Functions Guide

Read an image

```
Mat CvInvoke.Imread(string filename, int flag)
```

Parameters:

filename: the name of the image file we want to read with the extension.

flag: specifies the color type of the loaded image. This value can be chosen from ***ImreadModes*** enumeration.

flag examples:

ImreadModes.Color: If set, always convert image to the color one. This is the **default value.**

ImreadModes.Grayscale: If set, always convert image to the grayscale one.

Convert image color type

```
Void CvInvoke.CvtColor(Mat src, Mat dst, int code)
```

Parameters:

src: input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.
dst: output image of the same size and depth as src.
code: color space conversion code. Can be chosen from ***ColorConversion*** enumeration.

Note that the default color format in OpenCV is often referred to as RGB but it is actually ***BGR*** (the bytes are reversed).

code examples:

ColorConversion.Bgr2Gray

ColorConversion.Bgr2Hsv

Display an image

```
void CvInvoke.Imshow(string name, Mat image)
```

Parameters:

name: Name of the window

image: Image to be shown

Gaussian Filter

```
Void CvInvoke.GaussianBlur(Mat src, Mat dst, Size ksize,
        double sigmaX, double sigmaY, int borderType)
```

Parameters:

src: input image; the image can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.

dst: output image of the same size and type as src.

Ksize: Gaussian kernel size. ksize.width and ksize.height can differ but they both must be **positive** and **odd**. Or, they can be zero's and then they are computed from sigma.

sigmaX: Gaussian kernel standard deviation in X direction.

sigmaY: Gaussian kernel standard deviation in Y direction; if sigmaY is zero, it is set to be equal to sigmaX, if both sigmas are zeros, they are computed
from ksize.width and ksize.height, respectively.

borderType: pixel extrapolation method. Value can be chosen from ***BorderType*** enumeration.

Laplacian Filter

```
void CvInvoke.Laplacian(Mat src, Mat dst, int ddepth, int ksize)
```

Parameters:

src: Source image.

dst: Destination image of the same size and the same number of channels as src.

ddepth: Desired depth of the destination image.

ksize: Aperture size used to compute the second-derivative filters. The size must be **positive** and **odd**.

Thresholding

```
double CvInvoke.Threshold(Mat src, Mat dst, double thresh,
                          double maxval, int type)
```

Parameters:

src: input array (single-channel, 8-bit or 32-bit floating point).

dst: output array of the same size and type as src.

thresh: threshold value.

maxval: maximum value to use with the Binary and BinaryInv thresholding types.

type: thresholding type. Can be chosen from ***ThresholdType*** enumeration.

Available Types:

- **THRESH_BINARY**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_BINARY_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

- **THRESH_TRUNC**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO**

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- **THRESH_TOZERO_INV**

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Build a structuring element

```
Mat CvInvoke.GetStructuringElement(int shape, Size ksize,
                                   Point anchor)
```

Parameters:

shape: Element shape that can be chosen from ***ElementShape*** enumeration

size: Size of the structuring element.

anchor: Anchor position within the element. The default value Point(-1,-1) means that the anchor is at the center.

Morphological operations

```
Void CvInvoke.MorphologyEx(Mat src, Mat dst, int op,Mat kernel,
 Point anchor,int iterations, int borderType, int borderValue)
```

Parameters:

src: Source image. The number of channels can be arbitrary. The depth should be one of CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.

dst: Destination image of the same size and type as src.

Op: Type of a morphological operation

Kernel: Structuring element.

anchor: Default value Point(-1,-1).

iterations: Number of times erosion and dilation are applied.

borderType: Pixel extrapolation method. You can choose value from **BorderType** enumeration

borderValue: Border value in case of a constant border. You can get the value from ***CvInvoke.MorphologyDefaultBorderValue***

Available Operations:

Opening operation:

$$\texttt{dst} = \mathrm{open}(\texttt{src}, \texttt{element}) = \mathrm{dilate}(\mathrm{erode}(\texttt{src}, \texttt{element}))$$

Closing operation:

$$\texttt{dst} = \mathrm{close}(\texttt{src}, \texttt{element}) = \mathrm{erode}(\mathrm{dilate}(\texttt{src}, \texttt{element}))$$

Morphological gradient:

$$\texttt{dst} = \mathrm{morph\_grad}(\texttt{src}, \texttt{element}) = \mathrm{dilate}(\texttt{src}, \texttt{element}) - \mathrm{erode}(\texttt{src}, \texttt{element})$$

"Top hat":

$$\texttt{dst} = \mathrm{tophat}(\texttt{src}, \texttt{element}) = \texttt{src} - \mathrm{open}(\texttt{src}, \texttt{element})$$

"Black hat":

$$\texttt{dst} = \mathrm{blackhat}(\texttt{src}, \texttt{element}) = \mathrm{close}(\texttt{src}, \texttt{element}) - \texttt{src}$$

Canny Edge Detector

```
Void CvInvoke.Canny(Mat src,Mat edges, double threshold1,double
                    threshold2,int size = 3)
```

Parameters:

src: single-channel 8-bit input image.

edges: output edge map; it has the same size and type as image.

threshold1: first threshold for the hysteresis procedure.

threshold2: second threshold for the hysteresis procedure.

apertureSize: aperture size for the Sobel operator.

Range Thresholding

```
Void CvInvoke.InRange(Mat src, ScalarArray lowerb, ScalarArray
                      upperb, Mat dst)
```

Parameters:

src: input image.

lowerb: inclusive lower boundary array.

upperb: inclusive upper boundary array.

dst: output image of the same size as src and CV_8U type.

Example: keep only pixels in the blue range.

```
CvInvoke.InRange(hsvImg, new ScalarArray(new MCvScalar(94, 80, 2)), new
ScalarArray(new MCvScalar(126, 255, 255)), binrayImg);
```

Custom Functions

Contrast Stretching

```
Mat CvFunctions.ContrastStretching(Mat src, int r1, int r2, int s1,
int s2)
```

Parameters:

src: single-channel 8-bit input image.

[r1, r2]: Contrast limits for the input image, where 255 > r2 > r1 > 0.

[s1, s2]: Contrast limits for the output image.

Returns the image after applying contrast stretching.

Contour Tracing

```
Void CvFunctions.DrawContours(Mat img, Mat src, MCvScalar
contourColor, MCvScalar centerColor)
```

Parameters:

img*:* Source, an 8-bit single-channel image. Non-zero pixels are treated as 1's. Zero pixels remain 0's, so the image is treated as binary.
You can use **compare**, **inRange**, **threshold** , **adaptiveThreshold**, **Canny**, and others to create a binary image out of a grayscale or color one.

src: input image to draw contours on.

contourColor: the color of the object contour created by BGR values.

centerColor: the color of the circle drawn in the center on the object by BGR values.

Example Red is:

```
MCvScalar red = new MCvScalar(0, 0, 255);
```