11/07/2016

REDCV FUNCTIONS DOCUMENTATION

Most of functions are calling Red/System routines implemented in /libs/core/rcvRoutines.red for faster image rendering. All redCV routines can be directly called from a red program (not for newbies). For a more convenient access, Red/System routines are "exported" to red functions. Only red functions are documented. Doc string summarizes the function and calling function from red code sample is in *italic.*

All includes to redCV libraries are declared in a single file (/libs/redcv.red).

```
[
#include %core/rcvRoutines.red          ; All Red/System routines
#include %core/rcvImage.red             ; Image creating functions
#include %core/rcvCore.red              ; Basic image processing functions
#include %imgproc/rcvImgProc.red        ; Color space conversions, Convolution and other
#include %math/rcvRandom.red            ; Random laws for generating random images
#include %math/rcvStats.red             ; Statistical functions for images
]
```

Just deactivate the #include if you don't need functions, but rcvRoutines.red, rcvImage.red and rcvCore are obligatory.

More documentation to come.

Library: redCV/libs/core/rcvImage.red
Functions: 12

Function	Doc String
Basic Image I/O	
rcvCreateImage: function [size [pair!] return: [image!] *img: rcvCreateImage 640x480*	Create empty (black) image
rcvReleaseImage: routine [src [image!]] *rcvRelease img*	Delete image from memory
rcvLoadImage: function [fileName [file!] return: [image!] *img: rcvLoadImage %test.jpg*	Load image from file
rcvLoadImageB: function [fileName [file!] return: [binary!] /alpha *bin: rcvLoadImageB %test.png (bin=rgb)* *bin: rcvLoadImageB/apha %test.png (bin=argb)*	Load image from file and return image as binary
rcvSaveImage: function [src [image!] fileName [file!] *rcvSaveImage img %test.jpg*	Save image to file
rcvCloneImage: function [src [image!] return: [image!] *dst: rcvCloneImage src*	Return a copy of source image
rcvCopyImage : function [src [image!] dst [image!] *rcvCopyImage src dst*	Copy source image to destination image
rcvRandomImage: function [size [pair!] value [tuple!] /uniform /alea return: [image!]] *dst: rcvRandomImage/uniform 640x480 red*	Create a random uniform or pixel random image
rcvZeroImage: function [src [image!] *rcvZeroImage src*	All pixels to 0
rcvDecodeImage	TBD
rcvDecodeImageM	TBD
cvEncodeImage	TBD

Library: redCV/libs/core/rcvcore.red
Functions: 42

Function	Doc String
Image Conversion	
rcv2Gray: function [src [image!] dst [image!] /average /luminosity /lightness return: [image!] *rcv2Gray/average src dst*	Convert RGB image to Grayscale according to refinement
rcv2BGRA: function [src [image!] dst [image!] *rcv2BGRA src dst*	Convert RGBA => BGRA
rcv2RGBA: function [src [image!] dst [image!] *rcv2RGBA src dst*	Convert BGRA => RGBA"
rcv2BW: function [src [image!] dst [image!] *rcv2BW src dst*	Convert RGB image => Black and White
rcv2BWFilter: function [src [image!] dst [image!] thresh [integer!] *rcv2BWFilter src dst 64*	Convert RGB image => Black and White according to threshold
rcvSplit: function [src [image!] dst [image!]/red /green /blue *rcvSplit/blue src dst (->blue channel)*	Split source image in RGB separate channels
rcvInvert: function [source [image!] dst [image!] *rcvInvert src dst*	Similar to NOT image
Math Operators on image	
rcvAdd: function [src1 [image!] src2 [image!] dst [image!] *rcvAdd image1 image2 destImage*	dst: src1 + src2
rcvSub: function [src1 [image!] src2 [image!] dst [image!] *rcvSub image1 image2 destImage*	dst: src1 - src2
rcvMul: function [src1 [image!] src2 [image!] dst [image!] *rcvMul image1 image2 destImage*	dst: src1 * src2
rcvDiv: function [src1 [image!] src2 [image!] dst [image!] *rcvDiv image1 image2 destImage*	dst: src1 / src2
rcvMod: function [src1 [image!] src2 [image!] dst [image!] *rcvMod image1 image2 destImage*	dst: src1 // src2 (modulo)
rcvRem: function [src1 [image!] src2 [image!] dst [image!] *rcvRem image1 image2 destImage*	dst: src1 % src2 (remainder)
rcvAbsDiff: function [src1 [image!] src2 [image!] dst [image!] *rcvAbsDiff image1 image2 destImage*	dst: absolute difference src1 src2

Math operators with scalar (integer !)	
rcvAddS: function [src [image!] dst [image!] val [integer!] *rcvAddS source destination 128*	dst: src + integer! value
rcvSubS: function [src [image!] dst [image!] val [integer!] *rcvSubS source destination 128*	dst: src - integer! value
rcvMulS: function [src [image!] dst [image!] val [integer!] *rcvMubS source destination 2*	dst: src * integer! value
rcvDivS: function [src [image!] dst [image!] val [integer!] *rcvDivS source destination 2*	dst: src / integer! value
rcvModS: function [src [image!] dst [image!] val [integer!] *rcvModS source destination 4*	dst: src // integer! value (modulo)
rcvRemS: function [src [image!] dst [image!] val [integer!] *rcvRemS source destination 2*	dst: src % integer! value (remainder)
rcvPow: function [src [image!] dst [image!] val [integer!] *rcvPow source destination 2*	dst: src ^integer! value
rcvLSH: function [src [image!] dst [image!]val [integer!] *rcvLSH source destination 2*	Left shift image by value
rcvRSH: function [src [image!] dst [image!] val [integer!] *rcvRSH source destination 2*	Right Shift image by value
rcvSQR: function [src [image!] dst [image!] val [integer!] *rcvSQR source destination 2*	Image square root
Math operators with scalar (tuple!)	
rcvAddT: function [src [image!] dst [image!] val [tuple!] *rcvAddT source destination 128.128.128*	dst: src + tuple! value
rcvSubT: function [src [image!] dst [image!] val [tuple!] *rcvSubT source destination 32.32.32*	dst: src - tuple! value
rcvMulT: function [src [image!] dst [image!] val [tuple!] *rcvMulT source destination 2.2.2*	dst: src * tuple! value
rcvDivT: function [src [image!] dst [image!] val [tuple!] *rcvDivT source destination 2.2.2*	dst: src / tuple! value
rcvModT: function [src [image!] dst [image!] val [tuple!] *rcvModT source destination 2.2.2*	dst: src // tuple! value (modulo)
rcvRemT: function [src [image!] dst [image!] val [tuple!] *rcvRemT source destination 2.2.2*	dst: src % tuple! value (remainder)
Logical operators on Image	
rcvAND: function [src1 [image!] src2 [image!] dst [image!]	dst: src1 AND src2

rcvAND source1 source2 destination	
rcvOR: function [src1 [image!] src2 [image!] dst [image!] *rcvOR source1 source2 destination*	dst: src1 OR src2
rcvXOR: function [src1 [image!] src2 [image!] dst [image!] *rcvXOR source1 source2 destination*	dst: src1 XOR src2
rcvNAND: function [src1 [image!] src2 [image!] dst [image!] *rcvNAND source1 source2 destination*	dst: src1 NAND src2
rcvNOR: function [src1 [image!] src2 [image!] dst [image!] *rcvNOR source1 source2 destination*	dst: src1 NOR src2
rcvNXOR: function [src1 [image!] src2 [image!] dst [image!] *rcvNXOR source1 source2 destination*	dst: src1 NXOR rc2
rcvMIN: function [src1 [image!] src2 [image!] dst [image!] *rcvMIN source1 source2 destination*	dst: minimum src1 src2
rcvMAX: function [src1 [image!] src2 [image!] dst [image!] *rcvMAX source1 source2 destination*	dst: maximum src1 src2
rcvNot: function [src [image!] dst [image!] *rcvNOT source destination*	dst: NOT src
logical operators and scalar (tuple!) on image	
rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!] *rcvANDS source red*	dst: src AND tuple! as image
rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!] *rcvANDS source green*	dst: src OR tuple! as image
rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!] *rcvANDS source blue*	dst: src XOR tuple! as image

Library: redCV/libs/imgproc/cvImgProc.red
Functions: 3

Function	Doc String
Space Color Conversion	
rcvRGB2XYZ: function [src [image!] dst [image!] *rcvRGB2XYZ src dst*	BGR to CIE XYZ color conversion
rcvXYZ2RGB: function [src [image!] dst [image!] *rcvXYZ2RGB src dst*	CIE XYZ to RBG color conversion
Image transformation	
rcvFlip: function [src [image!] dst [image!] /horizontal /vertical /both return: [image!] *rcvFlip/horizontal src dst* *rcvFlip/vertical src dst* *rcvFlip/both src dst*	Left Right, Up down or both directions flip
Image Convolution	
rcvConvolve: function [src [image!] dst [image!] kernel [block!] factor [float!] delta [float!] *rcvConvolve src dst noFilter 1.0 0.0*	Convolve an image with the kernel
rcvFilter2D: function [src [image!] dst [image!] kernel [block!] delta [integer!] *rcvFilter2D src dst noFilter 0*	Basic convolution Filter
rcvFastFilter2D: function [src [image!] dst [image!] kernel [block!] *rcvFastFilter2D src dst Filter*	Faster convolution Filter
rcvMakeGaussian: function [kSize [pair!] return: [block!] *knl: rcvMakeGaussian 3x3*	Create a Gaussian uneven kernel with the following equation $$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$ Where, x is the distance along horizontal axis measured from the origin, y is the distance along vertical axis measured from the origin and σ is the standard deviation of the distribution.
rcvGaussianFilter: function [src [image!] dst [image!] kernel [block!] delta [integer!] *rcvGaussianFilter src dst knl 0*	Gaussian 2D Filter