

REDCV FUNCTIONS DOCUMENTATION

Most of functions are calling Red/System routines 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. All red routines are prefixed with underscore (e.g. `_rcvCopy`). 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/rcvCore.red           ; Basic image creating and processing functions  
#include %highgui/rcvHighGui.red    ; Fast Highgui functions  
#include %matrix/rcvMatrix.red       ; Matrices 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 `rcvCore` is obligatory.

More documentation to come.

REDCV CORE FUNCTIONS

Library: redCV/libs/core/ rvcvcore.red

Functions: 12

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

Library: redCV/libs/core/rcvcore.red

Functions: 43

Function	Doc String
Image Conversion	
rcv2Gray: function [src [image!] dst [image!] /average /luminosity /lightness return: [image!]] <i>rcv2Gray/average src dst</i>	Convert RGB image to Grayscale according to refinement
rcv2BGRA: function [src [image!] dst [image!]] <i>rcv2BGRA src dst</i>	Convert RGBA => BGRA
rcv2RGBA: function [src [image!] dst [image!]] <i>rcv2RGBA src dst</i>	Convert BGRA => RGBA"
rcv2BW: function [src [image!] dst [image!]] <i>rcv2BW src dst</i>	Convert RGB image => Black and White
rcv2BWFilter: function [src [image!] dst [image!] thresh [integer!]] <i>rcv2BWFilter src dst 64</i>	Convert RGB image => Black and White according to threshold
rcvSplit: function [src [image!] dst [image!]/red /green /blue] <i>rcvSplit/blue src dst (->blue channel)</i>	Split source image in RGB separate channels
rcvInvert: function [source [image!] dst [image!]] <i>rcvInvert src dst</i>	Similar to NOT image
rcvInRange: function [src [image!] dst [image!] lower [tuple!] upper [tuple!]] <i>rcvInRange img mask 130.50.50 170.255.255</i>	Extracts sub array from image according to lower and upper rgb values
Math Operators on image	
rcvAdd: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvAdd image1 image2 destImage</i>	dst: src1 + src2
rcvSub: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvSub image1 image2 destImage</i>	dst: src1 - src2
rcvMul: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvMul image1 image2 destImage</i>	dst: src1 * src2
rcvDiv: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvDiv image1 image2 destImage</i>	dst: src1 / src2
rcvMod: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvMod image1 image2 destImage</i>	dst: src1 // src2 (modulo)
rcvRem: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvRem image1 image2 destImage</i>	dst: src1 % src2 (remainder)

rcvAbsDiff: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvAbsDiff image1 image2 destImage</i>	dst: absolute difference src1 src2
Math operators with scalar (integer !)	
rcvAddS: function [src [image!] dst [image!] val [integer!]] <i>rcvAddS source destination 128</i>	dst: src + integer! value
rcvSubS: function [src [image!] dst [image!] val [integer!]] <i>rcvSubS source destination 128</i>	dst: src - integer! value
rcvMulS: function [src [image!] dst [image!] val [integer!]] <i>rcvMulS source destination 2</i>	dst: src * integer! value
rcvDivS: function [src [image!] dst [image!] val [integer!]] <i>rcvDivS source destination 2</i>	dst: src / integer! value
rcvModS: function [src [image!] dst [image!] val [integer!]] <i>rcvModS source destination 4</i>	dst: src // integer! value (modulo)
rcvRemS: function [src [image!] dst [image!] val [integer!]] <i>rcvRemS source destination 2</i>	dst: src % integer! value (remainder)
rcvPow: function [src [image!] dst [image!] val [integer!]] <i>rcvPow source destination 2</i>	dst: src ^integer! value
rcvLSH: function [src [image!] dst [image!] val [integer!]] <i>rcvLSH source destination 2</i>	Left shift image by value
rcvRSH: function [src [image!] dst [image!] val [integer!]] <i>rcvRSH source destination 2</i>	Right Shift image by value
rcvSQR: function [src [image!] dst [image!] val [integer!]] <i>rcvSQR source destination 2</i>	Image square root
Math operators with scalar (tuple!)	
rcvAddT: function [src [image!] dst [image!] val [tuple!]] <i>rcvAddT source destination 128.128.128</i>	dst: src + tuple! value
rcvSubT: function [src [image!] dst [image!] val [tuple!]] <i>rcvSubT source destination 32.32.32</i>	dst: src - tuple! value
rcvMulT: function [src [image!] dst [image!] val [tuple!]] <i>rcvMulT source destination 2.2.2</i>	dst: src * tuple! value
rcvDivT: function [src [image!] dst [image!] val [tuple!]] <i>rcvDivT source destination 2.2.2</i>	dst: src / tuple! value
rcvModT: function [src [image!] dst [image!] val [tuple!]] <i>rcvModT source destination 2.2.2</i>	dst: src // tuple! value (modulo)

rcvRemT: function [src [image!] dst [image!] val [tuple!]] <i>rcvRemT source destination 2.2.2</i>	dst: src % tuple! value (remainder)
Logical operators on Image	
rcvAND: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvAND source1 source2 destination</i>	dst: src1 AND src2
rcvOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvOR source1 source2 destination</i>	dst: src1 OR src2
rcvXOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvXOR source1 source2 destination</i>	dst: src1 XOR src2
rcvNAND: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvNAND source1 source2 destination</i>	dst: src1 NAND src2
rcvNOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvNOR source1 source2 destination</i>	dst: src1 NOR src2
rcvNXOR: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvNXOR source1 source2 destination</i>	dst: src1 NXOR rc2
rcvMIN: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvMIN source1 source2 destination</i>	dst: minimum src1 src2
rcvMAX: function [src1 [image!] src2 [image!] dst [image!]] <i>rcvMAX source1 source2 destination</i>	dst: maximum src1 src2
rcvNot: function [src [image!] dst [image!]] <i>rcvNOT source destination</i>	dst: NOT src
logical operators and scalar (tuple!) on image	
rcvANDS: function [src [image!] dst [image!] value [tuple!] return: [image!]] <i>rcvANDS source red</i>	dst: src AND tuple! as image
rcvORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] <i>rcvANDS source green</i>	dst: src OR tuple! as image
rcvXORS: function [src [image!] dst [image!] value [tuple!] return: [image!]] <i>rcvANDS source blue</i>	dst: src XOR tuple! as image

REDCV MATRICES DOCUMENTATION

Library: redCV/libs/core/ rcvMatrix.red

Functions: 31

Attention: Matrices with the same type and the same size!

Function	Doc String
rcvCreateMat: function [type [word!] bitSize [integer!] mSize [pair!] return: [vector!]] <i>mat: rcvCreateMat 'integer! 8 512x512</i>	Create 2-D matrix
rcvReleaseMat: function [mat [vector!]] <i>rcvReleaseMat mat</i>	Releases Matrix
rcvCloneMat: function [src [vector!] return: [vector!]] <i>newmat: rcvCloneMat src</i>	Returns a copy of source matrix
rcvCopyMat: function [src [vector!] dst [vector!]] <i>rcvCopyMat src dst</i>	Copy source matrix to destination matrix
rcvRandomMat: function [mat [vector!] value [integer!]] <i>rcvRandomMat mat 255</i>	Randomizes matrix
rcvColorMat: function [mat [vector!] value [integer!]] <i>rcvColorMat mat 128</i>	Sets matrix color
rcvImage2Mat: function [src [image!] mat [vector!]] <i>rcvImage2Mat src mat</i>	Red Image to 2-D Matrix Converts Red Image to grayscale before matrix transformation Only 8-bit integer image
rcvMat82Image: function [mat [vector!] dst [image!]] <i>rcvMat82Image mat dst</i>	8-bit integer matrix to Red image
rcvMat16Image: function [mat [vector!] dst [image!]] <i>rcvMat162Image mat dst</i>	16-bit integer matrix to Red image
rcvMat322Image: function [mat [vector!] dst [image!]] <i>rcvMat322Image mat dst</i>	32-bit integer matrix to Red image
rcvSplit2Mat: function [src [image!] mat0 [vector!] mat1 [vector!] mat2 [vector!] mat3 [vector!]] <i>rcvSplit2Mat img1 mat0 mat1 mat2 mat3</i>	Split Red image to 4 8-bit matrices
rcvMerge2Image: function [mat0 [vector!] mat1 [vector!] mat2 [vector!] mat3 [vector!] dst [image!]] <i>rcvMerge2Image mat0 mat1 mat2 mat3 imgD</i>	Merge 4 8-bit matrices to Red image (argb)
rcvConvolveMat: function [src [vector!] dst [vector!] mSize[pair!] kernel [block!] factor [float!] delta [float!]] <i>rcvConvolveMat mat1 mat2 img1/size mask 1.0 0.0</i>	Fast 2-D matrix convolution For 8-bit integer matrix only
rcvConvertMatScale: function [src [vector!] dst [vector!] srcScale [number!] dstScale [number!] /fast /normal]	Converts Matrix Scale 8<->16<->bit conversions /fast for 8 to 32-bits

<i>rcvConvertMatScale/fast mat1 mat2 255 32768</i>	/normal for 8 to 16 or 32-bits
rcvMatInt2Float: function [src [vector!] dst [vector!] srcScale [float!]] <i>rcvMatInt2Float source destination 255.0</i>	Converts Integer Matrix to Float [0..1] matrix
Math operator for matrix	
rcvAddMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m : rcvAddMat m1 m2</i>	Adds two matrices to a new matrice
rcvSubMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m : rcvSubMat m1 m2</i>	Subtracts two matrices to a new matrice
rcvMulMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m : rcvMulMat m1 m2</i>	Multiplies two matrices to a new matrice
rcvDivMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m : rcvDivMat m1 m2</i>	Divides two matrices and returns the result
rcvRemMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m : rcvRemMat m1 m2</i>	Remainder src1 by src2
Scalar operations directly modify vector	
rcvAddSMat: function [src [vector!] value [integer!]] <i>rcvAddSMat m1 255</i>	Adds scalar
rcvSubSMat: function [src [vector!] value [integer!]] <i>rcvSubSMat m1 255</i>	Subtracts scalar
rcvMulSMat: function [src [vector!] value [integer!]] <i>rcvMulSMat m1 255</i>	Multiplies matrix by scalar
rcvDivSMat: function [src [vector!] value [integer!]] <i>rcvDivSMat m1 16</i>	Divides matrix by scalar
rcvRemSMat: function [src [vector!] value [integer!]] <i>rcvRemSMat m1 16</i>	Remainder
Logical operators	
rcvAndMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m: rcvAndMat m1 m2</i>	dst: src1 AND src2
rcvOrMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m: rcvOrMat m1 m2</i>	dst: src1 OR src2
rcvXorMat: function [src1 [vector!] src2 [vector!] return: [vector!]] <i>m: rcvXorMat m1 m2</i>	dst: src1 XOR src2
Scalar operations directly modify vector	

rcvAndSMat: function [src [vector!] value [integer!] <i>rcvAndSMat src 127</i>	src AND value
rcvOrSMat: function [src [vector!] value [integer!] <i>rcvOrSMat src 127</i>	src OR value
rcvXorSMat: function [src [vector!] value [integer!]] <i>rcvXorSMat src 127</i>	src XOR value

REDCV IMAGE PROCESSING FUNCTIONS

Library: redCV/libs/imgproc/ rcvImgProc.red

Functions: 17

Function	Doc String
Space Color Conversion	
rcvRGB2XYZ: function [src [image!] dst [image!]] <i>rcvRGB2XYZ src dst</i>	BGR to CIE XYZ color conversion
rcvBGR2XYZ: function [src [image!] dst [image!]] <i>rcvBGR2XYZ src dst</i>	Convert BGR => CIE XYZ
rcvXYZ2RGB: function [src [image!] dst [image!]] <i>rcvXYZ2RGB src dst</i>	CIE XYZ to RGB color conversion
rcvRGB2HSV: function [src [image!] dst [image!]] <i>rcvRGB2HSV src dst</i>	RGB color to HSV conversion
rcvBGR2HSV: function [src [image!] dst [image!]] <i>rcvBGR2HSV src dst</i>	BGR color to HSV conversion
rcvBGR2YCrCb: function [src [image!] dst [image!]] <i>rcvBGR2YCrCb src dst</i>	BGR color to YCrCb conversion
rcvRGB2YCrCb: function [src [image!] dst [image!]] <i>rcvRGB2YCrCb src dst</i>	RGB color to YCrCb conversion
rcvRGB2HLS: function [src [image!] dst [image!]] <i>rcvRGB2HLS src dst</i>	RGB color to HLS conversion
rcvBGR2HLS: function [src [image!] dst [image!]] <i>rcvBGR2HLS src dst</i>	BGR color to HLS conversion
Image transformation	
rcvFlip: function [src [image!] dst [image!]] /horizontal /vertical /both return: [image!] <i>rcvFlip/horizontal src dst</i> <i>rcvFlip/vertical src dst</i> <i>rcvFlip/both src dst</i>	Left Right, Up down or both directions flip
Image Convolution	
rcvConvolve: function [src [image!] dst [image!]] kernel [block!] factor [float!] delta [float!] <i>rcvConvolve src dst noFilter 1.0 0.0</i>	Convolve an image with the kernel
rcvFastConvolve: function [src [image!] dst [image!]] Channel [integer !] kernel [block!] factor [float!] delta [float!] <i>rcvFastConvolve img1 img2 1 mask 1.0 0.0</i>	Convolves a 8-bit image with the kernel by channel <i>Convolves img1/channel 1</i>
rcvFilter2D: function [src [image!] dst [image!]] kernel [block!] delta [integer!] <i>rcvFilter2D src dst noFilter 0</i>	Basic convolution Filter
rcvFastFilter2D: function [src [image!] dst [image!]] kernel [block!] <i>rcvFastFilter2D src dst Filter</i>	Faster convolution Filter

Image Filters	
rcvMakeGaussian: function [kSize [pair!] return: [block!]] <i>knl: rcvMakeGaussian 3x3</i>	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!]] <i>rcvGaussianFilter src dst knl 0</i>	Gaussian 2D Filter
Gaussian Pyramid Decomposition	
rcvResizeImage: function [src [image!] canvas iSize [pair!]/gaussian return: [pair!]] <i>rcvResizeImage/gaussian src canvas iSize</i> <i>rcvResizeImage src canvas iSize</i>	Resizes image and applies filter for Gaussian pyramidal resizing if required. Only Gaussian 5x5 kernel is currently supported. Canvas is a base facet. If you don't call /Gaussian refinement image is just resized.

REDCV STATISTICAL FUNCTIONS

Library: redCV/libs/math/rcvStats.red

Functions: 12

Function	Doc String
rcvCountNonZero: function [arr [image! vector!] return: [integer!]] <i>n: rcvCountNonZero source</i>	Returns number of non zero values in image or matrix
rcvMean: function [arr [image! vector!] return: [tuple!] /argb] <i>mean: rcvMeanarr</i>	Returns mean value of array as a tuple rgb or argb
rcvSum: function [src [image! vector !] return: [block!] /argb] <i>sum : rcvSum source</i>	Returns sum value of image as a block rgb or argb
rcvSTD: function [src [image!]] return: [tuple!] /argb] <i>std: rcvVarImage source</i>	returns standard deviation value of array as a tuple rgb or argb
rcvMedian: function [source [image! vector !] return: [tuple!]] <i>median : rcvMedianImage source</i>	Returns median value of array as tuple argb
rcvMinValue: function [source [image! vector !] return: [tuple!]] <i>mini : rcvMinImage source</i>	Minimal value in array as a tuple argb
rcvMaxValue: function [source [image! vector !] return: [tuple!]] <i>maxi : rcvMaxImage source</i>	Maximal value in array as a tuple arg
rcvMaxLoc: function [arr [image! vector!] arrSize [pair!]return: [pair!]] <i>rcvMaxLoc arr 512x512</i>	Finds global maximum location in array
rcvMinLoc: function [arr [image! vector!] arrSize [pair!]return: [pair!]] <i>rcvMinLoc arr 512x512</i>	Finds global minimum location in array
rcvHistogram: function [arr [image! vector!] return: [vector!] /red /green /blue /alpha]	Calculates array histogram for 1 channel matrix or for rgb channels in case of color image
rcvRangeImage: function [source [image!]] return: [tuple!]] <i>range: rcvRangeImage source</i>	Range value in Image as a tuple argb
rcvSortImage: function [source [image!] dst [image!]] <i>rcvSortImage source destination</i>	Ascending image sorting

Library: redCV/libs/highgui/ rcvHighGui.red

Functions: 6

Function	Doc String
rcvNamedWindow: function [name [string!] return: [window!]] <i>s1: rcvNamedWindow "Source"</i>	Creates and shows a window
rcvDestroyWindow: function [window [face!]] <i>rcvDestroyWindow s1</i>	Destroys a window
rcvDestroyAllWindows: function [] <i>rcvDestroyAllWindows</i>	Destroys all windows
rcvResizeWindow: function [window [face!] wSize [pair!]] <i>rcvResizeWindow s1 512x512</i>	Sets window size
rcvMoveWindow: function [window [face!] position [pair!]] <i>rcvMoveWindow s1 10x10</i>	Sets window position
rcvShowImage: function [window [face!] image [image!]] <i>rcvShowImage s1 image</i>	Shows image in window