Intel Image Processing Library — Quick Reference

The Intel Image Processing Library (IPL) is optimized for the Intel Architecture (IA). The IPL functions ensure high performance when run on IA processors, especially on those with MMX™ technology. This ***Quick Reference*** describes 1.0 Beta 2 release of the library. It includes two sections: the *Functions by Category* section lists all IPL functions divided into groups by functionality; the *How Do I* section allows you to choose IPL functions for your needs. For more information, refer to ***Intel Image Processing Library Reference Manual***, order number 663791B.

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
```
IPLErrCallBack iplRedirectError(IPLStatus status);
```
SetErrMode sets the error processing mode
```
void iplSetErrMode(int errMode);
```
SetErrStatus sets the error status
```
void iplSetErrStatus(IPLStatus status);
```

Image Creation Functions (see also **Memory Allocation Functions**)

AllocateImage allocates memory for image data using the specified header
```
void iplAllocateImage(IplImage* image, int fillValue);
```
DeallocateImage frees the image data memory pointed to in the image header
```
void iplDeallocateImage(IplImage* image);
```
Deallocate frees the memory for image header or data or ROI or all three
```
void iplDeallocate(IplImage* image, int flag);
```
CreateImageHeader creates an IPL 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 height, int width, IplROI* roi);
```
CreateROI allocates and sets the region of interest (ROI) structure
```
IplROI* iplCreateROI(int coi, int xOffset, int yOffset, int height, int width);
```
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, int width, int height);
```
SetROI sets the region of interest (ROI) structure
```
void iplSetROI(IplROI* roi, int coi, int xOffset, int yOffset, int height, int width);
```
SetBorderMode sets the mode for handling the border pixels
```
void iplSetBorderMode(IplImage * src, int mode, int border, int constVal);
```

Windows* DIB Conversion Functions (see also **Conversion and Data Exchange Functions**)

ConvertFromDIB converts a Windows* DIB image to an IPL image with specified attributes
```
void iplConvertFromDIB(BITMAPINFOHEADER* dib, IplImage* image);
```
ConvertToDIB converts an IPL image to a Windows DIB image with specified attributes
```
void iplConvertToDIB(iplImage* image, BITMAPINFOHEADER* dib, int dither, int paletteConversion);
```
TranslateDIB translates a Windows DIB image into an IPL image
```
iplImage* iplTranslateDIB(BITMAPINFOHEADER* dib, BOOL cloneData);
```

Memory Allocation Functions (see also **Image Creation Functions**)

Free frees memory allocated by a function of the Malloc group
```
void iplMalloc(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
```
short* ipldMalloc(int size); /* size in double FP elements */
```
iMalloc allocates an 8-byte aligned memory block for 32-bit double words
```
short* 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**)

ApplyColorTwist converts an image from one color model to another by using a color twist matrix
```
void iplApplyColorTwist(IplImage* srcImage, IplImage* dstImage, IplColorTwist* cTwist, int offset,
IplCoord* map);
```
BitonalToGray converts a bitonal image to a gray-scale one
```
void iplBitonalToGray(IplImage* srcImage, IplImage* dstImage, int ZeroScale, int OneScale,
IplCoord* map);
```
ColorToGray converts a color image to a gray-scale one
```
void iplColorToGray(IplImage* srcImage, IplImage* dstImage, IplCoord* map);
```
Convert converts image data from one IPL image to another according to the image headers
```
void iplConvert(IplImage* srcImage, IplImage* dstImage, int convertMode, IplCoord* map);
```
Copy copies data from one IPL image to another
```
void iplCopy(IplImage* srcImage, IplImage* dstImage, IplCoord* map);
```
CreateColorTwist creates a color twist matrix for image conversion between color models
```
IplColorTwist* iplCreateColorTwist(int data[16], int scalingValue);
```
DeleteColorTwist frees memory used for a color twist matrix
```
void iplDeleteColorTwist(IplColorTwist* cTwist);
```
Exchange exchanges image data between two IPL images
```
void iplExchange(IplImage* ImageA, IplImage* ImageB, IplCoord* map);
```
GrayToColor converts a gray-scale image to a color one
```
void iplGrayToColor (IplImage* srcImage, IplImage* dstImage, float FractR, float FractG, float FractB,
IplCoord* map);
```
HLS2RGB converts an image from the HLS color model to the RGB color model
```
void iplHLS2RGB(IplImage* hlsImage, IplImage* rgbImage, IplCoord* map);
```
HSV2RGB converts an image from the HSV color model to the RGB color model
```
void iplHSV2RGB(IplImage* hsvImage, IplImage* rgbImage, IplCoord* map);
```
ReduceBits reduces the number of bits per channel in the image
```
void iplReduceBits(IplImage* srcImage, IplImage* dstImage, int ditherType, int jitterType, int levels,
IplCoord* map);
```
RGB2HLS converts an image from the RGB color model to the HLS color model
```
void iplRGB2HLS(IplImage* rgbImage, IplImage* hlsImage, IplCoord* map);
```
RGB2HSV converts an image from the RGB color model to the HSV color model
```
void iplRGB2HSV(IplImage* rgbImage, IplImage* hsvImage, IplCoord* map);
```
Set sets a value for an IPL image's pixel data
```
void iplSet(IplImage* image, int fillValue, IplCoord* map);
```
SetColorTwist sets a color twist matrix for image conversion between color models
```
void iplSetColorTwist(IplColorTwist* cTwist,  int data[16], int scalingValue, IplCoord* map);
```

Logical Functions

And computes a bitwise AND of pixel values of two IPL images
```
void iplAdd(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, IplCoord* map);
```
AndS computes a bitwise AND of each pixel's value and a constant
```
void iplAndS(IplImage* srcImage, IplImage* dstImage, unsigned int value, IplCoord* map);
```
LShiftS shifts pixel values' bits to the left
```
void iplLShiftS(IplImage* srcImage, IplImage* dstImage, unsigned int nShift, IplCoord* map);
```

Logical Functions (continued)

Not computes a bitwise NOT of pixel values
```
void iplNot(IplImage* srcImage, IplImage* dstImage, IplCoord* map);
```
Or computes a bitwise OR of pixel values of two IPL images
```
void iplOr(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, IplCoord* map);
```
OrS computes a bitwise OR of each pixel’s value and a constant
```
void iplOrS(IplImage* srcImage, IplImage* dstImage, unsigned int value, IplCoord* map);
```
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, IplCoord* map);
```
Xor computes a bitwise XOR of pixel values of two IPL images
```
void iplXor(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, IplCoord* map);
```
XorS computes a bitwise XOR of each pixel’s value and a constant
```
void iplXorS(IplImage* srcImage, IplImage* dstImage, unsigned int value, IplCoord* map);
```

Arithmetic Functions

Add adds pixel values of two IPL images
```
void iplAdd(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, IplCoord* map);
```
AddS adds a constant to pixel values of the source image
```
void iplAddS(IplImage* srcImage, IplImage* dstImage, int value, IplCoord* map);
```
Multiply multiplies pixel values of two IPL images
```
void iplAdd(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, IplCoord* map);
```
MultiplyS multiplies pixel values of the source image by a constant
```
void iplMultiplyS(IplImage* srcImage, IplImage* dstImage, unsigned int value, IplCoord* map);
```
MultiplyScale multiplies pixel values of two IPL images and scales the products
```
void iplMultiplyScale(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, IplCoord* map);
```
MultiplySScale multiplies pixel values of the source image by a constant and scales the products
```
void iplMultiplySScale(IplImage* srcImage, IplImage* dstImage, int value, IplCoord* map);
```
Square squares the pixel values of the source image
```
void iplSquare(IplImage* srcImage, IplImage* dstImage, IplCoord* map);
```
Subtract subtracts pixel values of two IPL images
```
void iplSubtract(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage, BOOL flip, IplCoord*
map);
```
SubtractS subtracts a constant from pixel values, or pixel values from a constant
```
void iplSubtractS(IplImage* srcImage, IplImage* dstImage, int value, BOOL flip, IplCoord* map);
```

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, IplCoord* map);
```
AlphaCompositeC composites two images using a constant alpha (opacity) value
```
void iplAlphaCompositeC(IplImage* srcImageA, IplImage* srcImageB, IplImage* dstImage,
int compositeType, int aA, int aB, BOOL premulAlpha, BOOL divideMode, IplCoord* map);
```
PreMultiplyAlpha pre-multiplies pixel values of an IPL image by alpha (opacity) value(s)
```
void iplPreMultiplyAlpha (IplImage* image, int alphaValue);
```

Filtering Functions

Blur applies a simple neighborhood averaging filter to blur the image
```
void iplBlur(IplImage* srcImage, IplImage* dstImage, int nRows, int nCols, int anchorX, int anchorY,
IplCoord* map);
```
Convolve2D convolves an IPL image with one or more convolution kernels
```
void iplConvolve2D(IplImage* srcImage, IplImage* dstImage, IplConvKernel** kernel, int nKernels, int
combineMethod, IplCoord* map);
```
ConvolveSep2D convolves an IPL image with a separable convolution kernels
```
void iplConvolveSep2D(IplImgreg* srcImage, IplImage* dstImage, IplConvKernel* xKernel, IplConvKernel*
yKernel, IplCoord* map);
```
CreateConvKernel creates a convolution kernel
```
IplConvKernel* iplCreateConvKernel(int nRows, int nCols, int anchorX, int anchorY, char* values, int
nShiftR);
```
DeleteConvKernel deletes the convolution kernel
```
void iplDeleteConvKernel(IplConvKernel* kernel);
```

Filtering Functions (continued)

GetConvKernel reads the attributes of the convolution kernel
```
void iplGetConvKernel(IplConvKernel* kernel, int* nRows, int* nCols, int* anchorX, int* anchorY, char**
values, int* nShiftR);
```
MaxFilter applies a maximum filter to an IPL image
```
void iplMaxFilter(IplImage* srcImage, IplImage* dstImage, int nRows, int nCols, int anchorX, int
anchorY, IplCoord* map);
```
MedianFilter applies a median filter to an IPL image
```
void iplMedianFilter(IplImage* srcImage, IplImage* dstImage, int nRows, int nCols, int anchorX, int
anchorY, IplCoord* map);
```
MinFilter applies a minimum filter to an IPL image
```
void iplMinFilter(IplImage* srcImage, IplImage* dstImage, int nRows, int nCols, int anchorX, int
anchorY, IplCoord* map);
```

Fast Fourier and Discrete Cosine Transform Functions

CcsFft2D computes the forward or inverse 2D complex fast Fourier transform of an image
```
void iplCcsFft2D(IplImage* srcImage, IplImage* dstImage, int flags, IplCoord* map);
```
DCT2D computes the forward or inverse 2D discrete cosine transform of an image
```
void iplDCT2D(IplImage* srcImage, IplImage* dstImage, int flags, IplCoord* map);
```
RealFft2D computes the forward or inverse 2D real fast Fourier transform of an image
```
void iplRealFft2D(IplImage* srcImage, IplImage* dstImage, int flags, IplCoord* map);
```

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

Histogram and Thresholding Functions

ComputeHisto computes the image intentsity histogram
```
void iplComputeHisto(IplImage* srcImage, IplLUT** lut, IplCoord* map);
```
ContrastStretch stretches the constrast of an image using an intensity transformation
```
void iplContrastStretch(IplImage* srcImage, IplImage* dstImage, IplLUT** lut, IplCoord* map);
```
HistoEqualize equalizes the image intensity histogram
```
void iplHistoEqualize(IplImage* srcImage, IPLImage* dstImage, IplLUT** lut, IplCoord* map);
```
Threshold performs a simple thresholding of an image
```
void iplThreshold(IplImage* srcImage, IplImage* dstImage, int threshold, IplCoord* map);
```

Geometric Transformation Functions

Decimate shrinks (decimates) the image
```
void iplDecimate(IplImage* srcImage, IplImage* dstImage, int xDst, int xSrc, int yDst,
int ySrc, int interpolate, IplCoord* map);
```
Mirror finds a mirror image
```
void iplMirror(IplImage* srcImage, IplImage* dstImage, int flipAxis, IplCoord* map);
```
Rotate rotates the image
```
void iplRotate(IplImage* srcImage, IplImage* dstImage, int angle, int centerX,
int centerY, int interpolate, IplCoord* map);
```
Zoom magnifies (zooms) the image
```
void iplZoom(IplImage* srcImage, IplImage* dstImage, int xDst, int xSrc, int yDst,
int ySrc, int interpolate, IplCoord* map);
```

How Do I…

add a constant to pixel values, *see* AddS *in* **Arithmetic 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 *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**
apply a color twist matrix, *see* ApplyColorTwist *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**
change the image orientation, *see* Rotate, Mirror *in* **Geometric Transformation Functions**
change the image size, *see* Zoom, Decimate *in* **Geometric Transformation Functions**
composite images using the alpha channel, *see* AlphaComposite, AlphaCompositeC *in* **Alpha-Blending 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 complex fast Fourier transform, *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 real fast Fourier transform, *see* RealFft2D *in* **Fast Fourier and Discrete Cosine Transform Functions**
compute the image histogram, *see* ComputeHisto *in* **Histogram and Thresholding Functions**
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 an HLS image to RGB, *see* HLS2RGB *in* **Conversion and Data Exchange Functions**
convert an HSV image to RGB, *see* HSV2RGB *in* **Conversion and Data Exchange Functions**
convert an RGB image to HLS, *see* RGB2HLS *in* **Conversion and Data Exchange Functions**
convert an RGB image to HSV, *see* RGB2HSV *in* **Conversion and Data Exchange Functions**
convert images from DIB (changing attributes), *see* ConvertFromDIB *in* **Windows DIB Conversion Functions**
convert images from DIB (preserving attributes), *see* TranslateDIB *in* **Windows DIB Conversion Functions**
convert images to DIB, *see* ConvertToDIB *in* **Windows DIB Conversion Functions**
convert one IPL image to another, *see* Convert *in* **Conversion and Data Exchange Functions**
convolve an image with 2D kernel, *see* Convolve2D *in* **Filtering Functions**
convolve an image with a separable 2D kernel, *see* ConvolveSep2D *in* **Filtering Functions**
copy image data, *see* Copy *in* **Conversion and Data Exchange Functions**
create 2D convolution kernel, *see* CreateConvKernel *in* **Filtering Functions**
create a color twist matrix, *see* CreateColorTwist *in* **Conversion and Data Exchange Functions**
create a region of interest (ROI), *see* CreateROI *in* **Image Creation Functions**
create image header, *see* CreateImageHeader *in* **Image Creation Functions**
create the IplTileInfo structure, *see* CreateTileInfo *in* **Image Creation Functions**
decimate the image, *see* Decimate *in* **Geometric Transformation Functions**
delete 2D convolution kernel, *see* DeleteConvKernel *in* **Filtering Functions**
delete a color twist matrix, *see* DeleteColorTwist *in* **Conversion and Data Exchange Functions**
delete the IplTileInfo structure, *see* DeleteTileInfo *in* **Image Creation Functions**
divide pixel values by 2^N, *see* RShiftS *in* **Logical Functions**
equalize the image histogram, *see* HistoEqualize *in* **Histogram and Thresholding Functions**
exchange data of two images, *see* Exchange *in* **Conversion and Data Exchange Functions**
fill image's pixels with a value, *see* Set *in* **Conversion and Data Exchange Functions**
free memory allocated by Malloc functions, *see* Free *in* **Memory Allocation Functions**
free the image data memory, *see* DeallocateImage *in* **Image Creation Functions**

How Do I… (continued)

free the image header memory, *see* Deallocate *in* **Image Creation Functions**
free the memory for image data or ROI, *see* Deallocate *in* **Image Creation Functions**
free the memory used for a color-twist matrix, *see* DeleteColorTwist *in* **Conversion and Data Exchange Functions**
get the error-handling mode, *see* GetErrMode *in* **Error-Handling Functions**
get the error status code, *see* GetErrStatus *in* **Error-Handling Functions**
handle an error, *see* Error *in* **Error-Handling Functions**
magnify the image, *see* Zoom *in* **Geometric Transformation Functions**
mirror the image, *see* Mirror *in* **Geometric Transformation Functions**
multiply pixel values by a constant, *see* MultiplyS *in* **Arithmetic Functions**
multiply pixel values by a constant and scale the products, *see* MultiplySScale *in* **Arithmetic Functions**
multiply pixel values of two images, *see* Multiply *in* **Arithmetic Functions**
multiply pixel values of two images and scale the products, *see* MultiplyScale *in* **Arithmetic Functions**
pre-multiply pixel values by alpha values, *see* PreMultiplyAlpha *in* **Alpha-Blending Functions**
produce error messages for users, *see* ErrorStr *in* **Error-Handling Functions**
read convolution kernel's attributes, *see* GetConvKernel *in* **Filtering Functions**
reduce the image bit resolution, *see* ReduceBits *in* **Conversion and Data Exchange Functions**
report an error, *see* Error *in* **Error-Handling Functions**
rotate the image, *see* Rotate *in* **Geometric Transformation Functions**
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 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 *in* **Arithmetic Functions**
subtract pixel values of two images, *see* Subtract *in* **Arithmetic Functions**
threshold the source image, *see* Threshold *in* **Histogram and Thresholding Functions**
zoom the image, *see* Zoom *in* **Geometric Transformation Functions**

Order number 673261-002