Intel® Image Processing Library — Quick Reference

The Intel® Image Processing Library is a set of C functions for performing typical image-processing tasks. Intel Image Processing Library is optimized for the Intel® architecture (IA). The library functions ensure high performance when run on IA processors, especially on those with MMX™ technology and the latest processor generations. This *Quick Reference* includes two sections: the Functions by Category section lists all the library functions divided into groups by functionality; the How Do I... section allows you to quickly choose functions for your needs.

This document describes Intel Image Processing Library release 2.5.

For more information, refer to Intel Image Processing Library Reference Manual, document number 663791-05.

Functions by Category

Error-handling Functions

Error performs basic error handling
`void iplError(IPLStatus status, const char* func, const char* context);`

ErrorStr translates an error status code into a textual description
`const char* iplErrorStr(IPLStatus status);`

GetErrMode returns the error processing mode
`int iplGetErrMode();`

GetErrStatus returns the error status code
`IPLStatus iplGetErrStatus(); /* typedef int IPLStatus */`

RedirectError assigns a new error-handling function
`IPLErrorCallBack iplRedirectError(IPLErrorCallBack iplErrorFunc);`

SetErrMode sets the error processing mode
`void iplSetErrMode(int errMode);`

SetErrStatus sets the error status
`void iplSetErrStatus(IPLStatus status);`

NullDevReport directs error reporting to the NULL device
`IPLStatus iplNullDevReport ( IPLStatus status, const char *funcname, const char *context, const char *file, int line);`

StdErrReport directs error reporting to the console
`IPLStatus iplStdErrReport ( IPLStatus status, const char *funcname, const char *context, const char *file, int line);`

GuiBoxReport directs error reporting to the message box
`IPLStatus iplGuiBoxReport ( IPLStatus status, const char *funcname, const char *context, const char *file, int line);`

Image Creation Functions

See also Memory Allocation Functions

AllocateImage allocates memory for image data using the specified header
`void iplAllocateImage(IplImage* image, int doFill, int fillValue);`

AllocateImageFP allocates memory for floating-point image data using the specified header
`void iplAllocateImageFP(IplImage* image, int doFill, float fillValue);`

DeallocateImage frees the image data memory pointed to in the image header
`void iplDeallocateImage(IplImage* image);`

Deallocate deallocates the image header or data or ROI or mask, or all four
`void iplDeallocate(IplImage* image, int flag);`

CreateImageHeader creates an image header according to the specified attributes
`IplImage* iplCreateImageHeader(int nChannels, int alphaChannel, int depth, char* colorModel, char* channelSeq, int dataOrder, int origin, int align, int width, int height, IplROI* roi, IplImage* maskROI, void* imageID, IplTileInfo* tileInfo);`

CloneImage creates a copy of an image, including its data and ROI
`IplImage* iplCloneImage(const IplImage* image);`

CheckImageHeader validates field values of an image header
`IPLStatus iplCheckImageHeader ( const IplImage* hdr );`

CreateImageJaehne creates a one-channel test image
`IplImage* iplCreateImageJaehne ( int depth, int width, int height )`

CreateROI allocates and sets the region of interest (ROI) structure
`IplROI* iplCreateROI(int coi, int xOffset, int yOffset, int width, int height);`

DeleteROI deallocates the ROI structure
```
void iplDeleteROI(IplROI* roi);
```
CreateTileInfo creates the IplTileInfo structure
```
IplTileInfo* iplCreateTileInfo(IplCallBack callBack, void* id, int width, int height);
```
DeleteTileInfo deletes the IplTileInfo structure
```
void iplDeleteTileInfo(IplTileInfo* tileInfo);
```
SetTileInfo sets the IplTileInfo structure fields
```
void iplSetTileInfo(IplTileInfo* tileInfo, IplCallBack callBack, void* id, int width, int height);
```
SetROI sets the region of interest (ROI) structure
```
void iplSetROI(IplROI* roi, int coi, int xOffset, int yOffset, int width, int height);
```
SetBorderMode sets the mode for handling the border pixels
```
void iplSetBorderMode(IplImage * src, int mode, int border, int constVal);
```

Memory Allocation Functions

See also Image Creation Functions

Free frees memory allocated by a function of the Malloc group
```
void iplFree(void * ptr);
```
Malloc allocates an 8-byte aligned memory block
```
void* iplMalloc(int size); /* size in bytes */
```
dMalloc allocates an 8-byte aligned memory block for double floating-point elements
```
double* ipldMalloc(int size); /* size in double FP elements */
```
iMalloc allocates an 8-byte aligned memory block for 32-bit double words
```
int* ipliMalloc(int size); /* size in double words */
```
sMalloc allocates an 8-byte aligned memory block for floating-point elements
```
float* iplsMalloc(int size); /* size in float elements */
```
wMalloc allocates an 8-byte aligned memory block for 16-bit words
```
short* iplwMalloc(int size); /* size in words */
```

Conversion and Data Exchange Functions

See also Windows* DIB Conversion Functions

Convert converts image data from one `IplImage` to another according to the image headers
```
void iplConvert(IplImage* srcImage, IplImage* dstImage);
```
Copy copies data from one `IplImage` to another
```
void iplCopy(IplImage* srcImage, IplImage* dstImage);
```
Exchange exchanges image data between two images
```
void iplExchange(IplImage* ImageA, IplImage* ImageB);
```
Set sets an integer value for all image pixels
```
void iplSet(IplImage* image, int fillValue);
```
SetFP sets a floating-point value for the image pixels
```
void iplSetFP(IplImage* image, float fillValue);
```
PutPixel sets the pixel with coordinates (x, y) to a new value
```
void iplPutPixel(IplImage* image, int x, int y, void* pixel);
```
GetPixel retrieves the value of the pixel with coordinates (x, y)
```
void iplGetPixel(IplImage* image, int x, int y, void* pixel);
```

Noise Generation

NoiseImage Generates noise signal and adds it to an image data.
```
IPLStatus iplNoiseImage ( IplImage* image, const IplNoiseParam* noiseParam);
```
NoiseUniformInit Initializes parameters for generating integer noise signal with uniform distribution.
```
void iplNoiseUniformInit ( IplINoiseParam* noiseParam, unsigned int seed, int low, int high);
```
NoiseUniformInitFp Initializes parameters for generating floating-point noise signal with uniform distribution.
```
void iplNoiseUniformInitFp ( IplINoiseParam* noiseParam, unsigned int seed, float low, float high);
```
NoiseGaussianInit Initializes parameters for generating integer noise signal with Gaussian distribution.
```
void iplNoiseGaussianInit ( IplINoiseParam* noiseParam, unsigned int seed, int mean, int stDev);
```
NoiseGaussianInitFp Initializes parameters for generating floating-point noise signal with Gaussian distribution.
```
void iplNoiseGaussianInitFp ( IplINoiseParam* noiseParam, unsigned int seed, float mean, float stDev);
```

Bit Depth Conversion

BitonalToGray converts a bitonal image to a gray-scale one
```
void iplBitonalToGray(IplImage* srcImage, IplImage* dstImage, int ZeroScale, int OneScale);
```

ReduceBits reduces the number of bits per channel in the image
`void iplReduceBits(IplImage* srcImage, IplImage* dstImage, int noise, int ditherType, int levels);`

Scale converts the pixel data of the image `srcImage` to the pixel data of `dstImage`. The images must have integer pixel data with different bit depths. The full range of input pixel data type is scaled (linearly mapped) to the full range of output pixel data type.
`IPLStatus iplScale (const IplImage* srcImage, IplImage* dstImage);`

ScaleFP converts the pixel data of the input image `srcImage` to the pixel data of `dstImage`. One of the images (either the input or the output) must have floating-point pixel data. The user-defined range of the floating-point pixel data (`minVal..maxVal`) is linearly mapped to the full range of the integer pixel data type.
`IPLStatus iplScaleFP (const IplImage* srcImage, IplImage* dstImage, float minVal, float maxVal);`

Color Twist

ApplyColorTwist applies a color-twist matrix to the image pixel values
`void iplApplyColorTwist(IplImage* srcImage, IplImage* dstImage, IplColorTwist* cTwist, int offset);`

CreateColorTwist creates a color twist matrix
`IplColorTwist* iplCreateColorTwist(int data[16], int scalingValue);`

DeleteColorTwist frees memory used for a color twist matrix
`void iplDeleteColorTwist(IplColorTwist* cTwist);`

SetColorTwist sets a color twist matrix for image colors conversion
`void iplSetColorTwist(IplColorTwist* cTwist, int data[16], int scalingValue);`

ColorTwistFP applies a floating-point color-twist matrix `cTwist` to the first 3 channels of the input image with floating-point pixel data.
`IPLStatus iplColorTwistFP (const IplImage* src, IplImage* dst, float* cTwist)`

Color Models Conversion

ColorToGray converts a color image to a gray-scale one
`void iplColorToGray(IplImage* srcImage, IplImage* dstImage);`

GrayToColor converts a gray-scale image to a color one
`void iplGrayToColor(IplImage* srcImage, IplImage* dstImage, float FractR, float FractG, float FractB);`

HLS2RGB converts an image from the HLS color model to the RGB color model
`void iplHLS2RGB(IplImage* hlsImage, IplImage* rgbImage);`

HSV2RGB converts an image from the HSV color model to the RGB color model
`void iplHSV2RGB(IplImage* hsvImage, IplImage* rgbImage);`

LUV2RGB converts an image from the LUV color model to the RGB color model
`void iplLUV2RGB(IplImage* luvImage, IplImage* rgbImage);`

RGB2HLS converts an image from the RGB color model to the HLS color model
`void iplRGB2HLS(IplImage* rgbImage, IplImage* hlsImage);`

RGB2HSV converts an image from the RGB color model to the HSV color model
`void iplRGB2HSV(IplImage* rgbImage, IplImage* hsvImage);`

RGB2LUV converts an image from the RGB color model to the LUV color model
`void iplRGB2LUV(IplImage* rgbImage, IplImage* luvImage);`

RGB2XYZ converts an image from the RGB color model to the XYZ color model
`void iplRGB2XYZ(IplImage* rgbImage, IplImage* xyzImage);`

RGB2YCrCb converts an image from the RGB color model to the YCrCb color model
`void iplRGB2YCrCb(IplImage* rgbImage, IplImage* YCrCbImage);`

RGB2YUV converts an image from the RGB color model to the YUV color model
`void iplRGB2YUV(IplImage* rgbImage, IplImage* yuvImage);`

XYZ2RGB converts an image from the XYZ color model to the RGB color model
`void iplXYZ2RGB(IplImage* xyzImage, IplImage* rgbImage);`

YCC2RGB converts an image from the Kodak PhotoYCC color model to the RGB color model
`void iplYCC2RGB(IplImage* YCCImage, IplImage* rgbImage);`

YCrCb2RGB converts an image from the YCrCb color model to the RGB color model
`void iplYCrCb2RGB(IplImage* YCrCbImage, IplImage* rgbImage);`

YUV2RGB converts an image from the YUV color model to the RGB color model
`void iplYUV2RGB(IplImage* yuvImage, IplImage* rgbImage);`

Windows* DIB Conversion Functions

See also Conversion and Data Exchange Functions

ConvertFromDIB converts a Windows* DIB image to an `IplImage` with specified attributes
`void iplConvertFromDIB(BITMAPINFOHEADER* dib, IplImage* image);`

ConvertFromDIBSep converts a Windows* DIB image to an `IplImage` using the DIB header and data stored separately
`IPLStatus iplConvertFromDIBSep(BITMAPINFOHEADER* dibHeader, const char* dibData, IplImage* image);`

ConvertToDIB converts an `IplImage` to a Windows DIB image with specified attributes
```
void iplConvertToDIB(iplImage* image, BITMAPINFOHEADER* dib, int dither, int paletteConversion);
```

ConvertToDIBSep converts an IplImage to a DIB image; uses two separate parameters for the DIB header and data
```
void iplConvertToDIB(iplImage* image, BITMAPINFOHEADER* dib, char* dibData, int dither, int paletteConversion);
```

TranslateDIB translates a Windows DIB image into an `IplImage`
```
iplImage* iplTranslateDIB(BITMAPINFOHEADER* dib, BOOL* cloneData);
```

Arithmetic Functions

See also Alpha-blending Functions

Abs returns absolute pixel values of the source image
```
void iplAbs(IplImage* srcImage, IplImage* dstImage);
```

Add adds pixel values of two images
```
void iplAdd(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage);
```

AddS adds an integer constant to pixel values of the source image
```
void iplAddS(IplImage* srcImage, IplImage* dstImage, int value);
```

AddSFP adds a floating-point constant to pixel values of the source image
```
void iplAddSFP(IplImage* srcImage, IplImage* dstImage, float value);
```

Multiply multiplies pixel values of two images
```
void iplMultiply(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage);
```

MultiplyS multiplies pixel values of the source image by an integer constant
```
void iplMultiplyS(IplImage* srcImage, IplImage* dstImage, int value);
```

MultiplySFP multiplies pixel values of the source image by a floating-point constant
```
void iplMultiplySFP(IplImage* srcImage, IplImage* dstImage, float value);
```

MultiplyScale multiplies pixel values of two images and scales the products
```
void iplMultiplyScale(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage);
```

MultiplySScale multiplies pixel values of the source image by a constant and scales the products
```
void iplMultiplySScale(IplImage* srcImage, IplImage* dstImage, int value);
```

Square squares the pixel values of the source image
```
void iplSquare(IplImage* srcImage, IplImage* dstImage);
```

Subtract subtracts pixel values of two images
```
void iplSubtract(IplImage* srcImage, IplImage* dstImage, IplImage* dstImage);
```

SubtractS subtracts an integer constant from pixel values, or pixel values from the constant
```
void iplSubtractS(IplImage* srcImage, IplImage* dstImage, int value, BOOL flip);
```

SubtractSFP subtracts a floating-point constant from pixel values, or pixel values from the constant
```
void iplSubtractSFP(IplImage* srcImage, IplImage* dstImage, float value, BOOL flip);
```

Logical Functions

And computes a bitwise AND of pixel values of two images
```
void iplAnd(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage);
```

AndS computes a bitwise AND of each pixel's value and a constant
```
void iplAndS(IplImage* srcImage, IplImage* dstImage, unsigned int value);
```

LShiftS shifts pixel values' bits to the left
```
void iplLShiftS(IplImage* srcImage, IplImage* dstImage, unsigned int nShift);
```

Not computes a bitwise NOT of pixel values
```
void iplNot(IplImage* srcImage, IplImage* dstImage);
```

Or computes a bitwise OR of pixel values of two images
```
void iplOr(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage);
```

OrS computes a bitwise OR of each pixel’s value and a constant
```
void iplOrS(IplImage* srcImage, IplImage* dstImage, unsigned int value);
```

RShiftS divides pixel values by a constant power of 2 by shifting bits to the right
```
void iplRShiftS(IplImage* srcImage, IplImage* dstImage, unsigned int nShift);
```

Xor computes a bitwise XOR of pixel values of two images
```
void iplXor(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage);
```

XorS computes a bitwise XOR of each pixel’s value and a constant
```
void iplXorS(IplImage* srcImage, IplImage* dstImage, unsigned int value);
```

Alpha-blending Functions

AlphaComposite composites two images using alpha (opacity) values
```
void iplAlphaComposite(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, int compositeType,
IplImage* alphaImageA, IplImage* alphaImageB, IplImage* alphaImageDst, BOOL premulAlpha, BOOL divideMode);
```

AlphaCompositeC composites two images using a constant alpha value
```
void iplAlphaCompositeC(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, int compositeType, int
aA, int aB, BOOL premulAlpha, BOOL divideMode);
```

PreMultiplyAlpha pre-multiplies pixel values of an image by alpha value(s)
```
void iplPreMultiplyAlpha (IplImage* image, int alphaValue);
```

Image Filtering Functions

Blur applies a simple neighborhood averaging filter to blur the image
```
void iplBlur(IplImage* srcImage, IplImage* dstImage, int nCols, int nRows, int anchorX, int anchorY);
```

Convolve2D convolves an image with integer convolution kernel(s)
```
void iplConvolve2D(IplImage* srcImage, IplImage* dstImage, IplConvKernel** kernel, int nKernels, int
combineMethod);
```

Convolve2DFP convolves an image with floating-point convolution kernel(s)
```
void iplConvolve2DFP(IplImage* srcImage, IplImage* dstImage, IplConvKernelFP** kernel, int nKernels, int
combineMethod);
```

ConvolveSep2D convolves an image with a separable convolution kernel
```
void iplConvolveSep2D(IplImage* srcImage, IplImage* dstImage, IplConvKernel* xKernel, IplConvKernel*
yKernel);
```

ConvolveSep2DFP convolves an image with a separable floating-point kernel
```
void iplConvolveSep2DFP (IplImage* srcImage, IplImage* dstImage, IplConvKernelFP* xKernel, IplConvKernelFP*
yKernel);
```

CreateConvKernel creates an integer convolution kernel
```
IplConvKernel* iplCreateConvKernel(int nCols, int nRows, int anchorX, int anchorY, int* values, int nShiftR);
```

CreateConvKernelChar creates an integer convolution kernel using char input for the kernel values
```
IplConvKernel* iplCreateConvKernelChar(int nCols, int nRows, int anchorX, int anchorY, char* values, int
nShiftR);
```

CreateConvKernelFP creates a floating-point convolution kernel
```
IplConvKernelFP* iplCreateConvKernelFP(int nCols, int nRows, int anchorX, int anchorY, float* values);
```

DeleteConvKernel deletes the convolution kernel
```
void iplDeleteConvKernel(IplConvKernel* kernel);
```

DeleteConvKernelFP deletes the convolution kernel
```
void iplDeleteConvKernelFP(IplConvKernelFP* kernel);
```

GetConvKernel reads the attributes of an integer convolution kernel
```
void iplGetConvKernel(IplConvKernel* kernel, int* nCols, int* nRows, int* anchorX, int* anchorY, int** values,
int* nShiftR);
```

GetConvKernelChar reads the attributes of a convolution kernel previously created by CreateConvKernelChar
```
void iplGetConvKernelChar(IplConvKernel* kernel, int* nCols, int* nRows, int* anchorX, int* anchorY, char**
values, int* nShiftR);
```

GetConvKernelFP reads the attributes of a floating-point convolution kernel
```
void iplGetConvKernelFP(IplConvKernelFP* kernel, int* nCols, int* nRows, int* anchorX, int* anchorY,
float** values);
```

MaxFilter applies a maximum filter to an image
```
void iplMaxFilter(IplImage* srcImage, IplImage* dstImage, int nCols, int nRows, int anchorX, int anchorY);
```

MedianFilter applies a median filter to an image
```
void iplMedianFilter(IplImage* srcImage, IplImage* dstImage, int nCols, int nRows, int anchorX, int anchorY);
```

ColorMedianFilter applies a color median filter to an image
```
void iplColorMedianFilter(IplImage* srcImage, IplImage* dstImage, int nCols, int nRows, int anchorX, int
anchorY);
```

MinFilter applies a minimum filter to an image
```
void iplMinFilter(IplImage* srcImage, IplImage* dstImage, int nCols, int nRows, int anchorX, int anchorY);
```

FixedFilter applies a commonly used (predefined) filter to an image
```
void iplFixedFilter(IplImage* srcImage, IplImage* dstImage, IplFilter filter);
```

Fast Fourier and Discrete Cosine Transforms

CcsFft2D computes the 2D fast Fourier transform of complex data
```
void iplCcsFft2D(IplImage* srcImage, IplImage* dstImage, int flags);
```

DCT2D computes the forward or inverse 2D discrete cosine transform of an image
```
void iplDCT2D(IplImage* srcImage, IplImage* dstImage, int flags);
```

RealFft2D computes the forward or inverse 2D fast Fourier transform of a real image
`void iplRealFft2D(IplImage* srcImage, IplImage* dstImage, int flags);`

MpyRCPack2D multiplies the data of the image `srcA` by that of `srcB` and writes the result to `dst`. All images are assumed to be in the RCPack format (the format for storing the results of forward FFTs).
`void iplMpyRCPack2D (IplImage* srcA, IplImage* srcB, IplImage* dst);`

Morphological Operations

Close performs a number of dilations followed by the same number of erosions of an image
`void iplClose(IplImage* srcImage, IplImage* dstImage, int nIterations);`

Dilate sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors
`void iplDilate(IplImage* srcImage, IplImage* dstImage, int nIterations);`

Erode sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors
`void iplErode(IplImage* srcImage, IplImage* dstImage, int nIterations);`

Open performs a number of erosions followed by the same number of dilations of an image
`void iplOpen(IplImage* srcImage, IplImage* dstImage, int nIterations);`

Histogram and Thresholding Functions

ComputeHisto computes the image intentsity histogram
`void iplComputeHisto(IplImage* srcImage, IplLUT** lut);`

ContrastStretch stretches the constrast of an image using an intensity transformation
`void iplContrastStretch(IplImage* srcImage, IplImage* dstImage, IplLUT** lut);`

HistoEqualize equalizes the image intensity histogram
`void iplHistoEqualize(IplImage* srcImage, IplImage* dstImage, IplLUT** lut);`

Threshold performs a simple thresholding of an image
`void iplThreshold(IplImage* srcImage, IplImage* dstImage, int threshold);`

Compare Functions

Greater tests if the pixel values of the first input image are greater than those of the second input image, and sets the corresponding output pixels to 1 (greater) or 0 (not greater).
`IPLStatus iplGreater (IplImage* img1, IplImage* img2, IplImage* dst);`

Less tests if the pixel values of the first input image are less than those of the second input image, and sets the corresponding output pixels to 1 (less) or 0 (not less).
`IPLStatus iplLess (IplImage* img1, IplImage* img2, IplImage* dst);`

Equal tests if the pixel values of the first input image are equal to those of the second input image, and sets the corresponding output pixels to 1 (equal) or 0 (not equal).
`IPLStatus iplEqual (IplImage* img1, IplImage* img2, IplImage* dst);`

EqualFPEps tests if the pixel values of the first input image are equal to those of the second input image within a tolerance `eps`, and sets the corresponding output pixels to 1 (equal) or 0 (not equal).
`IPLStatus iplEqualFPEps (IplImage* img1, IplImage* img2, IplImage* dst, float eps);`

GreaterS tests if the input image's pixel values are greater than an integer `s`, and sets the corresponding output pixels to 1 (greater) or 0 (not greater).
`IPLStatus iplGreaterS (IplImage* src, int s, IplImage* dst);`

LessS tests if the input image's pixel values are less than an integer `s`, and sets the corresponding output pixels to 1 (less) or 0 (not less).
`IPLStatus iplLessS (IplImage* src, int s, IplImage* dst);`

EqualS tests if the input image's pixel values are equal to an integer `s`, and sets the corresponding output pixels to 1 (equal) or 0 (not equal).
`IPLStatus iplEqualS (IplImage* src, int s, IplImage* dst);`

GreaterSFP tests if the input image's pixel values are greater than a floating-point value `s`, and sets the corresponding output pixels to 1 (greater) or 0 (not greater).
`IPLStatus iplGreaterSFP (IplImage* src, float s, IplImage* dst);`

LessSFP tests if the input image's pixel values are less than a floating-point value `s`, and sets the corresponding output pixels to 1 (less) or 0 (not less).
`IPLStatus iplLessSFP (IplImage* src, float s, IplImage* dst);`

EqualSFP tests if the input image's pixel values are equal to a floating-point value `s`, and sets the corresponding output pixels to 1 (equal) or 0 (not equal).
`IPLStatus iplEqualSFP (IplImage* src, float s, IplImage* dst);`

EqualSFPEps tests if the input image's pixel values are equal to a floating-point value `s` within a tolerance `eps`, and sets the corresponding output pixels to 1 (equal) or 0 (not equal).
`IPLStatus iplEqualSFP (IplImage* src, float s, IplImage* dst, float eps);`

Geometric Transformation Functions

Decimate shrinks (decimates) the image
`void iplDecimate(IplImage* srcImage, IplImage* dstImage, int xDst, int xSrc, int yDst, int ySrc, int interpolate);`

DecimateBlur blurs the input image using an `xMaskSize` by `yMaskSize` mask, and then decimates the image
`void iplDecimateBlur (IplImage* srcImage, IplImage* dstImage, int xDst, int xSrc, int yDst, int ySrc, int interpolate, int xMaskSize, int yMaskSize);`

Mirror finds a mirror image
`void iplMirror(IplImage* srcImage, IplImage* dstImage, int flipAxis);`

Zoom magnifies (zooms) the image
`void iplZoom(IplImage* srcImage, IplImage* dstImage, int xDst, int xSrc, int yDst, int ySrc, int interpolate);`

Resize resizes the image
`void iplResize(IplImage* srcImage, IplImage* dstImage, int xDst, int xSrc, int yDst, int ySrc, int interpolate);`

Rotate rotates and shifts the image
`void iplRotate(IplImage* srcImage, IplImage* dstImage, double angle, double xShift, double yShift, int interpolate);`

GetRotateShift computes shifts to be passed to `iplRotate` for rotating the image by the specified angle, around the specified center
`void iplGetRotateShift(double xCenter, double yCenter, double angle, double *xShift, double *yShift);`

Shear performs a shear of the source image
`void iplShear(IplImage* srcImage, IplImage* dstImage, double xShear, double yShear, double xShift, double yShift, int interpolate);`

Remap fills the pixels in the output image `dstImage` using the values from `srcImage`'s points with coordinates (`xMap`, `yMap`). Both `xMap` and `yMap` must be 1-channel images with floating-point data.
`void iplRemap (IplImage* srcImage, IplImage* xMap, IplImage* yMap, IplImage* dstImage, int interpolate);`

WarpAffine performs an affine transform with the specified coefficients
`void iplWarpAffine(IplImage* srcImage, IplImage* dstImage, const double coeffs[2][3], int interpolate);`

WarpBilinear performs a bilinear transform with the specified coefficients
`void iplWarpBilinear(IplImage* srcImage, IplImage* dstImage, const double coeffs[2][4], int warpFlag, int interpolate);`

WarpBilinearQ performs a bilinear transform mapping the source ROI to the specified quadrangle, or mapping the specified source quadrangle to the destination ROI
`void iplWarpBilinearQ(IplImage* srcImage, IplImage* dstImage, const double quad[4][2], int warpFlag, int interpolate);`

WarpPerspective performs a perspective transform with the specified coefficients
`void iplWarpPerspective(IplImage* srcImage, IplImage* dstImage, const double coeffs[3][3], int warpFlag, int interpolate);`

WarpPerspectiveQ performs a perspective transform mapping the source ROI to the specified quadrangle, or mapping the specified source quadrangle to the destination ROI
`void iplWarpPerspectiveQ(IplImage* srcImage, IplImage* dstImage, const double quad[4][2], int warpFlag, int interpolate);`

GetAffineBound computes the bounding rectangle for the image's ROI transformed by WarpAffine
`void iplGetAffineBound(IplImage* image, const double coeffs[2][3], double rect[2][2]);`

GetAffineQuad computes coordinates of the quadrangle to which the image's ROI is mapped by WarpAffine
`void iplGetAffineQuad(IplImage* image, const double coeffs[2][3], double quad[4][2]);`

GetAffineTransform computes the WarpAffine transform coefficients, given the quadrangle to which the ROI is transformed
`void iplGetAffineTransform(IplImage* image, double coeffs[2][3], const double quad[4][2]);`

GetBilinearBound computes the bounding rectangle for the image's ROI transformed by WarpBilinear
`void iplGetBilinearBound(IplImage* image, const double coeffs[2][4], double rect[2][2]);`

GetBilinearQuad computes coordinates of the quadrangle to which the image's ROI is mapped by WarpBilinear
`void iplGetBilinearQuad(IplImage* image, const double coeffs[2][4], double quad[4][2]);`

GetBilinearTransform computes the WarpBilinear transform coefficients, given the quadrangle to which the ROI is transformed
`void iplGetBilinearTransform(IplImage* image, double coeffs[2][4], const double quad[4][2]);`

GetPerspectiveBound computes the bounding rectangle for the image's ROI transformed by WarpPerspective
`void iplGetPerspectiveBound(IplImage* image, const double coeffs[3][3], double rect[2][2]);`

GetPerspectiveQuad computes coordinates of the quadrangle to which the image's ROI is mapped by WarpPerspective
`void iplGetPerspectiveQuad(IplImage* image, const double coeffs[3][3], double quad[4][2]);`

GetPerspectiveTransform computes the WarpPerspective transform coefficients, given the quadrangle to which the ROI is mapped
`void iplGetPerspectiveTransform(IplImage* image, double coeffs[3][3], const double quad[4][2]);`

Norms and Moments

Norm computes the C, L_1 or L_2 norm of the image's pixel values or of the differences in pixel values of two images
```
double iplNorm(IplImage* srcImageA, IplImage* srcImageB, int normType);
```

Moments computes spatial moments (from order 0 to order 3) for an image
```
void iplMoments(IplImage* image, IplMomentState mState);
```

GetCentralMoment returns the central moment of the specified order (0 to 3) previously computed by Moments
```
double iplGetCentralMoment(IplMomentState mState, int mOrd, int nOrd);
```

GetNormalizedCentralMoment returns the normalized central moment of the specified order (0 to 3) computed by Moments
```
double iplGetNormalizedCentralMoment(IplMomentState mState, int mOrd, int nOrd);
```

GetSpatialMoment returns the spatial moment of the specified order (0 to 3) previously computed by Moments
```
double iplGetSpatialMoment(IplMomentState mState, int mOrd, int nOrd);
```

GetNormalizedSpatialMoment returns the normalized spatial moment of the specified order (0 to 3) computed by Moments
```
double iplGetNormalizedSpatialMoment(IplMomentState mState, int mOrd, int nOrd);
```

CentralMoment computes the central moment of the specified order (0 to 3)
```
double iplCentralMoment(IplImage* image, int mOrd, int nOrd);
```

NormalizedCentralMoment computes the normalized central moment of the specified order (0 to 3)
```
double iplNormalizedCentralMoment(IplImage* image, int mOrd, int nOrd);
```

SpatialMoment computes the spatial moment of the specified order (0 to 3)
```
double iplSpatialMoment(IplImage* image, int mOrd, int nOrd);
```

NormalizedSpatialMoment computes the normalized spatial moment of the specified order (0 to 3)
```
double iplNormalizedSpatialMoment(IplImage* image, int mOrd, int nOrd);
```

NormCrossCorr Computes normalized cross-correlation between an image and a template.
```
IPLStatus iplNormCrossCorr (IplImage* srcImage, IplImage* tplImage, IplImage* dstImage);
```

MinMaxFP determines the minimum and maximum pixel values in the image.
```
IPLStatus MinMaxFP (const IplImage* srcImage, float* min, float* max);
```

User Defined Functions

UserProcess Calls user-defined function `cbFunc` of type `IplUserFunc` to separately process each channel value of pixels in an image with integer data.
```
void iplUserProcess( IplImage* srcImage, IplImage* dstImage, IplUserFunc cbFunc );
```

UserProcessFP Calls user-defined function `cbFunc` of type `IplUserFuncFP` to separately process each channel value of pixels in images with all data types.
```
void iplUserProcessFP( IplImage* srcImage, IplImage* dstImage, IplUserFuncFP cbFunc );
```

UserProcessPixel Calls user-defined function `cbFunc` of type `IplUserFuncPixel` to simultaneously process channel values of pixels in an image.
```
void iplUserProcessPixel( IplImage* srcImage, IplImage* dstImage, IplUserFuncPixel cbFunc );
```

Library Version

GetLibVersion retrieves information about the current version of the Image Processing Library.
```
const IPLLibVersion* iplGetLibVersion(void);
```

How Do I...

A-B

add a constant to pixel values, see AddS, AddSFP in Arithmetic Functions

add a noise signal to image pixel values, see NoiseImage in Conversion and Data Exchange Functions

add pixel values of two images, see Add in Arithmetic Functions

allocate a quadword-aligned memory block, see Malloc in Memory Allocation Functions

allocate image data, see AllocateImage, AllocateImageFP in Image Creation Functions

allocate memory for 16-bit words, see wMalloc in Memory Allocation Functions

allocate memory for 32-bit double words, see iMalloc in Memory Allocation Functions

allocate memory for double floating-point elements, see dMalloc in Memory Allocation Functions

allocate memory for floating-point elements, see sMalloc in Memory Allocation Functions

AND pixel values (bitwise), see And, AndS in Logical Functions

apply a color twist matrix, see ApplyColorTwist , ColorTwistFP in Conversion and Data Exchange Functions

assign a new error-handling function, see RedirectError in Error-Handling Functions

average neighboring pixels, see Blur, MedianFilter in Filtering Functions

blur the image, see Blur, MedianFilter in Filtering Functions

C

change the image orientation, see Rotate, Mirror in Geometric Transformation Functions

change the image size, see Zoom, Decimate, Resize in Geometric Transformation Functions

clone images, see CloneImage in Image Creation Functions

composite images using the alpha channel, see AlphaComposite, AlphaCompositeC in Alpha-blending Functions

compare pixel values and a constant, see Compare Functions

compare pixel values in two images, see Compare Functions

compute absolute pixel values, see Abs in Arithmetic Functions

compute bitwise AND of pixel values and a constant, see AndS in Logical Functions

compute bitwise AND of pixel values of two images, see And in Logical Functions

compute bitwise NOT of pixel values, see Not in Logical Functions

compute bitwise OR of pixel values and a constant, see OrS in Logical Functions

compute bitwise OR of pixel values of two images, see Or in Logical Functions

compute bitwise XOR of pixel values and a constant, see XorS in Logical Functions

compute bitwise XOR of pixel values of two images, see Xor in Logical Functions

compute cross-correlation between an image and a template, see NormCrossCorr in Norms and Moments

compute fast Fourier transform of complex data, see CcsFft2D in Fast Fourier and Discrete Cosine Transform Functions

compute discrete cosine transform, see DCT2D in Fast Fourier and Discrete Cosine Transform Functions

compute moments, see Norms and Moments

compute fast Fourier transform of a real image, see RealFft2D in Fast Fourier and Discrete Cosine Transform Functions

compute the image histogram, see ComputeHisto in Histogram and Thresholding Functions

compute the norm of pixel values, see Square in Norms and Moments

convert a bitonal image to a gray-scale image, see BitonalToGray in Conversion and Data Exchange Functions

convert a color image to a gray-scale image, see ColorToGray in Conversion and Data Exchange Functions

convert a gray-scale image to a color image, see GrayToColor in Conversion and Data Exchange Functions

convert colors, see Color Twist and Color Models Conversion in Conversion and Data Exchange Functions

convert DIB images to IplImage structures (changing attributes), see ConvertFromDIB in Windows DIB Conversion Functions

convert DIB images to IplImage structures (preserving attributes), see TranslateDIB in Windows DIB Conversion Functions

convert images with scaling, see Scale, ScaleFP in Conversion and Data Exchange Functions

convert IplImage to DIB, see ConvertToDIB , ConvertToDIBSep in Windows DIB Conversion Functions

convert one IplImage to another, see Convert in Conversion and Data Exchange Functions

convert RGB images to and from other color spaces, see Color Models Conversion in Conversion and Data Exchange Functions

convolve an image with 2D kernel, see Convolve2D, Convolve2DFP in Filtering Functions

D

E

F

G-H

I-N

multiply images' data packed in RCPack format, see MpyRCPack2D in Filtering Functions

NOT pixel values (bitwise), see Not in Logical Functions

O-R

OR pixel values (bitwise), see Or, OrS in Logical Functions

pre-multiply pixel values by alpha values, see PreMultiplyAlpha in Alpha-Blending Functions

process image channel values separately with user-defined function, see UserProcess, UserProcessFP in User-Defined Functions

process image channel values simultaneously with user-defined function, see UserProcessPixel in User-Defined Functions

produce error messages for users, see ErrorStr in Error-Handling Functions

read convolution kernel's attributes, see GetConvKernel in Filtering Functions

redirect error reporting, see NullDevReport, StdErrReport, GuiBoxReport in Error-Handling Functions

reduce the image bit resolution, see ReduceBits in Conversion and Data Exchange Functions

remap an image, see Remap in Geometric Transformation Functions

report an error, see Error in Error-Handling Functions

resize the image, see Resize in Geometric Transformation Functions

rotate the image, see Rotate in Geometric Transformation Functions

S

set a color twist matrix, see SetColorTwist in Conversion and Data Exchange Functions

set a region of interest (ROI), see SetROI in Image Creation Functions

set error-handling mode, see SetErrMode in Error-Handling Functions

set each pixel to the maximum of its 8 neighbors and itself, see Dilate in Morphological Operations

set each pixel to the minimum of its 8 neighbors and itself, see Erode in Morphological Operations

set pixel (x, y) to a new value, see PutPixel in Conversion and Data Exchange Functions

set pixels to a constant value, see Set, SetFP in Conversion and Data Exchange Functions

set pixels to the maximum value of the neighbors, see MaxFilter in Filtering Functions

set pixels to the median value of the neighbors, see MedianFilter in Filtering Functions

set pixels to the minimum value of the neighbors, see MinFilter in Filtering Functions

set the error status code, see SetErrStatus in Error-Handling Functions

set the image border mode, see SetBorderMode in Image Creation Functions

set the IplTileInfo structure fields, see SetTileInfo in Image Creation Functions

shift the pixel bits to the left, see LShiftS in Logical Functions

shift the pixel bits to the right, see RShiftS in Logical Functions

shrink the image, see Decimate in Geometric Transformation Functions

square pixel values, see Square in Arithmetic Functions

stretch the image contrast, see ContrastStretch in Histogram and Thresholding Functions

subtract pixel values from a constant, or a constant from pixel values, see SubtractS, SubtractSFP in Arithmetic Functions

subtract pixel values of two images, see Subtract in Arithmetic Functions

T-Z

threshold the source image, see Threshold in Histogram and Thresholding Functions

twist image colors, see Color Twist in Conversion and Data Exchange Functions

validate image header fields, see CheckImageHeader in Image Creation Functions

warp the image, see WarpAffine, WarpBilinear, WarpPerspective in Geometric Transformation Functions

XOR pixel values (bitwise), see Xor, XorS in Logical Functions

zoom the image, see Zoom in Geometric Transformation Functions

Document number 673261-005