;
`pImageWindow_ptr` – output window object;

- NNANPR_FreeImageWindow

```
NNANPR_RESULT NNANPR_FreeImageWindow(
      NNANPR_ImageWindow_ptr pImageWindow_ptr)
```

It closes the window associated with the image window object and frees resources.

- NNANPR_DrawImage

```
NNANPR_RESULT NNANPR_DrawImage(
      NNANPR_ImageWindow_ptr pImageWindow_ptr,
      NNANPR_Image_ptr pImage_ptr)
```

The function renders the image in the window.

parameters:

`pImageWindow_ptr` - image window object holding the graphical window;
`pImage_ptr` - image object with image to be rendered;

- NNANPR_ImageWindowStartDrawCrop

```
NNANPR_RESULT NNANPR_ImageWindowStartDrawCrop(
      NNANPR_ImageWindow_ptr pImageWindow_ptr,
      NNANPROnCropSelectedCallback *pOnCropSelectCallback,
      void *pAddInfo)
```

The function enables drawing a rectangle representing crop region on the image window by the mouse. Pressing the left mouse button starts drawing the rectangle. When the left mouse button is released the drawing is finished and the `pOnCropSelectCallback` is called with the `pAddInfo` additional parameter. The rectangle will be rendered in green colour with line width equal to 3.

parameters:

`pImageWindow_ptr` - image window object for drawing;
`pOnCropSelectCallback` - callback function which will be called when the left mouse button is released;
`pAddInfo` - parameter passed to the callback function;

- NNANPR_ImageWindowStopDrawCrop

```
NNANPR_RESULT NNANPR_ImageWindowStopDrawCrop(
      NNANPR_ImageWindow_ptr pImageWindow_ptr)
```

The function disables drawing on the specified image window by the mouse.

- NNANPR_ImageWindowShowCrop

```
NNANPR_RESULT NNANPR_ImageWindowShowCrop(
      NNANPR_ImageWindow_ptr pImageWindow_ptr,
      bool showCrop)
```

The function shows / hides the current crop region on the specified image window depending on the `showCrop` parameter.

- NNANPR_WaitImageWindow

```
NNANPR_RESULT NNANPR_WaitImageWindow(
      NNANPR_ImageWindow_ptr pImageWindow_ptr,
      DWORD AutoCloseWaitTimeMS,
      bool closeByPressAnyKey)
```

The function closes the image window with no parents. The window will be closed either after pressing any key (if `closeByPressAnyKey` is true) or after specified timeout. The function blocks till the window is closed.

parameters:

`pImageWindow_ptr` – the window which should be closed;
`AutoCloseWaitTimeMS` – closing timeout in milliseconds;
`closeByPressAnyKey` – if true the window will be closed on the keypress;

return values:

`1` – the windows was closed by keypress;
`AutoCloseWaitTimeMS` – the window was closed after specified time interval;
`0` – the window has parent window and nothing happened;

- NNANPR_CreateVideoCapture

```
NNANPR_RESULT NNANPR_CreateVideoCapture(
      NNANPR_ImageCapture_ptr *pImageCapture_ptr,
      const wchar_t *pVideoFileorURL)
```

The function creates the video capture object to parse a video stream either from the video file or from the online source referenced by URL.

parameters:

`pImageCapture_ptr` - created video capture object;
`pVideoFileorURL` - UTF-16 encoded null terminated C-string containing either video file path on local PC or URL of the online video stream;

- NNANPR_GetSpecialVideoCaptureList

```
NNANPR_RESULT NNANPR_GetSpecialVideoCaptureList(
      wchar_t *pSpecialVideoCaptureNamesBuffer,
      size_t BuffSize)
```

The function returns a list of video sources names which require special treatment. Currently It can be Flir and Point Grey industrial cameras (only USB 3.0 and GigE). In the near future, the SDK will also support cameras manufactured by Basler, IDS and Lucid Vision Labs.

parameters:

`pSpecialVideoCaptureNamesBuffer` - memory buffer supplied by the caller for the video sources names list returned by the function;
`BuffSize` - the buffer size in characters;

- NNANPR_CreateSpecialVideoCapture

```
NNANPR_RESULT NNANPR_CreateSpecialVideoCapture(
      NNANPR_ImageCapture_ptr *pImageCapture_ptr,
      const wchar_t *pSpecialVideoCaptureName)
```

The function creates the video capture object based on the video source name returned by the `NNANPR_GetSpecialVideoCaptureList` function.

parameters:

`pImageCapture_ptr` – on the function return it holds the created video capture object;
`pSpecialVideoCaptureName` – the name of the special video source device returned by the `NNANPR_GetSpecialVideoCaptureList` function.

- NNANPR_FreeVideoCapture

```
NNANPR_RESULT NNANPR_FreeVideoCapture(
      NNANPR_ImageCapture_ptr pImageCapture_ptr)
```

It closes the video capture object and frees all associated resources.

- NNANPR_CaptureFrame

```
NNANPR_RESULT NNANPR_CaptureFrame(
      NNANPR_ImageCapture_ptr pImageCapture_ptr,
      NNANPR_Image_ptr pImage_ptr)
```

The function gets the next frame as an image object from the video stream associated with the capture object. The image object must be created first by the `NNANPR_CreateImage` function.

- NNANPR_StartCaptureFrameAsync

```
NNANPR_RESULT NNANPR_StartCaptureFrameAsync(
      NNANPR_ImageCapture_ptr pImageCapture_ptr,
      NNANPROnFrameCallback *fOnFrameCallback,
      void *pAddInfo)
```

The function starts asynchronous frames capture from the video capture object with rate of the stream. It doesn't block the calling thread.

parameters:

`pImageCapture_ptr` – video capture object;
`fOnFrameCallback` – callback function which will be called on each frame arrived from the video stream;
`pAddInfo` – additional parameter which will be sent to the callback function;

- NNANPR_StopCaptureFrameAsync

```
NNANPR_RESULT NNANPR_StopCaptureFrameAsync(
      NNANPR_ImageCapture_ptr pImageCapture_ptr)
```

It stops asynchronous frame capturing.

- NNANPR_GetVideoFPS

```
NNANPR_RESULT NNANPR_GetVideoFPS(
      NNANPR_ImageCapture_ptr pImageCapture_ptr,
      double *pFPS
```

The function tries to determine the video stream FPS if possible.

- NNANPR_FindPlates

```
NNANPR_RESULT NNANPR_FindPlates(
      NNANPR_PlateFinder_ptr pPlateFinder_ptr,
      NNANPR_Image_ptr pImage_ptr,
      NNANPR_Object *pPlateArray,
      size_t *plateArraySizeInOut)
```

The function finds car license plates on the provided image.

parameters:

`pPlateFinder_ptr` – Plate Finder object;
`pImage_ptr` – image object on which the license plates will be found;
`pPlateArray` – in / out buffer for license plate objects;
`plateArraySizeInOut` – on input it defines the size of the provided `pPlateArray` buffer; on output it holds the actual number of found plates;

return values:

`NNANPR_RESULT_FALSE` – if it can not find any plate;
`NNANPR_BUFFER_TOO_SMALL` – provided `pPlateArray` buffer is too small to hold all found plate objects;

- NNANPR_FindPlatesAsync

```
NNANPR_RESULT NNANPR_FindPlatesAsync(
      NNANPR_PlateFinder_ptr pPlateFinder_ptr,
      NNANPR_Image_ptr pImage_ptr,
      NNANPROnPlateFoundCallback *pOnPlateFoundCallback,
      void *pAddInfo)
```

The function finds plates on the provided image asynchronously in a separate thread so that it returns immediately before the Plate Finder object will do the job. The image queue depth is equal to 1, i.e. if the plate finding thread is busy and the image queue already contains the image then the next call to the function overwrites the image (and the corresponding `pAddInfo`) in the queue.

parameters:

`pPlateFinder_ptr` – plate finder object;
`pImage_ptr` – input image object;
`pOnPlateFoundCallback` – callback function which will be called at the end of the plate finding operation;
`pAddInfo` – additional parameter which will be sent to callback function;

- NNANPR_ReconizePlate

```
NNANPR_RESULT NNANPR_ReconizePlate(
      NNANPR_PlateRecognizer_ptr pPlateRecognizer_ptr,
      NNANPR_Image_ptr pImage_ptr,
      RECT Position,
      NNANPR_RecognizerResult *resultRR)
```

The function recognizes (finds license text string, plate type, country of origin and so on if possible) the license plate on the image at the defined position.

parameters:

`pPlateRecognizer_ptr` – plate recognizer object;
`pImage_ptr` – input image object;
`Position` – position in pixels of license plate on the image (it can be the result of plate finding operation); coordinate origin is at the top left image corner;
`resultRR` – plate recognition result object;

return values:

`NNANPR_RESULT_SUCCESS` – if the function succeeds;
`NNANPR_RESULT_FALSE` – if the plate was not recognized;

- NNANPR_ReconizePlateAsync

```
NNANPR_RESULT NNANPR_ReconizePlateAsync(
      NNANPR_PlateRecognizer_ptr pPlateRecognizer_ptr,
      NNANPR_Image_ptr pImage_ptr,
      RECT Position,
      NNANPROnRecognitionCallback *pCallback,
      void *pAddInfo)
```

The function recognizes car license plates on the provided image asynchronously in a separate thread so that it returns immediately before the Plate Recognizer object will do the job. The image queue depth is equal to 1, i.e. if the plate recognizing thread is busy and the image queue already contains the image then the next call to the function overwrites the image (and the corresponding `pAddInfo`) in the queue.

parameters:

`pPlateRecognizer_ptr` – plate recognizer object;
`pImage_ptr` – input image object;
`Position` – position of the plate on the input image in pixels (possibly it's the result of the plate finding operation); coordinate origin is at the top - left corner of the image;
`pCallback` – callback function called at the end of the successive plate recognition operation;
`pAddInfo` – additional parameter for the callback function; it is associated with the image object;

- NNANPR_CreateRecognitionAccumulator

```
NNANPR_RESULT NNANPR_CreateRecognitionAccumulator(
      NNANPR_RecognitionAccumulator_ptr *pRecognitionAccumulator_ptr)
```

The function creates the recognition accumulation object. The recognition accumulator is a technique of improving the license plate recognition confidence by merging (accumulating) several recognition results of the same plate from different frames.

- NNANPR_SetupRecognitionAccumulator

```
NNANPR_RESULT NNANPR_SetupRecognitionAccumulator(
      NNANPR_RecognitionAccumulator_ptr pRecognitionAccumulator_ptr,
```

```
    size_t MaxAccumulationTimeMs,
    size_t MaxAccumulationFrames,
    size_t MinPlateSizeForUse,
    size_t MaxDifferenceForEquivalence)
```

The function configures the recognition accumulator object.

parameters:

`pRecognitionAccumulator_ptr` – accumulation object;
`MaxAccumulationTimeMs` – maximum time span of accumulating recognition results for one plate number in video stream;
`MaxAccumulationFrames` – maximum recognition results accumulated for one plate number;
`MinPlateSizeForUse` – minimum accepted license plate text size in characters;
`MaxDifferenceForEquivalence` – maximum difference in plate text characters of merged recognition results;

- NNANPR_FreeRecognitionAccumulator

```
NNANPR_RESULT NNANPR_FreeRecognitionAccumulator(
    NNANPR_RecognitionAccumulator_ptr pRecognitionAccumulator_ptr)
```

It releases the recognition accumulation object and frees associated resources.

- NNANPR_ReconizePlateAsyncWithAccumulator

```
NNANPR_RESULT NNANPR_ReconizePlateAsyncWithAccumulator(
    NNANPR_PlateRecognizer_ptr pPlateRecognizer_ptr,
    NNANPR_RecognitionAccumulator_ptr pRecognitionAccumulator_ptr,
    NNANPR_Image_ptr pImage_ptr,
    RECT Position,
    NNANPROnRecognitionCallback *pCallback,
    void *pAddInfo)
```

This function is the same as `NNANPR_ReconizePlateAsync` but it makes use of the recognition results accumulator (see **Architecture overview** section).

parameters:

`pPlateRecognizer_ptr` – plate recognizer object;
`pRecognitionAccumulator_ptr` – recognition results accumulator object;
`pImage_ptr` – input image object;

`Position` – position of the plate on the input image in pixels (possibly it's the result of the plate finding operation); coordinate origin is at the top - left corner of the image;
`pCallback` – callback function called when the recognition accumulator issues the accumulated recognition result;
`pAddInfo` – additional parameter for the callback function; it is associated with the image object having the best recognition confidences among accumulated for this recognition result;

- NNANPR_GetErrorString

```
NNANPR_RESULT NNANPR_GetErrorString(
      NNANPR_RESULT errCode,
      wchar_t *errorBuffer,
      size_t buffSize)
```

The function returns a human readable text string corresponding to the particular error code returned by the SDK API functions.