Red and OpenCV

How to install OpenCV binding for Red

You need first **the last Red master branch of Red** (red-28-dec16 or newer) due to evolution from the 0.61 version.
Basically, **you don't need to install OpenCV**. You'll find in /DLLs directory a 32-bit compiled version of the OpenCV framework (3.0 and 3.10) for the three main operating systems (Mac OS, Linux and Windows). Just copy the library (.dylib, .so or .dll) somewhere on your computer and then edit the *platforms.reds* file (in /libs) and make the links according to your paths as in the following example :

```
#switch OS [
   MacOSX          [#define cvWorld "/usr/local/lib32/opencv3/libopencv_world.dylib"]
   Windows         [#define cvWorld "c:\opencv310\build\x86\mingw\libopencv_world310.dll"]
   Linux           [#define cvWorld "/usr/local/lib/libopencv_world.so.3.1.0"]
]
```

Currently, **absolute path** to the library must be provided in OS-specific format, but relative path will be added in new versions of Red (probably in 0.62 version).

Note for Windows users: when making an installation from scratch, you need to install complementary Microsoft Visual C libraries, if there are not present on your system. These files (msvcp120.dll and msvcr120.dll) are also included in DDLs/Windows/MSVisualC directory. Just copy both files to your computer and add path to your path file.

In /lib/include.reds file you'll find a link to image and video samples that are used in code. Please adapt according to your system.

```
; to get an image and a movie. Adapt according to your paths
#switch OS [
        MacOSX   [picture: "/Users/fjouen/Pictures/lena.tiff" movie: "/Users/fjouen/Movies/skate.mp4"]
        Windows  [picture: "c:\Users\palm\Pictures\lena.tif"  movie: "c:\Users\palm\Videos\skate.mp4"]
        Linux    [picture: "/home/fjouen/Images/lena.tiff"  movie: "/home/fjouen/Vidéos/skate.mp4"]
]
```

Image and video files are included in /_images directory.

The /libs/ include.reds file contains all links to OpenCV library that are necessary and thus **MUST BE INCLUDED** in any Red program that wants use OpenCV functions (see samples)

About structures and routines

This binding uses a lot of OpenCV structures such as IplImage, CvMat and many others. Fortunately, Red is able to return structures as pointers, which is useful since basically OpenCV functions use pointers to structures. In this case structures are passed *as reference* to OpenCV functions.
However, many OpenCV functions use a specific type defined in C language as follows: *typedef void CvArr*. This is a metatype used only as a function parameter. It denotes that the function accepts arrays of multiple types, such as IplImage*, CvMat* or even CvSeq*. The particular array type is determined at runtime by analyzing the first 4 bytes of the header.

To be coherent with OpenCV in Red we use *an int-ptr! for CvArr!* which is defined as #define CvArr! int-ptr!. This means that you have to CAST returned structures when OpenCV functions require a *CvArr* parameter:

```
example
picture: "/Users/UserName/Pictures/lena.tiff"
img: cvLoadImage picture CV_LOAD_IMAGE_ANYCOLOR
;img is a IplImage! (a pointed structure) and we can use structure members

src: as int-ptr! img ; a pointer to img for functions requiring CvArr*
s0: as int-ptr! cvCreateImage img/width img/height IPL_DEPTH_8U 1
s1: as int-ptr! cvCreateImage img/width img/height IPL_DEPTH_8U 1
s2: as int-ptr! cvCreateImage img/width img/height IPL_DEPTH_8U 1

cvCreateImage: returns an IplImage!
These 3 images are thus casted to int-ptr! since the cvSplit function we want use to split 3 image channels requires CvArr parameter
cvSplit src s0 s1 s2 null
```

For this binding, most of 600 OpenCV functions are written with Red/System DSL and can be used with the *#import* directive. Imported OpenCV functions can be directly call by Red/System programs or can be accessed with *routines* if you use Red language. Routines are a fantastic tool that allows accessing C functions included in DLL. This means that you can use either Red/System DSL or Red language to write your image processing programs. Result will be the same. This a basic example to show how create an OpenCV window with both language versions.

Red/System

```
Red/System [
	Title:		"OpenCV Tests: Creating  Window"
	Author:		"F. Jouen"
	Rights:		"Copyright (c) 2012-2016 F. Jouen. All rights reserved."
	License:	"BSD-3 - https://github.com/dockimbel/Red/blob/master/BSD-3-License.txt"
]

; calls OpenCV binding
#include %../../../libs/include.reds ; all OpenCV  functions
windowsName: "OpenCV Window [ESC to close Window]"
cvNamedWindow windowsName CV_WND_PROP_AUTOSIZE
cvResizeWindow windowsName 640 480
cvMoveWindow windowsName 200 200
cvWaitKey 0
cvDestroyAllWindows
```

Now in Red Language

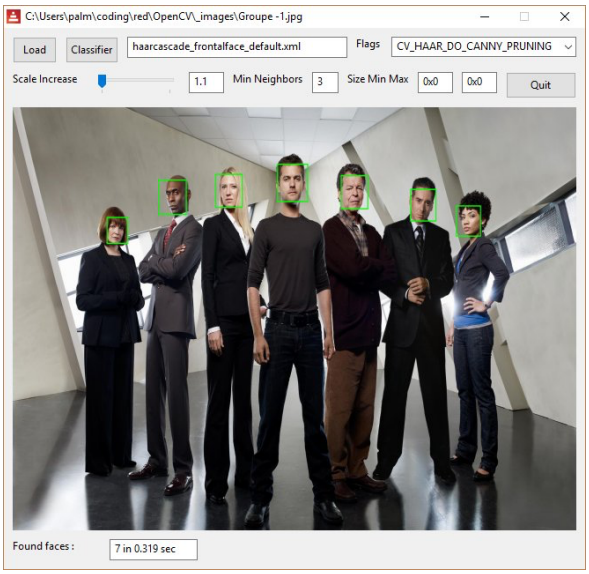
```
Red [
	Title:		"Creating  Window"
	Author:		"F. Jouen"
	Rights:		"Copyright (c) 2012-2016 F. Jouen. All rights reserved."
	License:	"BSD-3 - https://github.com/dockimbel/Red/blob/master/BSD-3-License.txt"
]

; import required OpenCV libraries
#system [
	; import required OpenCV libraries
	#include %../../libs/include.reds ; all OpenCV  functions
	; global variables
	windowsName: "OpenCV Window [Any Key to close Window]"
]

; red routines are interface between red/system and red code. Great!
makeWindow: routine [] [
	cvNamedWindow windowsName CV_WND_PROP_AUTOSIZE
	cvResizeWindow windowsName 640 480
	cvMoveWindow windowsName 200 200
	cvWaitKey 0
	cvDestroyAllWindows
]
; just a simple red code :)
makeWindow
```

There are some differences when writing Red Language code. First it's necessary to use the *#system* directive to include OpenCV libraries. This is also the place to declare any global variables that will be used by the program. Second, you have to write *routines* that behave as an interface between you red code and the Red/System functions. OpenCV functions and global variables are thus directly called inside routines.
In both cases, the output result will be similar as illustrated in next figure.

The interest of using Red Language is that you can employ Red/View and Red/Draw DSL for creating GUI and developing sophisticated interface for computer vision such as in the next sample of face processing with Red.



You'll find in sample directory, many illustrations of OpenCV usage with Red. This binding is based on OpenCV 3.0 version but is compatible with earlier versions of OpenCV since we are just using API C functions and not C++ implementation. Documentation of OpenCV functions is not included but can be found here: http://opencv.org/documentation.html.

Some Red/System functions parameters may differ with OpenCV due to some difference when passing pointer *by value* to a function.

size: declare CvSize2D32f! (structure with 2 float32! values)
size/width: 640.00
size/height: 480.00
in C, we can write *image= cvCreateImage size IPL_DEPTH_32F 3*
But since actually in Red structures are passed by reference and not by value we have to write: *image: cvCreateImage size/width size/height IPL_DEPTH_32F 3*
This is the best way to reserve correct memory size for the structure.

In /manual/_pdf directory, you'll find a list of OpenCV functions that have been adapted for Red.

core.pdf: the most important (more than 350 functions) file with basic and sophisticated functions for image processing.
core_tools.pdf: a series of Red functions to solve some problems when structures are returned as structure and not as pointer.
highgui.pdf: functions for creating OpenCV interface. Useful if you are not using Red/View for GUI.
imgcodecs.pdf: for loading and saving images.
imgproc.pdf: all about image processing with Red and OpenCV.
objdetect.pdf: how to use Haar classifier for finding objects in image.
photo.pdf: for image impainting.
video.pdf: for movement processing in video image.
videoio.pdf: to use camera or movies with Opencv.
capture.pdf: for Windows users. Experimental DLL to replace basic videoio when using newer versions of OpenCV (3.0 and >)

Detailed implementation of structures and functions can be found in /libs/XX directory.

Using OpenCV with Red/View

You'll find in /samples_gui directory some code which illustrates how to use OpenCV and Red/View DSL. Actually, Red/View is only supported by Windows but is programmed for Mac OS soon.
Of course, there are many differences between OpenCV and Red/View for the image type implementation. For example, Red images are in ARBG order and OpenCV images are in BGRA order. OpenCV can create 8, 16, 32 or 64-bit images or matrices since Red is limited to 8-bits images. Similarly, OpenCV images can contain 1, 2, 3 or 4 channels and Red uses only 4-channels images. This explains why I had to add in /libs/red/cvroutines.red a series of routines and functions which makes possible the communication between OpenCV and Red images, such as *makeRedImage* function that transforms an OpenCV image to a Red Image.

```
makeRedImage: function [im [integer!] w [integer!] h [integer!] return: [image!]] [
	lineRequired: getImageOffset im
	either lineRequired [makeImagebyLine im w h] [makeImage im w h]
]
```

Routines and functions included in this file require the **address of the image** to be processed considered as an integer! Basically, CvArr! Red/System type could be used for these routines. However, int-ptr! type doesn't exist in Red and thus you have to cast pointers as an integer! which is used as an address to the image structure. This is easily done as follows:

```
#include %../../libs/red/cvroutines.red ; some routines for image conversion from OpenCV to Red
fname: "lena.jpg"
src: as int-ptr! cvLoadImage fName CV_LOAD_IMAGE_COLOR ; a pointer to a color OpenCV image
img: as integer! src ; the address of the pointed image
wsz: getIWidth img ; routine to get image size in x
hsz: getIHeight img; routine to get image size in  y
canvas/image: makeRedImage img wsz hsz ; function that makes a Red image
```