Red and OpenCV

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 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! tmp ; 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 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 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
```

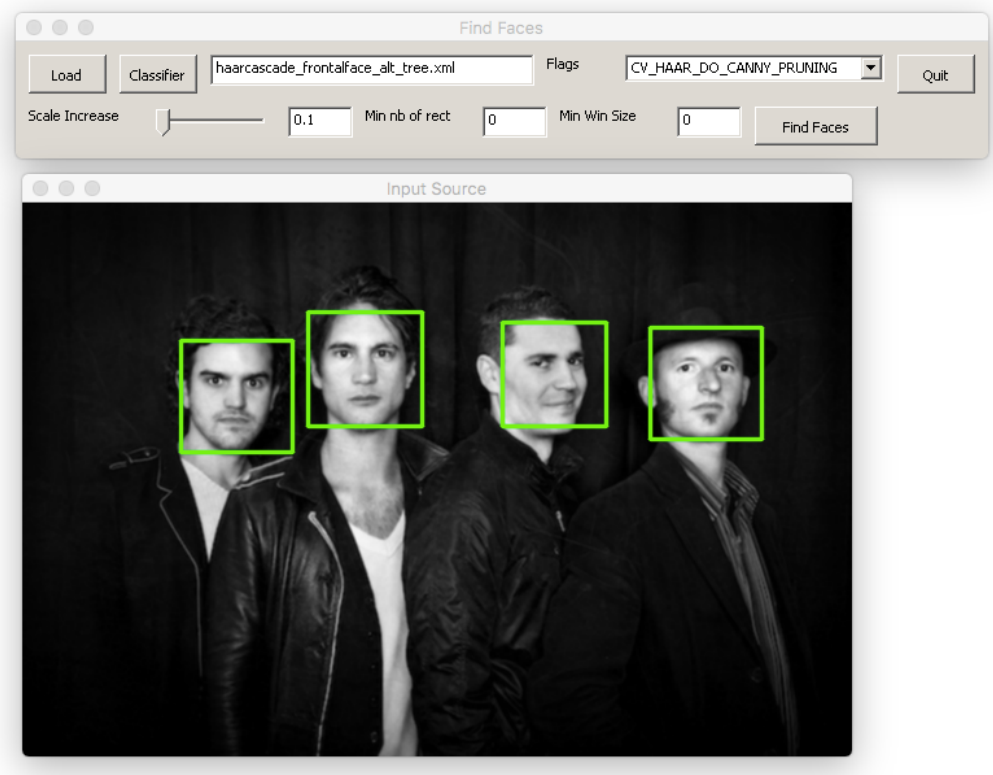
Now in Red Language

```
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 librairies. 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 funtions. OpenCV functions 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 View and Draw DSL for creating GUI and developing sophisticated interface for computer vision in Red 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 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 be 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.

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