
IM Image Representation, Storage, Capture and Processing

Version 3.0

(im@tecgraf.puc-rio.br)

IM is a toolkit for Digital Imaging. IM is based on 4 concepts: Image Representation, Storage, Processing and Capture. The main goal of the library is to provide a simple API and abstraction of images for scientific applications.

The most popular file formats are supported: TIFF, BMP, PNG, JPEG, GIF and AVI. Image representation includes scientific data types. About a hundred Image Processing operations are available.

This work was developed at Tecgraf/PUC-Rio by means of the partnership with PETROBRAS/CENPES.

The **IM** Team:

Antonio Escaño Scuri

Tecgraf - Computer Graphics Technology Group, PUC-Rio, Brazil
<http://www.tecgraf.puc-rio.br/im>

Overview

IM is a toolkit for Digital Imaging. IM is based on 4 concepts: Image Representation, Storage, Processing and Capture. Image Visualization is a task that it is left for a graphics library.

It provides support for image capture, several image file formats and many image processing operations. The most popular file formats are supported: TIFF, BMP, PNG, JPEG, GIF and AVI.

Image representation includes scientific data types (like IEEE floating point data) and attributes (or metadata like GeoTIFF and Exif tags). Animation, video and volumes are supported as image sequences, but there is no digital audio support.

The main goal of the library is to provide a simple API and abstraction of images for scientific applications.

The toolkit API is written in C. The core library source code is implemented in C++ and it is very portable, it can be compiled in Windows and UNIX with no modifications. New image processing operations can be implemented in C or in C++.

IM is free software, can be used for public and commercial applications.

Availability

The library is available for several **compilers**:

- GCC and CC, in the UNIX environment
- Visual C++, Borland C++, Watcom C++ and GCC (Cygwin and MingW), in the Windows environment

The library is available for several **operating systems**:

- UNIX (SunOS, IRIX, AIX and Linux)
- Microsoft Windows NT/2K/XP

Support

The official support mechanism is by e-mail, using **im AT tecgraf.puc-rio.br** (replace " AT " by "@"). Before sending your message:

- Check if the reported behavior is not described in the user guide.
- Check if the reported behavior is not described in the specific format characteristics.
- Check the History to see if your version is updated.
- Check the To Do list to see if your problem has already been reported.

After all of the above have been checked, report the problem, including in your message: **function, element, format, platform, and compiler.**

Announcements of new versions are done by the read only list **im-l AT tecgraf.puc-rio.br** (replace " AT " by @). Send a request to the support e-mail to be added or removed from the list.

Credits

This work was developed at Tecgraf by means of the partnership with PETROBRAS/CENPES.

Thanks to the people that worked in the library:

- *Marcelo Gattass* and *Luiz Henrique Figueiredo*
- *Antonio Nabuco Tartarini*
- *Diego Fernandes Nehab*
- *Erick de Moura Ferreira*
- *Carolina Alfaro*

We also thank the developers of the third party libraries (also free) that we use:

- Sam Leffler (libTIFF author)
- Frank Warmerdam, Andrey Kiselev, Mike Welles and Dwight Kelly ([libTIFF](#) actual maintainers)
- Thomas Lane ([libJPEG](#))
- Lutz Müller ([libExif](#))

- Glenn Randers-Pehrson ([libPNG](#))
- Jean-loup Gailly and Mark Adler ([zlib](#))
- Gershon Elber (GIFLib)
- Michael Adams ([libJasper](#))
- Svein Bøe, Tor Lønnestad and Otto Milvang ([XITE](#))
- (to many others that contribute to these library, keeping them free and updated)

Documentation

This toolkit is available at <http://www.tecgraf.puc-rio.br/im>.

The full documentation can be downloaded from the [Download](#) by choosing the "Documentation Files" option.

The documentation is also available in Adobe Acrobat ([im.pdf](#) ~600Kb) and Windows HTML Help ([im.chm](#) ~400Kb) formats.

The HTML navigation uses the WebBook tool, available at <http://www.tecgraf.puc-rio.br/webbook>.

The library Reference documentation is generated by Doxygen (<http://www.stack.nl/~dimitri/doxygen/>).

Tecgraf Library License

This product is free software: it can be used for both academic and commercial purposes at absolutely no cost. There are no royalties or GNU-like "copyleft" restrictions. It is licensed under the terms of the [MIT license](#) reproduced below, and so is compatible with [GPL](#) and also qualifies as [Open Source](#) software. It is not in the public domain, Tecgraf and Petrobras keep its copyright. The legal details are below.

The spirit of this license is that you are free to use the library for any purpose at no cost without having to ask us. The only requirement is that if you do use it, then you should give us credit by including the copyright notice below somewhere in your product or its documentation. A nice, but optional, way to give us further credit is to include a Tecgraf logo in a web page for your product.

The library is designed and implemented by a team at Tecgraf/PUC-Rio in Brazil. The implementation is not derived from licensed software. The library was developed by request of Petrobras. Petrobras permits Tecgraf to distribute the library under the coditions here presented.

Copyright © 1994-2004 [Tecgraf / PUC-Rio](#) and [PETROBRAS S/A](#).

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.

History

Version 3.0.3 (14 Oct 2004)

- Improved C API of **imAttribTable**. New utility class **imAttribArray**.
- Fixed file read with bitmap conversion when original data changes only data type.
- Improved **imProcessThreshold**, **imProcessRangeContrastThreshold** and **imProcessLocalMaxThreshold** now also supports **IM_USHORT** and **IM_INT** data types.
- Rank convolution operations did not accept even kernel sizes.
- New group of functions Image Analysis: **imAnalyzeFindRegions**, **imAnalyzeMeasureArea**, **imAnalyzeMeasurePerimArea**, **imAnalyzeMeasureCentroid**, **imAnalyzeMeasurePrincipalAxis**, **imAnalyzeMeasureHoles**, **imProcessPerimeterLine**, **imAnalyzeMeasurePerimeter**, **imProcessPrune**, **imProcessFillHoles**.
- New Image Transform **imProcessDistanceTransform**.
- The default color conversion to binary now can be done for all color spaces.
- Fixed bug in **imProcessHoughLinesDraw** that was ignoring some lines.
- New **imConvertMapToRGB** to help loading data as RGB.
- New sample **iupglcap**.
- **im_process.h** was split into 4 files: **im_process_pont.h**, **im_process_loc.h**, **im_process_glo.h**, **im_process_ana.h**. But it still exists and includes the new files for compatibility.
- New **imProcessRenderChessboard** and **imProcessRenderGrid**.
- Improved the border extensions in several types of convolution. Rank convolution do not extend the borders. Binary morphology use zero extension. Gray morphology do not extend the borders.

Version 3.0.2 (25 Aug 2004)

- New utility functions **imPaletteHighContrast**, **imImageLoadImage** and **imImageLoadBitmap**.
- New operation **imProcessNormalizeComponents**.
- Changed name **imProcessGaussianConvolve** to **imProcessGaussianConvolveRep**. New operation **imProcessGaussianConvolve** that uses a float kernel. New utility functions

imGaussianStdDev2Repetitions and **imGaussianStdDev2KernelSize**.

- Changed name **imProcessDiffOfGaussianConvolve** to **imProcessDiffOfGaussianConvolveRep**. New operation **imProcessDiffOfGaussianConvolve** that uses a float kernel.
- Changed **IM_GAMUT_BRIGHTCONT** parameters to the interval [-100,100]. Fixed **IM_GAMUT_EXPAND** and **IM_GAMUT_BRIGHTCONT** normalization.
- Removed logical operation flag **IM_BIT_NOT**. Replaced by operation **imProcessBitwiseNot**.
- Improved in **imImageSetAttribute** count can be -1 for zero terminated data.
- Fixed operations **imProcessBitwiseNot** and **imProcessNegative** for **IM_BINARY** images.
- Fixed bug in the **color_mode_flags** parameter interpretation by **imFileReadImageData**.
- Fixed bug in **imProcessEqualizeHistogram** and **imProcessExpandHistogram** for color images.
- Fixed bug in **imProcessMultipleStdDev**.
- Fixed bug in **imProcessDifusionErrThreshold** for **IM_GRAY** images.
- Fixed bug in "KRN" format, internal format is topdown.
- Fixed bug in initialization of TGA image_count.

Version 3.0.1 (22 Apr 2004)

- Improved compatibility with the old version, it was missing the load of Map images with **imLoadRGB**.
- The FFTW code was from version 2.1.3, not from 2.1.5 as supposed, it was updated. The FFT functions were condensed in only one file with an "#ifdef" for FFTW version 2 and 3. The FFT functions also were renamed to remove the "W" that belongs only to the FFTW library.
- The **SetAttribute** functions now accept NULL in data to remove the attribute.
- New **imProcessCrossCorrelation** and **imProcessAutoCorrelation** functions.
- The **imCalcGrayHistogram** function now can calculate the histogram of **IM_MAP** and **IM_BINARY** images.

Version 3.0 (April 2004)

A major rewrite of the library. Everything changed, check the manual, but backward compatibility is kept for old applications. A new API more flexible, new formats, support for attributes and video, image capture and image processing. New color spaces and data types. The library now got a professional look for scientific applications.

Version 2.6 (May 2002)

Correction of bug in resolution reading and writing for format JPEG.

Version 2.5 (August 2001)

Correction of bug in the default GIF compression. Two new callbacks: transparency color index for GIF files and image description for TIFF files.

Version 2.4 (February 2000)

Change in the treatment of LZW compression in formats TIFF and GIF.
Now compression is no longer the default.

Version 2.3 (June 1998)

Close function of the access driver for files in memory corrected. JPEG library updated to 6b. Correction of a problem with the reading of some JPEG files.

Version 2.2 (November 1997)

The definition of the counter callback was changed to inform, in a parameter, the type of access being performed, either reading or writing. Type **imCallback** defined to make type casting easier when using function **imRegisterCallback**. Correction of a problem with the makefile in UNIX, which was generating link errors in some platforms.

Version 2.1 (October 1997)

Correction of a problem with internal memory liberation when reading Map images in TIFF files. Conversion **RGB to Map** is now made using the

algorithm implemented by LibJPEG. The algorithm of **imResize** was improved for cases in which the size is being reduced instead of increased. Correction of a problem with functions **imImageInfo** and **imFileFormat**: when the provided file was not in a format recognized by IM, there was an error in format TGA which caused these functions to access an invalid memory area.

Version 2.0 (September 1997)

The library was virtually rewritten to implement a new structure which allowed greater flexibility, simplifying the addition of new formats. Formats **TGA**, **PCL**, **JPEG** and **LED** were added to the list of supported formats, and new functions were added: **imMap2RGB**, **imRGB2Gray**, **imMap2Gray**, **imResize**, **imStretch**.

Version 1.1 (June 1996)

Small corrections to increase portability. Changes in return codes. Identifiers were created to return codes and predefined parameters. Online manual concluded.

Version 1.0 (October 1995)

To Do

For the next versions

- Binding for Lua 5
- Linux Capture (using Video4Linux)
- Use libavcodec and libavformat in Linux
- AVI using libavifile in Linux (UNIX ?)
- MPEG-2 (using MSSG?)
- MOV (using SDK and QT4Linux)
- DICOM
- TIFF Annotations
- TIFF EXIF tags

For the Processing library:

- Dithering Techniques
- Adaptative Thresholds
- Warping
- Rolling Ball Filter
- A free FFT implementation
- Butterworth, Deconvolution
- Inverse Filter, Homomorphic Restoration
- Watershed, Convex Hull
- Other Measures

Our plans for the future include:

- **Imaging Tutorial in the documentation**
- JPEG and TIFF Thumbnails
- Formats: FLI, DV, FPX (Flash Pix), EXR (Industrial Light & Magic High Dynamic Range Format), MNG
- Other scientific formats. FITS, VICAR, SEGY

Comparing IM with Other Imaging Toolkits

Still today there is a need for something easier to code and understand in Imaging. The available free libraries are sometimes close, sometimes very far from easier. IM is an unexplored solution and proposed as a simple and clean one. It is another Imaging tool with a different approach to the many possibilities in the area. Its organization was designed so it can be used for teaching Imaging concepts. We invite you to try it.

First we list some libraries mainly target for storage, then some scientific libraries, and then a small comparsion of IM and those libraries.

Here are some free storage libraries:

Imlib2

Last Update 2003-09 / Version 1.1.0

<http://www.enlightenment.org/pages/imlib2.html>

Language C

Documentation is terrible. Depends on the X-Windows System libraries.

It is designed for display/rendering performance.

Corona

Last Update 2003-09 / Version 1.0.2

<http://corona.sourceforge.net/>

Language C++

Very simple library. Only a few formats. Only bitmap images, no video.

PaintLib

Last Update 2004-04 / Version 2.61

<http://www.paintlib.de/paintlib/>

Language C++

A very simple library.

Has an interesting ActiveX component. Only bitmap images, no video.

NetPBM

Last Update 2004-07 / Version 10.23

<http://netpbm.sourceforge.net/>

Language C

A traditional library that starts at the Pbmplus package more than 10 years ago.

Very stable, it has support for the PNM format family and many processing operations.

Only bitmap images, no video.

DevIL ***

Last Update 2004-06 / Version 1.6.7

<http://openil.sourceforge.net/>

Language C (Has also a C++ Wrapper)

Called initially OpenIL. Supports many formats and have a very interesting API, that works very similar the OpenGL API (that's why the original name). Also supports the display in several graphics systems. Has several data types as OpenGL has.

FreeImage ***

Last Update 2004-07 / Version 3.4.0

<http://freeimage.sourceforge.net/>

Language C (Has also a C++ Wrapper)

Supports many formats. Many data types, but only RGB and subclasses (gray, map, etc).

Very well written, stable and simple to use.

ImageMagick and GraphicsMagick ***

Last Update 2004-07 / Version 6.0.3 || Last Update 2004-04 / Version 1.0.6

<http://www.imagemagick.org/> || <http://www.graphicsmagick.org/>

Language C (Has also a C++ Wrapper)

The two libraries are listed together because GraphicsMagick is totally

and explicitly based on ImageMagick version 5.5.2. They have very similar or identical APIs but the development process is completely different. GraphicsMagick propose a more organized development process (a more precise comparison requires detailed knowledge about the two libraries). These are very complete libraries. They support lots of file formats, several color spaces, but use only the byte data type. They use a big image structure with everything inside. Image creation may involve about 40 parameters.

And here are some free scientific libraries:

TINA

Last Update 2002-03 / Version 4.0.2

<http://www.niac.man.ac.uk/Tina>

Language C

Very UNIX oriented. Lots of functions for Computer Vision.

Developed by a researcher of the University of Manchester.

XITE

Last Update 2002-09 / Version 3.44

<http://www.ifi.uio.no/forskning/grupper/dsb/Software/Xite/>

Language C

Very UNIX oriented, but compiles fine in Windows. Several separated command line routines, it is a package not a library. But inspired several aspects of the IM library. Seems to be not updated anymore.

Developed by a researcher of the University of Oslo.

VIGRA

Last Update 2004-09 / Version 1.3.0

<http://kogs-www.informatik.uni-hamburg.de/~koethe/vigra/>

Language C++

STL based. Many operators. Developed by a researcher of the University of Hamburg.

Wild Magic

Last Update 2004-09 / Version 2.4

<http://www.magic-software.com/>

Language C++

Game development oriented, very rich in mathematics. Developed by Magic Software, Inc.

VIPS

Last Update 2004-09 / Version 7.10.2

<http://www.vips.ecs.soton.ac.uk/>

Language C/C++

Support for very large images. Powerful macro language. Good implementation. Many functions. Developed by researchers at the University of Southampton and The National Gallery in the UK.

MegaWave2

Last Update 2004-06 / Version 2.3

<http://www.cmla.ens-cachan.fr/Cmla/Megawave/>

Language C

Very UNIX oriented. Good implementation. Many functions. C preprocessor. Developed by French researchers at l'École Normale Supérieure de Cachan.

JAI

Last Update 2003-07 / Version 1.1.2

<http://java.sun.com/products/java-media/jai/index.jsp>

Language Java

It is becoming more and more popular. Java is slow than C/C++ but the performance of the image processing operations is very acceptable. Also it has several C optimized functions. Developed by the Sun Corporation.

OpenCV ***

Last Update 2004-08 / Version 4.0

<http://sourceforge.net/projects/opencvlibrary/>

Language C/C++

Only a few formats but lots of image processing operations. One of the

most interesting libraries available. It is more than an Imaging library, it is designed for Computer Vision. Developed by Intel Russian researchers.

VTK ***

Last Update 2004-03 / Version 4.2

<http://www.vtk.org/>

Language C++

Another very important library. Very huge. Much more than Imaging, includes also 3D Computer Graphics and Visualization. Has a book about the library. Developed by Kitware Inc.

IM

Last Update 2004-08 / Version 3.0.2

<http://www.tecgraf.puc-rio.br/im>

Language C/C++

Support for several data types, i.e. scientific images and different color spaces. Support for input and output of image sequences. Support for generic image attributes (metadata), which includes several standard TIFF tags, GeoTIFF tags and Exif tags. Image storage and capture data can be accessed using an image structure or with raw data. Internal implementation in C++ but with a simple C API. Code is portable for Windows and UNIX. Many image processing operations.

Comparsion

The idea behind IM was to create a toolkit that was not so complex as OpenCV, neither so big as VTK, but that can be used as a solid base to the development of thesis and dissertations, as for commercial applications.

As the academic environment is very heterogeneous the IM project choose some directives:

- Portability (Windows and UNIX)

- C API
- Totally Free
- Focus in Scientific Applications
- Easy to Learn
- Easy to Reuse

Considering these directives there are only a few similar toolkits. Making some exceptions the following should be mentioned:

- JAI - Java, Sun.com
- VIGRA - C++ / STL Based, University
- VIPS - Large Images / Macros, University
- VTK - C++ / Huge / Visualization, Kitware.com
- OpenCV best similar choice, Intel.com

Today OpenCV and VTK are the most professional and complete choices of free libraries that are similar to IM. But they are more complicated than IM. For instance VTK it is very large, it has about 700 C++ classes.

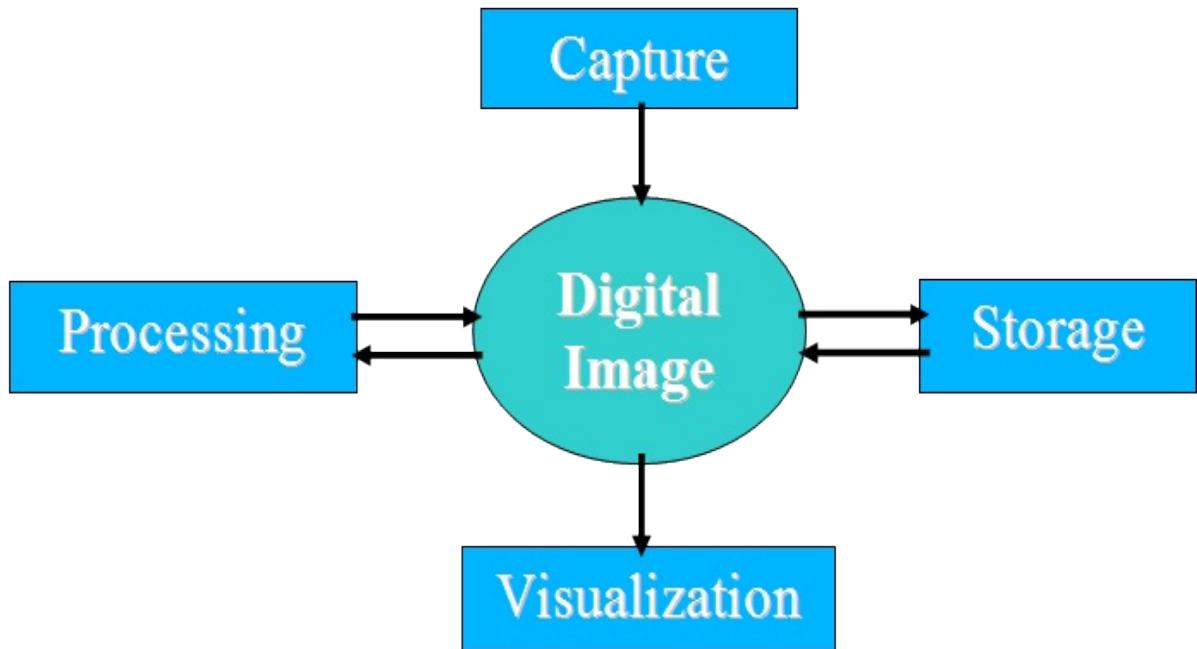
Although OpenCV has many resources, its code is very hard to reuse. The simplicity of the IM code, mainly the image processing routines, make it a good reference to be reused by other applications extracting only the code needed with little changes. And can be used as a complement to learn image processing algorithms and techniques.

This page was last updated in Sep 2004.

Guide

Getting Started

It is important to understand that IM is based in 4 concepts: **Image Representation**, **Image Storage**, **Image Processing** and **Image Capture**. The following picture illustrates the relation between these concepts.



IM does not have support for **Image Visualization**, because we think this is a task for a graphics library like OpenGL, Windows GDI or CD.

Image Representation describes the image model and its details. Which color systems are going to be used, which data types, how the data is organized in memory, and how other image characteristics are accessed.

Image Storage describes the file format model and how images are obtained or saved. **Image Capture** describes the access to a capture device and obtaining an image from it. **Image Processing** describes the image processing operations.

There are infinite ways to implement these concepts. There is no common definition in the literature, but there is a standard called Programmer's

Imaging Kernel System (PIKS) published at the ISO/IEC 12087. PIKS is a very complete and also complex standard, very hard to implement. There are only a few implementations available, and the one that I know is commercial software, Pixel Soft of William Pratt
<http://www.pixelsoft.com/>, also author of several books on the subject.

But we want something easier to implement and understand. The free available libraries that we found were sometimes close to what we want, sometimes very far. So we developed our own.

The documentation is divided into two major parts: **Guide** and **Reference**.

The **Guide** is where you are going to find the explanation about the concepts and decisions made during the library design. It is the best place to understand how things work.

The **Reference** contains pure essential information for function and structure usage. But there is no information on how to put the functions to work together. It is generated automatically from the source code using Doxygen, this means also that the include files (*.h) are very well commented.

You should read at least the [Guide / Basics](#) section. There are all the important concepts developed and used in the library, as many samples on how to combine the available functions to obtain the desired results.

Building Applications

Inside your code you should at least include the <im.h> file and link with the "im.a/im.lib" file. This includes all the **Image Representation** functions and all the **Image Storage** functions (with the exception of the external SDK formats: AVI, JP2 and WMV).

For **imImage** structure you should include the <im_image.h> file. There are several other utilities that need different header files, check the Reference documentation.

For the **Image Capture** you should include the <im_capture.h> file and link with the "im_capture.lib" file. The same for **Image Processing**,

<im_process.h> file and "im_process.a/im_process.lib" file.

Each external format or processing usually needs a <im_xx.h> file and a "im_xx.a/im_xx.lib" file.

The WMV format is available only for the Visual C++ compiler, you will also need the file "[wmvcore.lib](#)". But when using the DLL you may try to generate an import library for other compilers. To compile the format source code you will need the Windows Media Format SDK.

The AVI format is available for gcc 3 and Mingw 3, but to link your application you will need the files "[libvfw_ms32.a](#) and [libvfw_avi32.a](#)".

To link with the capture library in Windows using Visual C you will need the file "[strmiids.lib](#)". To compile the capture source code you will need the Direct X 9 SDK. To link it using Dev-C++ or Mingw 3 you will need the "[im_capture.dll](#)".

Even if your application is only in C, you must link with a C++ capable linker. Using Tecmake set "LINKER := g++" in your "config.mak" when compiling with gcc (UNIX and Windows).

Building the Library

The easiest way to build the library is to install the Tecmake tool into your system. It is easy and helps a lot. The Tecmake configuration files (*.mak) available at the "src" folder are very easy to understand also. But we also provide a makefile for Linux systems and a Visual Studio workspace with the respective projects.

Tecmake is a command line multi compiler build tool available at <http://www.tecgraf.puc-rio.br/tecmake>. Tecmake is used by all the Tecgraf libraries and many applications.

About File Formats

TIFF is still the most complete format available. It could be better if Adobe releases the revision 7, but it is on stand by. TIFF supports all the IM image representation concepts. In fact we were partially inspired by the TIFF

specification. My suggestion is whenever possible use TIFF.

But TIFF may not be the ideal format for many situations. The W3C standards include only JPEG, GIF and PNG for Web browsers. JPEG forces the image to be RGB or Gray with a lossy compressed. GIF forces the image to be MAP with LZW compression. PNG forces the image to be RGB, MAP, Gray or Binary, with Deflate compression. So these characteristics are necessary to force small values for faster downloads.

JPEG is to be used for photographic content, PNG should be used for the remaining cases, but GIF is still the best to do simple animated images.

Except for some specific cases where a format is needed for compatibility, the other formats are less important. TGA, PCX, RAS, SGI and BMP have almost the same utility.

JP2 must be used for JPEG-2000 compression, would be nice if a new TIFF specification includes this standard.

Since PNM has a textual header it is very simple to teach for students so they can actually "see" the header. It is also a format easy to share images, but it does not do much more than that.

The TIFF and the GIF format also have support for multiple images. This does not necessarily defines an animation, pyramid nor a volume, but sometimes they are used in these ways.

GIF became very popular to build animations for the Web, and since the LZW patent expired Unisys realized that charging the usage isn't going to work and so they did not renew it. LZW is fully supported at IM.

IM also supports video formats like AVI and WMV as external libraries. In these cases the frames are also loaded as a sequence of individual images. Sound is not supported.

TIFF, JPEG and PNG have an extensive list of attributes, most of them are listed in the documentation, but some custom attributes may come up when reading an image from file.

CD Compatibility

IM version 2 was designed to perfectly work with the [CD - Canvas Draw](#) toolkit. Version 3 has many more options and only for a subset of the images called Bitmaps can be used with the CD functions. These images have data type `IM_BYTE`, and color mode `IM_RGB`, `IM_GRAY`, `IM_MAP` or `IM_BINARY`. They can not have the flags `IM_TOPDOWN` and `IM_PACKED`. But it can have the flag `IM_ALPHA` for `IM_RGB` images.

You can convert an image to a bitmap version of it using the function `imConvertToBitmap`, see [Reference / Image Representation / Conversion](#).

Function `cdGetImageRGB` captures an image from the active canvas. Functions `cdPutImageRGB` and `cdPutImageMap` place an RGB image or an indexed image, respectively, on the active canvas. These functions allow reducing or increasing the image when placing it on the canvas.

For applications in systems with only 256 colors available, we recommend the use of function `cdPalette` before drawing the image, to improve its quality.

When using the `imImage` structure the macro `cdPutBitmap` can be used. It is defined as:

```
#define cdPutBitmap(_image, _x, _y, _w, _h, _xmin, _xmax, _ymin,  
{  
    if (_image->color_space == IM_RGB)  
        cdPutImageRectRGB(_image->width, _image->height,  
                           (unsigned char*)_image->data[0],  
                           (unsigned char*)_image->data[1],  
                           (unsigned char*)_image->data[2],  
                           _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax  
    else  
        cdPutImageRectMap(_image->width, _image->height,  
                           (unsigned char*)_image->data[0], _image->  
                           _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax  
}
```

CD Library is the Tecgraf 2D graphics library available at <http://www.tecgraf.puc-rio.br/cd>.

OpenGL Compatibility

The function `glDrawPixels` accepts several data types and color modes.

Here are the **format** and **type** mapping for OpenGL usage:

IM	\leftrightarrow	OpenGL
color_mode		format
IM_RGB IM_ALPHA IM_PACKED	=	GL_RGBA
IM_RGB IM_PACKED	=	GL_RGB
IM_GRAY	=	GL_LUMINANCE
IM_GRAY IM_ALPHA IM_PACKED	=	GL_LUMINANCE_ALPHA
data_type		type
IM_BYTE	=	GL_UNSIGNED_BYTE
IM_BINARY	=	GL_BITMAP
IM USHORT	=	GL_UNSIGNED_SHORT
IM_INT	=	GL_INT
IM_FLOAT	=	GL_FLOAT

There is no mapping for non IM_PACKED images so if you use unpacked planes (ex: you use the `imImage` structure) then you have to convert one data into another, the function `imConvertPacking` does this, so you just have to keep an extra buffer for the display image and call this function only when your original image has changed. See [Reference / Image Representation / Conversion](#). For example:

```
imConvertPacking(image->data[0], gl_data, image->width, image-> height,
glPixelStorei(GL_UNPACK_ALIGNMENT, 1); /* data alignment must be
glDrawPixels(image->width, image->height, GL_RGB, GL_UNSIGNED_BY
```

When loading color image data you can use the function `imConvertMapToRGBPacked` to convert in-place IM_MAP image data into IM_RGB after loading it from file. For example:

```
if (imColorSpace(color_mode) == IM_MAP)
{
    long palette[256];
    int palette_count;
    imFileGetPalette(ifile, palette, &palette_count);
    imConvertMapToRGBPacked(gl_data, width*height, depth, palette,
```

```
}
```

IM 2.x Compatibility

In version 3.0 the library was completely rewritten. And we changed the main API to allow more powerful features. But the old API is still available

for backward compatibility. Version 3 is also binary compatible with version 2.

The only change that must be updated in old applications if they were recompiled is some error code definitions. If you use them in a case there will cause a compiler error because `IM_ERR_READ` and `IM_ERR_WRITE` are now defined as `IM_ERR_ACCESS` both.

Migrating OLD Code

The old API is very inefficient because the file is opened and closed three times, for: `imFileInfo`, `imImageInfo` and `imLoadRGB/imLoadMap`. There is no room for attributes, so we use the callbacks. And we can not load sequences of images. For these reasons we change the API.

If you would like to migrate your code using the old API the most important thing to change is the memory allocation. For RGB images instead of allocating 3 separate pointers you should allocate only one pointer with room for all three planes. If you still want to keep the three pointers, just do `green = red + width*height` and `blue = red + 2*width*height`.

Also you should change your callbacks usage for attributes access using `imFileGetAttribute` and `imFileSetAttribute`. `IM_RESOLUTION_CB` is replaced by the attributes "XResolution", "YResolution", "ResolutionUnit". `IM_GIF_TRANSPARENT_COLOR_CB` is replaced by "TransparencyIndex" and `IM_TIF_IMAGE_DESCRIPTION_CB` by "Description".

Except `IM_COUNTER_CB` that is not an attribute, still works with a callback, but now we implement a counter system for all the library including loading, saving and processing. The user just use the `imCounterSetCallback` (like before) to register it counter callback, now there are a few more parameters and a user data pointer. See [Reference / Utilities / Counter](#).

The function calls to `imImageInfo` and `imLoadRGB/imLoadMap` will be replaced by a sequence of function calls to `imFileOpen/imFileNew`, `imFileReadImageInfo/imFilewriteImageInfo`,

`imFileReadImageData`/`imFileWriteImageData` and `imFileClose`. See [Reference / Image Storage](#).

Complete Samples

im_info

This is a command line application that displays information obtained from a file using the IM I/O functions, basically **imFile** functions. It depends only on the IM main library.

Here is an output sample:

```
IM Info
  File Name:
    exif_test.tif
  File Size: 9.00 Mb
  Format: TIFF - Tagged Image File Format
  Compression: NONE
  Image Count: 1
  Image #0
    Width: 2048
    Height: 1536
    Color Space: RGB
      Has Alpha: No
      Is Packed: Yes
      Is Top Down: Yes
    Data Type: byte
    Data Size: 9.00 Mb
    Attributes:
      YResolution: 72.00
      XResolution: 72.00
      DateTime: 2004:01:14 11:30:11
      Make: SONY
      ResolutionUnit: DPI
      Model: CD MAVICA
      Photometric: 2
```

You can view the source code here: [im_info.cpp](#)

im_copy

This is a command line application that copies all the information from one file to another using the IM I/O functions. It depends only on the IM main library. It is usefull for testing the drivers.

You can view the source code here: [im_copy.cpp](#)

proc_fourier

This is another command line application that process an image in the Fourier Frequency Domain. In this domain the image is a map of the spatial frequencies of the original image. It depends on the IM main library and on the IM_FFTW library. The FFTW is a very fast Fourier transform, but is contaminated by the GPL license, so everything must be also GPL. To use it in a commercial application you must contact the MIT and pay for a commercial license.

Se also [Reference / Image Processing / Domain Transform Operations](#).

You can view the source code here: [proc_fourier.cpp](#)

im_view

This application uses IUP and CD to create a window with a canvas and draw the image into that canvas. It is a very simple application, no zoom nor scrollbar management. The image is obtained from a file using the IM I/O functions, but using the **imImage** structure to make the implementation easier.

For more IUP <http://www.tecgraf.puc-rio.br/iup> and more CD <http://www.tecgraf.puc-rio.br/cd>

You can view the source code here: [im_view.c](#)

glut_capture

This application uses GLUT and OpenGL to create a window with a canvas and draw the image into that canvas. But the image is obtained from a capture device. The image can be processed before display and a sequence of captured images can be saved in an AVI file during capture.

You can view the source code here: [glut_capture.c](#)

iupglcap

This application uses IUP and OpenGL to create a window with two canvases and draw a video capture image into one canvas. A processed image can be displayed in the second canvas. It can also process frames from a video file.

You can download the source code and some compiler projects here:
[iupglcap.zip](#)

IMLAB

If you want to see a more complex application with all the IM features explored the IMLAB is a complete example. It displays each image in an individual image with zoom and pan capabilities. All the IM processing operations are available together with some extra operations.

For more IMLAB go to <http://www.tecgraf.puc-rio.br/~scuri/imlab>.

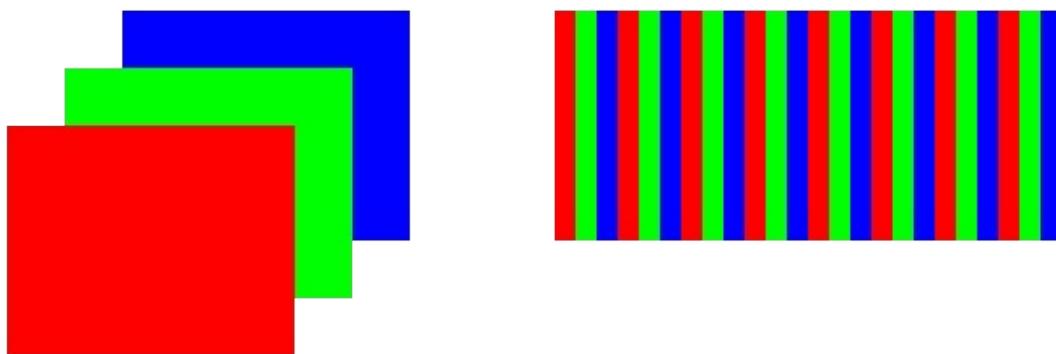
Architecture Guide

Image Representation (Data Model)

In the IM library images are 2D matrices of pixels defining **width** and **height**. Stacks, Animations, Videos and Volumes are represented as a sequence of individual images.

The pixels can have one of several **color spaces**: `IM_RGB`, `IM_MAP`, `IM_GRAY`, `IM_BINARY`, `IM_CMYK`, `IM_YCBCR`, `IM_LAB`, `IM_LUV`, `IM_XYZ`. `IM_MAP` is a subset of the `IM_RGB` color space. `IM_MAP` can have a max of 256 colors. `IM_BINARY` a subset of the `IM_GRAY` color space, and it has only 2 colors black and white. `IM_GRAY` usually means luma (nonlinear Luminance), but it can represent any other intensity value that is not necessarily related to color.

The number of components of the color space defines the **depth** of the image. The color components can be packed sequentially in one plane (like `rgbrgbrrgb...`) or separated in several planes (like `rrr...ggg...bbb...`). Packed color components are normally used by graphics systems. We allow these two options because many user define their own image structure that can have a packed or an separated organization. The following picture illustrates the difference between the two options:



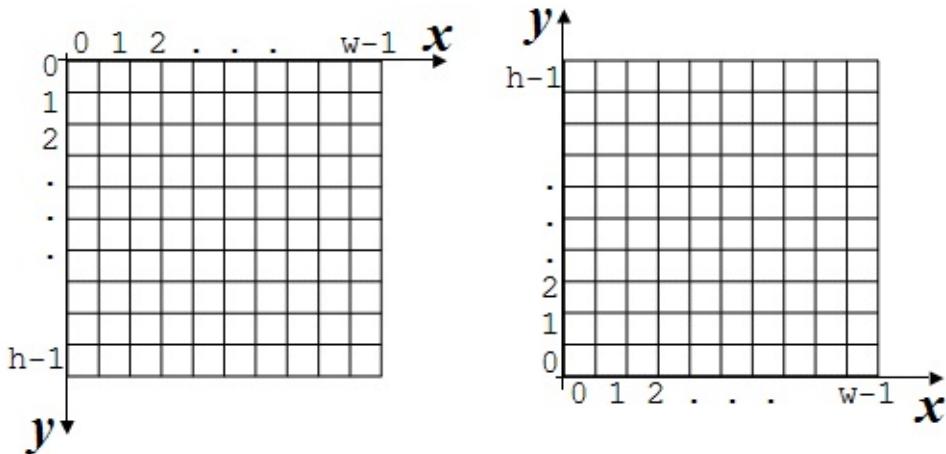
Separated and Packed RGB Components

An extra component, the **alpha** channel, may be present. The number of

components is then increased by one. Its organization follows the rules of packed and unpacked components.

There are several numeric representations for the color component, or several **data types**: `IM_BYTE`, `IM USHORT`, `IM_INT`, `IM_FLOAT`, `IM_CFLOAT`. There is no bit type, binary images use 1 byte (waist space but keep processing simple).

Image orientation can be bottom up to top with the origin at the bottom left corner, or top down to bottom with the origin at the top left corner.



Top Down and Bottom Up Orientations

Since all these options are relative to data organization we created a parameter called **color mode** that contains the color space, the packing, the orientation and the optional alpha channel. The color space definitions are binary "or" combined with the flags: `IM_ALPHA`, `IM_PACKED` and `IM_TOPDOWN`. When a flag is absent the opposite definition is assumed: no alpha, separated components and bottom up orientation. See some examples:

`IM_RGB | IM_ALPHA` - rgb color space with an alpha channel, bottom up orientation and separated components

`IM_GRAY | IM_TOPDOWN` - gray color space with no alpha channel and top down orientation

`IM_RGB | IM_ALPHA | IM_PACKED` - rgb color space with an alpha

channel, bottom up orientation and packed components

So these four parameters define our raw image data: **width**, **height**, **color_mode** and **data_type**. The raw data buffer is always byte aligned and each component is stored sequentially in the buffer with the specified packing.

If the raw data buffer is typecasted to the proper C type, then locating the pixel at line y, column x, component d is done like this:

```
if (is_packed) idata[y*width*depth + x*depth + d]
else           idata[d*width*height + y*width + x]
```

But this will return different pixel locations for top down and bottom up orientations.

We could restrict the data organization by eliminating the extra flags, but several users requested these features in the library. So we keep them but restricted to raw data access. For the high level image processing functions we created a structure called **imImage** that eliminates the extra flags and assume no alpha channel, bottom up orientation and separated components.

The **imImage** structure is created using the four image parameters: **width**, **height**, **color_space** and **data_type**. It is an open structure in C where you can access all the parameters. In addition to the 4 creation parameters there are many auxiliary parameters like **depth**, **count**, **line_size**, **plane_size** and **size**, with values calculated in the creation.

The data is allocated like the raw image data with separated color components one after another, but we access the data through an array of pointers each one starting at the beginning of each color component. So **data[0]** contains a pointer to all the data, and **data[1]** is a short cut to the second component and so on.

To the structure contains all the image information obtained from file it also has support for attributes and for the palette. The palette can be used for **IM_MAP** images and for pseudo color of **IM_GRAY** images.

An important subset of images is what we call a bitmap image. It is an image that can be directly used to graphics display. Color space must be:

`IM_RGB`, `IM_MAP`, `IM_GRAY` or `IM_BINARY`, and data type must be `IM_BYTE`.

The conversion between image data types, color modes and to bitmap are defined only for the `imImage` structure.

See: [Reference / Image Representation](#),
[Reference / Image Representation / Structure](#),
[Reference / Image Representation / Conversion](#),
[Reference / Structures / imImage](#),
[Guide / Basics / Creating](#).

Image Storage (File Format Model)

Essentially all the file formats save the same image data. There is no such thing like a GIF image, instead we have a color indexed image that can be saved in a file with a GIF format, or a TIFF format, etc. However the compression encoding can be lossy and degrade the original image. The point is file formats and image data are two different things.

A file format is a file organization of the image data and its attributes. The IM library model considers all the file formats under the same model, including image, video, animation, stacks and volume file formats. When there is more than one image each one is treated as an independent frame. Each frame can have its own parameters and set of attributes.

We consider only formats that starts with a signature so we can recognize the format without using its file extension. If there is more than one driver that handles the same signature the first registered driver will open the file. Since the internal drivers are automatically registered all the external drivers can be loaded first if no `imFile` function has been called. In this way you can also control which external driver goes first.

See: [Reference / Image Storage](#),
[Guide / Basics / Reading](#),
[Guide / Basics / Writing](#).

Image Capture (Live Image Input Model)

You must have a video capture device installed. It must be a device capable

of live video, it can not be a passive digital camera to only transfer the already taken pictures. Are valid: USB cameras (like most Webcams), Firewire (IEEE 1394) cameras, and analog video capture boards, including TV Tuners.

You can list the installed devices and once you connect to a specific device you can control its parameters. Each connected device captures data frames continuously when in Live state otherwise it stays in standby. Then the user should retrieve frames from the device. This can be done inside a closed loop or inside an idle function. The user is not notified when a new frame is available, but every time the user retrieve a frame if it is successful then it is a new frame, old frames are discarded when a new frame arrives.

See [Reference / Image Capture](#),
[Guide / Basics / Capturing](#).

Image Processing (Operation Model)

We use the simpliest model possible, a function with input data, output data and control parameters. There is no ROI (Region Of Interest) management.

The operations have usually one or more input images, and one or more output images. We avoid implementing in-place operations, but many operations can use the same data for input and output. The data type, color mode and size of the images depends on the operation. Sometimes the operations can change the data type to increase the precision of the results, but normally only a few operations will change the size (resize and geometric) and color mode (color conversion). All of these details are described in each function documentation, check before using them.

See [Reference / Image Processing](#),
[Guide / Basics / Processing](#).

Basics Guide

Creating

To create a raw image buffer you can simply use the utility function:

```
size = imImageContentSize(width, height, color_mode, data_type);  
void* data = malloc(size);
```

To create an **imImage** structure you can do it in several ways.

```
image = imImageCreate(width, height, color_space, data_type)  
image = imImageInit(width, height, color_space, data_type, data,  
image = imImageDuplicate(image)  
image = imImageClone(image)
```

The **imImageInit** function allow you to initialize an **imImage** structure with an user allocated buffer. It is very useful you use your own image structure and wants to temporally use the image processing functions of the library.

To destroy the **imImage** structure simply call **imImageDestroy(image)**. If you did "data[0] = NULL" before calling it the raw data buffer will not be destroyed.

See: [Guide / Architecture / Image Representation](#),
[Reference / Image Representation / Structure](#).

Reading

When reading the file extension is not relevant to determine the file format, but it is used to speed up the process of finding the correct format. With few exceptions the format drivers that access multiple images can read them in any sequence you want.

During the read process the original data can be converted to some options of user data. Not all conversions are available. You can convert any data to

a bitmap version of it, and you can select any of the color mode flags `IM_ALPHA`, `IM_PACKED` and `IM_TOPDOWN`, regardless of the file original configuration.

In the following example all the images in the file are read.

```
char format[10], compression[10];
int error, image_count;
int width, height, color_mode, data_type;
void* data;

imFile* ifile = imFileOpen("test.tif", &error);
if (error != IM_ERR_NONE)
    // handle the error

imFileGetInfo(ifile, format, compression, &image_count);

for (i = 0; i < image_count, i++)
{
    error = imFileReadImageInfo(ifile, i, &width, &height, &color_
    if (error != IM_ERR_NONE)
        // handle the error

    // prepare data

    error = imFileReadImageData(ifile, data, 0, -1); // no bitmap
    if (error != IM_ERR_NONE)
        // handle the error

    // store data somewhere
}

imFileClose(ifile);
```

A more simple code reads only the first image in the file:

```
imFile* ifile = imFileOpen(file_name, &error);

imFileReadImageInfo(ifile, 0, &width, &height, &color_mode, &dat
imFileReadImageData(ifile, data, 0, -1);

imFileClose(ifile);
```

If you are using the `imImage` structure it is even easier:

```
imFile* ifile = imFileOpen(file_name, &error);
imImage* image = imFileLoadImage(ifile, 0, &error);
// or use imFileLoadBitmap to force a bitmap conversion
imFileClose(ifile);
```

See: [Guide / Architecture / Image Storage](#),
[Reference / Image Storage](#),
[Reference / Image Storage / Read Access](#).

Writing

When writing there is no color space or data type conversion. Only color mode flags can be different: `IM_ALPHA`, `IM_PACKED` and `IM_TOPDOWN`. You just have to describe your data and the `imFileWriteImageData` will handle the color mode flag differences.

Of course you still have to check the error codes because, not all color spaces and data types are supported by each format.

When saving a sequence of images you must provide each image in the order that they will be in the file. For a video or animation start from frame 0 and go on, you can not jump or change the frame order. Also when saving videos you should not forget to save the numbers of frames per second in the attribute "FPS", the default value is 15.

For all the formats it is not necessary to set the compression, each driver will choose a default compression. But you may set it using the function `imFileSetInfo`.

To save several images to the same file:

```
int error, width, height;
void *data;

imFile* ifile = imFileNew("test.tif", "TIFF", &error);
if (error != IM_ERR_NONE)
    // handle the error

for (i = 0; i < image_count, i++)
```

```

{
    error = imFileWriteImageInfo(ifile, width, height, IM_RGB, IM_
        if (error != IM_ERR_NONE)
            // handle the error

    error = imFileWriteImageData(ifile, data);
    if (error != IM_ERR_NONE)
        // handle the error
}

imFileClose(ifile);

```

But remember that not all file formats supports several images. To save just one image is more simple:

```

imFile* ifile = imFileNew(file_name, format, &error);

error = imFileWriteImageInfo(ifile, width, height, color_mode, c
error = imFileWriteImageData(ifile, data);

imFileClose(ifile);

```

If you are using the `imImage` structure it is even easier:

```

imFile* ifile = imFileNew(file_name, format, &error);

error = imFileSaveImage(ifile, image);

imFileClose(ifile);

```

See: [Guide / Architecture / Image Storage](#),
[Reference / Image Storage](#),
[Reference / Image Storage / Write Access](#).

Capturing

You can list the installed capture devices using:

```

int imVideoCaptureDeviceCount(void)
const char* imVideoCaptureDeviceDesc(int device)

```

If a device was removed or added in run time, you must update the list calling:

```
int imVideoCaptureReloadDevices(void)
```

To handle devices you must create a `imVideoCapture` structure using the function `imVideoCaptureCreate`. With this handle you can manage any of the available devices, but only one device. The handle must be destroyed with `imVideoCaptureDestroy`.

If you want to access two or more devices at the same time you must create two different structures, but be aware that this usually work for high quality devices like Firewire and USB 2.0. Webcams that use USB1.x can be used if connected to different USB 2.0 controllers.

The next thing is to connect to a specific device, because all the other remaining functions depends on this connection. Just call `imVideoCaptureConnect` with one of the available capture device numbers.

You control when a device start processing frames using `imVideoCaptureLive`. Once live the frames can be captured using `imVideoCaptureFrame`. Or you can use `imVideoCaptureOneFrame`, it will start capturing, returns the captured frame and stop capturing.

But before capturing a frame you may want to configure the device. You can do it using Attributes, or at least in Windows you can do it using the configuration dialogs with a call to `imVideoCaptureShowDialog`.

A very simple sequence of operations to capture just one frame from the first device available:

```
imVideoCapture* vc = imVideoCaptureCreate();
if (!imVideoCaptureConnect(vc, 0))
    return;

int width, height;
imVideoCaptureGetImageSize(vc, &width, &height);

// initializes the data pointer
void* data = malloc(width*height*3);

imVideoCaptureOneFrame(vc, data, IM_RGB);
imVideoCaptureDestroy(vc);
```

The capture library is completely independent from the other libraries. It

just uses the same description of the data buffer used in `imFileReadImageData`.

See: [Guide / Architecture / Image Capture](#),
[Reference / Image Capturing](#).

Processing

The processing operations are very simple to use. Usually you just have to call the respective function. But you will have to ensure yourself that the image parameters for the input and output data are correct. Here is an example:

```
void imProcessFlip(const imImage* src_image, imImage* dst_image)
```

The processing operations are exclusive for the **imImage** structure. This makes the implementation cleaner and much easier to process color images since the planes are separated. But you can always use the `imImageInit` function to initializes an **imImage** structure with your own buffer.

See: [Guide / Architecture / Image Processing](#),
[Reference / Image Processing](#).

Advanced Guide

Memory I/O and Others

For the majority of the formats, with the exception of the ones that use external SDKs, the I/O is done by the **imBinFile** module.

This module can be configured to access other types of media by implementing a driver. There are some predefined drivers see [Reference / Utilities / Binary File Access](#).

One very useful is the **Memory Buffer** where you can read and write a file in memory. The activation is very simple, it needs to happen just before the `imFileOpen/imFileNew` functions. But the file name must be a pointer to an `imBinMemoryFileName` structure instead of a string. See the example below:

```
int old_mode = imBinFileSetCurrentModule(IM_MEMFILE);

imBinMemoryFileName MemFileName; // This structure must
exists while the file remains open.
MemFileName.buffer = NULL; // Let the library initializes
the buffer,
                                // but it must be freed the the
application, free(MemFileName.buffer) MemFileName.size =
1024; // The initial size
MemFileName.reallocate = 1.5; // The reallocation will
increase 50% the buffer.
                                // This is used only when
writing with a variable buffer.
                                // Use 0 to fix the buffer
size.

int error;
imFile* ifile = imFileNew((const char*)&MemFileName, "GIF",
&error);

imBinFileSetCurrentModule(old_mode); // The mode needs to
be active only for the imFileOpen/imFileNew call.

if (error != IM_ERR_NONE) ....
```

Another driver interesting is the **Subfile** where you can read and write from

a file that is already open. This is very important for formats that can have an embedded format inside. In this module the `file_name` is a pointer to an `imBinFile` structure from any other module that uses the **imBinFile** functions. The `imBinFileSize` will return the full file size, but the `imBinFileSeekTo` and `imBinFileTell` functions will compensate the position when the subfile was open.

Using `imBinFileSetCurrentModule(IM_SUBFILE)` just like the example above will allow you to open a subfile using the `imFileOpen/imFileNew` functions.

New Operations

An operation complexity is directly affected by the number of data types it will operate.

If it is only one, than it is as simple as:

```
void DoProc(imbyte* data, int width, int height)
{
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            // Do something
            int offset = y * width + x;

            data[offset] = 0;
        }
    }
}

void SampleProc(imImage* image)
{
    // a loop for all the color planes
    for (int d = 0; d < image->depth; d++)
    {
        // Notice that the same operation may be used to process each
        DoProc((imbyte*)image->data[d], image->width, image->height)
    }
}
```

Or if you want to use templates to allow a more number of types:

```

template <class T>
void DoProc2(const T* src_data, T* dst_data, int count)
{
    for (int i = 0; i < count; i++)
    {
        src_data[i] = dst_data[i];

        // or a more low level approach

        *src_data++ = *dst_data++;
    }
}

// This is a sample that do not depends on the spatial distribut
// It uses data[0], the pointer where all depths depends on.

void SampleProc2(const imImage* src_image, imImage* dst_image)
{
    int total_count = src_image->count * src_image->depth;
    switch(src_image->data_type)
    {
    case IM_BYTE:
        DoProc((imbyte*)src_image->data[0], (imbyte*)dst_image->data[0]);
        break;
    case IM USHORT:
        DoProc((imushort*)src_image->data[0], (imushort*)dst_image->data[0]);
        break;
    case IM_INT:
        DoProc((int*)src_image->data[0], (int*)dst_image->data[0]);
        break;
    case IM_FLOAT:
        DoProc((float*)src_image->data[0], (float*)dst_image->data[0]);
        break;
    case IM_CFLOAT:
        DoProc((imcfloat*)src_image->data[0], (imcfloat*)dst_image->data[0]);
        break;
    }
}

```

The first sample can be implemented in C, but the second sample can not, it must be in C++. Check the manual and the source code for many operations already available.

New File Formats

Again the easiest way is to look at the source code of an already implemented format. The RAS, BMP, TGA and SGI formats are very simple to follow.

Basically you have to implement a class that inherits from **imFormat** and implement its virtual methods. You can use the **imBinFile** functions for I/O or use an external SDK.

For more information see [Reference / Image Storage / File Format SDK](#).

Counters

To add support for the counter callback to a new operation is very simple. The following code shows how:

```
int counter = imCounterBegin("Process Test 1");
imCounterTotal(counter, count_steps, "Processing");

for (int i = 0; i < count_steps; i++)
{
    // Do something

    if (!imCounterInc(counter))
        return IM_ERR_COUNTER;
}

imCounterEnd(counter);
```

Every time you call **imCounterTotal** between a **imCounterBegin/imCounterEnd** for the same counter means that you are starting a count at that counter. So one operation can be composed by many sub-operations and still have a counter to display progress. For example, each call to the **imFileReadImageData** starts a new count for the same counter.

A nice thing to do when counting is not to display too small progress. To accomplish that in the implementation of the counter callback consider a minimum delay from one display to another.

See [Reference / Utilities / Counter](#).

Names Convention

To improve the readability of the code we use a very simple naming convention:

- Global Functions and Types - "im[Object][Action]" using first capitals (imFileOpen)
- Local Functions and Types - "i[Object][Action]" using first capitals (iTIFFGetCompIndex)
- Local Static Variables - same as local functions and types (iFormatCount)
- Local Static Tables - same as local functions and types with "Table" suffix (iTIFFCompTable)
- Variables and Members - no prefix, all lower case (width)
- Defines and Enumerations - all capitals (IM_ERR_NONE)

C x C++ Usage

The library main API is in C. We adopt this because of the many C programmers out there. Some of the API is also available in C++ for those addicted to classes.

Internally C++ is used to implement the format driver base architecture. A virtual base class that every drivers inherits from. This made a lot of things easier to the driver development. But we keep it simple, no multiple inheritance, no exception handling, no complicated classes.

But because we need several data types C++ templates were inevitable used (since we do not like long macros everywhere). But they are used only for processing functions, not classes.

IM Modules

Here is a list of all modules:

- **Image Capture**
 - Windows Attributes
- **Library Management**
- **Image Representation**
 - Image Conversion
 - Image Structure
 - Image Utilities
 - Color Mode Utilities
- **Image Storage**
 - File Format SDK
 - Read Access
 - Write Access
 - File Formats
 - TIFF - Tagged Image File Format
 - JPEG - JPEG File Interchange Format
 - PNG - Portable Network Graphic Format
 - GIF - Graphics Interchange Format
 - BMP - Windows Device Independent Bitmap
 - RAS - Sun Raster File
 - LED - IUP image in LED
 - SGI - Silicon Graphics Image File Format
 - PCX - ZSoft Picture
 - TGA - Truevision Graphics Adapter File
 - PNM - Netpbm Portable Image Map
 - ICO - Windows Icon
 - KRN - IM Kernel File Format
 - AVI - Windows Audio-Video Interleaved RIFF
 - JP2 - JPEG-2000 JP2 File Format
 - RAW - RAW File
 - WMV - Windows Media Video Format
- **Image Processing**
 - Image Statistics Calculations

- **Image Analysis**
- **Domain Transform Operations**
- **Image Resize**
- **Geometric Operations**
- **Morphology Operations for Gray Images**
- **Morphology Operations for Binary Images**
- **Rank Convolution Operations**
- **Convolution Operations**
- **Arithmetic Operations**
- **Additional Image Quantization Operations**
- **Histogram Based Operations**
- **Color Processing Operations**
- **Logical Arithmetic Operations**
- **Synthetic Image Render**
- **Tone Gamut Operations**
- **Threshold Operations**
- **Special Effects**
- **Utilities**
 - **Binary File Access**
 - **Color Manipulation**
 - **HSI Color Coordinate System Conversions**
 - **Complex Numbers**
 - **Counter**
 - **Windows DIB**
 - **Math Utilities**
 - **Palette Generators**
 - **String Utilities**
 - **Color Utilities**
 - **Data Type Utilities**
 - **Binary Data Utilities**
 - **Data Compression Utilities**

Library Management

Detailed Description

Usefull definitions for about dialogs and for checking the compiled version with the linked version for dynamic libraries.

See [im_lib.h](#)

Defines

```
#define IM_AUTHOR "Antonio Scuri"  
#define IM_COPYRIGHT "Copyright (C) 1994-2004 Tecgraf/PUC-Rio  
and PETROBRAS S/A"  
#define IM_VERSION "3.0.3"  
#define IM_VERSION_DATE "2004/10/14"  
#define IM_VERSION_NUMBER 300003
```

Functions

```
const char * imVersion (void)
const char * imVersionDate (void)
    int imVersionNumber (void)
```

Define Documentation

```
#define IM_VERSION_NUMBER 300003
```

Library release number used in the compilation time.

You can compare this with the value returned by [imVersionNumber](#).

Function Documentation

const char* imVersion(void)

Returns the library current version.

const char* imVersionDate(void)

Returns the library current version release date.

int imVersionNumber(void)

Returns the library current version number.

Image Capture

Detailed Description

Captures images from video devices.

You must link the application with "im_capture.lib/.a/.so".

Depends also on the Direct Show 9 SDK, you must link with "strmiids.lib".

When using the "im_capture.dll" this extra library is not necessary. Since DX uses COM, CoInitialize(NULL) is called when the devices are enumerated.

For more information:

http://msdn.microsoft.com/library/en-us/directx9_c/directX/htm/directshow.asp

See **im_capture.h**

Modules

group [Windows Attributes](#)

Functions

```
int imVideoCaptureDeviceCount (void)
const char * imVideoCaptureDeviceDesc (int device)
    int imVideoCaptureReloadDevices (void)
imVideoCapture * imVideoCaptureCreate (void)
    void imVideoCaptureDestroy (imVideoCapture *vc)
    int imVideoCaptureConnect (imVideoCapture *vc, int
        device)
    void imVideoCaptureDisconnect (imVideoCapture *vc)
    int imVideoCaptureShowDialog (imVideoCapture *vc,
        int dialog, void *parent)
    int imVideoCaptureDialogCount (imVideoCapture *vc)
const char * imVideoCaptureDialogDesc (imVideoCapture *vc,
    int dialog)
    void imVideoCaptureGetImageSize (imVideoCapture *vc,
        int *width, int *height)
    int imVideoCaptureSetImageSize (imVideoCapture *vc,
        int width, int height)
    int imVideoCaptureFrame (imVideoCapture *vc,
        unsigned char *data, int color_mode, int timeout)
    int imVideoCaptureOneFrame (imVideoCapture *vc,
        unsigned char *data, int color_mode)
    int imVideoCaptureLive (imVideoCapture *vc, int live)
    int imVideoCaptureResetAttribute (imVideoCapture
        *vc, const char *attrib, int fauto)
    int imVideoCaptureGetAttribute (imVideoCapture *vc,
        const char *attrib, float *percent)
    int imVideoCaptureSetAttribute (imVideoCapture *vc,
        const char *attrib, float percent)
const char ** imVideoCaptureGetAttributeList (imVideoCapture
```

`*vc, int *num_attrib)`

Function Documentation

`int imVideoCaptureDeviceCount(void)`

Returns the number of available devices.

`const char* imVideoCaptureDeviceDesc(int device)`

Returns the device description. Returns NULL in the last device.

`int imVideoCaptureReloadDevices(void)`

Reload the device list. The devices can be dynamically removed or added to the system.

`imVideoCapture* imVideoCaptureCreate(void)`

Creates a new imVideoCapture object.

Returns NULL if there is no capture device available.

In Windows returns NULL if DirectX version is older than 8.

`void imVideoCaptureDestroy(imVideoCapture * vc)`

Destroys a imVideoCapture object.

<code>int imVideoCaptureConnect(imVideoCapture * <i>vc</i>,</code>		
	<code>int</code>	<code><i>device</i></code>
	<code>)</code>	

Connects to a capture device. More than one imVideoCapture object can be created but they must be connected to different devices.
If the object is conected it will disconnect first.
Use -1 to return the current connected device, in this case returns -1 if not connected.
Returns zero if failed.

void imVideoCaptureDisconnect (imVideoCapture * *vc*)

Disconnect from a capture device.

int imVideoCaptureShowDialog (imVideoCapture * <i>vc</i>,		
	int	<i>dialog,</i>
	void *	<i>parent</i>
)	

Displays a configuration modal dialog of the connected device.
In Windows, the capturing will be stopped in some cases.
In Windows parent is a HWND of a parent window, it can be NULL.
Returns zero if failed.

int imVideoCaptureDialogCount (imVideoCapture * *vc*)

Returns the number of available configuration dialogs.

const char* imVideoCaptureDialogDesc (imVideoCapture * <i>vc</i>,		
	int	<i>dialog</i>
)	

Returns the description of a configuration dialog.

<code>void imVideoCaptureGetImageSize (imVideoCapture * <i>vc</i>,</code>		
	<code>int * <i>width</i>,</code>	
	<code>int * <i>height</i></code>	
<code>)</code>		

Returns the current image size of the connected device.
 width and height returns 0 if not connected.

<code>int imVideoCaptureSetImageSize (imVideoCapture * <i>vc</i>,</code>		
	<code>int <i>width</i>,</code>	
	<code>int <i>height</i></code>	
<code>)</code>		

Changes the image size of the connected device.
 Returns zero if failed.

<code>int imVideoCaptureFrame (imVideoCapture * <i>vc</i>,</code>		
	<code>unsigned char * <i>data</i>,</code>	
	<code>int <i>color_mode</i>,</code>	
	<code>int <i>timeout</i></code>	
<code>)</code>		

Returns a new captured frame. Use -1 for infinite timeout.
 Color space can be IM_RGB or IM_GRAY, and mode can be packed
 (IM_PACKED) or not.
 It can not have an alpha channel and orientation is always bottom up.
 Returns zero if failed or timeout expired, the buffer is not changed.

<code>int imVideoCaptureOneFrame (imVideoCapture * <i>vc</i>,</code>		
	<code>unsigned char * <i>data</i>,</code>	
	<code>int <i>color_mode</i></code>	
<code>)</code>		

Start capturing, returns the new captured frame and stop capturing.
This is more usefull if you are switching between devices.
Data format is the same as imVideoCaptureFrame.
Returns zero if failed.

int imVideoCaptureLive (imVideoCapture * vc,	
	int	live
)		

Start capturing.
Use -1 to return the current state.
Returns zero if failed.

int imVideoCaptureResetAttribute (imVideoCapture * vc,	
	const char * attrib,	
	int fauto	
)		

Resets a camera or video attribute to the default value or to the automatic setting.
Not all attributes support automatic modes.
Returns zero if failed.

int imVideoCaptureGetAttribute (imVideoCapture * vc,	
	const char * attrib,	
	float * percent	
)		

Returns a camera or video attribute in percentage of the valid range value.
Returns zero if failed.

int imVideoCaptureSetAttribute (imVideoCapture * <i>vc</i>,
	const char * <i>attrib</i>,
	float <i>percent</i>
)	

Changes a camera or video attribute in percentage of the valid range value.

Returns zero if failed.

const char** imVideoCaptureGetAttributeList (imVideoCapture * <i>vc</i>,
	int * <i>numAttr</i>
)	

Returns a list of the description of the valid attributes.

Windows Attributes

[Image Capture]

VideoBrightness - Specifies the brightness, also called the black
VideoContrast - Specifies the contrast, expressed as gain factor.
VideoHue - Specifies the hue angle.
VideoSaturation - Specifies the saturation.
VideoSharpness - Specifies the sharpness.
VideoGamma - Specifies the gamma.
VideoColorEnable - Specifies the color enable setting. (0/100)
VideoWhiteBalance - Specifies the white balance, as a color temper
VideoBacklightCompensation - Specifies the backlight compensation
VideoGain - Specifies the gain adjustment.
CameraPanAngle - Specifies the camera's pan angle. To 100 rotate r
CameraTiltAngle - Specifies the camera's tilt angle. To 100 rotat
CameraRollAngle - Specifies the camera's roll angle. To 100 rotate
CameraLensZoom - Specifies the camera's zoom setting.
CameraExposure - Specifies the exposure setting.
CameraIris - Specifies the camera's iris setting.
CameraFocus - Specifies the camera's focus setting, as the distanc
FlipHorizontal - Specifies the video will be flipped in the horizo
FlipVertical - Specifies the video will be flipped in the vertical
AnalogFormat - Specifies the video format standard NTSC, PAL, etc
 NTSC_M = 0
 NTSC_M_J = 1
 NTSC_433 = 2
 PAL_B = 3
 PAL_D = 4
 PAL_H = 5
 PAL_I = 6
 PAL_M = 7
 PAL_N = 8
 PAL_60 = 9
 SECAM_B = 10
 SECAM_D = 11
 SECAM_G = 12
 SECAM_H = 13
 SECAM_K = 14
 SECAM_K1 = 15
 SECAM_L = 16
 SECAM_L1 = 17
 PAL_N_COMBO = 18

Image Representation

Detailed Description

See [im.h](#)

Modules

group [**Image Conversion**](#)
group [**Image Structure**](#)
group [**Image Utilities**](#)
group [**Color Mode Utilities**](#)

Enumerations

```
enum imDataType {
    IM_BYTE, IM USHORT, IM_INT, IM_FLOAT,
    IM_CFLOAT
}
enum imColorSpace {
    IM_RGB, IM_MAP, IM_GRAY, IM_BINARY,
    IM_CMYK, IM_YCBCR, IM_LAB, IM_LUV,
    IM_XYZ
}
enum imColorModeConfig { IM_ALPHA = 0x100, IM_PACKED =
    0x200, IM_TOPDOWN = 0x400 }
```

Enumeration Type Documentation

enum [imDataType](#)

Image data type descriptors.

See also [Data Type Utilities](#).

Enumeration values:

- IM_BYTE* "unsigned char". 1 byte from 0 to 255.
- IM USHORT* "unsigned short". 2 bytes from 0 to 65,535.
- IM_INT* "int". 4 bytes from -2,147,483,648 to 2,147,483,647.
- IM_FLOAT* "float". 4 bytes single precision IEEE floating point.
- IM_CFLOAT* complex "float". 2 float values in sequence, real and i

```
00022 {  
00023     IM_BYTE,    /**< "unsigned char". 1 byte from 0 to 255.  
00024     IM USHORT,  /**< "unsigned short". 2 bytes from 0 to 65,535  
00025     IM INT,     /**< "int". 4 bytes from -2,147,483,648 to 2,147,483,647.  
00026     IM FLOAT,   /**< "float". 4 bytes single precision IEEE floating point.  
00027     IM CFLOAT   /**< complex "float". 2 float values in sequence.  
00028 };
```

enum [imColorSpace](#)

Image color mode color space descriptors (first byte).

See also [Color Mode Utilities](#).

Enumeration values:

- IM_RGB* Red, Green and Blue (nonlinear).
- IM_MAP* Indexed by RGB color map (data_type=IM_BYTE).
- IM_GRAY* Shades of gray, luma (nonlinear Luminance), or an integer.
- IM_BINARY* Indexed by 2 colors: black (0) and white (1) (data_type=IM_BYTE).
- IM_CMYK* Cian, Magenta, Yellow and Black (nonlinear).

IM_YCBCR ITU-R 601 Y'CbCr. Y' is luma (nonlinear Luminance).
IM_LAB CIE L*a*b*. L* is Lightness (nonlinear Luminance, nea
IM_LUV CIE L*u*v*. L* is Lightness (nonlinear Luminance, nea
IM_XYZ CIE XYZ. Linear Light Tristimulus, Y is linear Luminan

```

00034 {
00035     IM_RGB,      /**< Red, Green and Blue (nonlinear).
00036     IM_MAP,      /**< Indexed by RGB color map (data_type=IM_BYTE)
00037     IM_GRAY,     /**< Shades of gray, luma (nonlinear Luminance),
00038     IM_BINARY,   /**< Indexed by 2 colors: black (0) and white (1)
00039     IM_CMYK,     /**< Cian, Magenta, Yellow and Black (nonlinear)
00040     IM_YCBCR,   /**< ITU-R 601 Y'CbCr. Y' is luma (nonlinear Luminance)
00041     IM_LAB,      /**< CIE L*a*b*. L* is Lightness (nonlinear Luminance)
00042     IM_LUV,      /**< CIE L*u*v*. L* is Lightness (nonlinear Luminance)
00043     IM_XYZ      /**< CIE XYZ. Linear Light Tristimulus, Y is linear Luminance
00044 };
    
```

enum imColorModeConfig

Image color mode configuration/extr descriptors (1 bit each in the second byte).

See also [Color Mode Utilities](#).

Enumeration values:

IM_ALPHA adds an Alpha channel
IM_PACKED packed components (rgbgrgbgrgb...)
IM_TOPDOWN orientation from top down to bottom

```

00050 {
00051     IM_ALPHA     = 0x100,  /**< adds an Alpha channel */
00052     IM_PACKED   = 0x200,  /**< packed components (rgbgrgbgrgb...)
00053     IM_TOPDOWN  = 0x400   /**< orientation from top down to bottom
00054 };
    
```

Image Structure

[Image Representation]

Detailed Description

Base definitions and functions for image representation.
Only the image processing operations depends on these definitions, Image Storage and Image Capture are completely independent.

You can also initialize a structure with your own memory buffer, see [imImageInit](#). To release the structure without releasing the buffer, set "data[0]" to 0 before calling imImageDestroy.

See [im_image.h](#)

Data Structures

struct [_imlImage](#)

Image Structure Definition. [More...](#)

Defines

```
#define cdPutBitmap(_image, _x, _y, _w, _h, _xmin, _xmax, _ymin,  
_ymax)
```

Typedefs

```
typedef _imImage imImage
```

Functions

```
imlImage * imlImageCreate (int width, int height, int color_space, int
                           data_type)
imlImage * imlImageInit (int width, int height, int color_space, int
                           data_type, void *data_buffer, long *palette, int
                           palette_count)
void imlImageDestroy (imlImage *image)
void imlImageReshape (imlImage *image, int width, int height)
void imlImageCopy (const imlImage *src_image, imlImage
                  *dst_image)
void imlImageCopyData (const imlImage *src_image, imlImage
                      *dst_image)
imlImage * imlImageDuplicate (const imlImage *image)
imlImage * imlImageClone (const imlImage *image)
void imlImageSetAttribute (imlImage *image, const char *attrib,
                           int data_type, int count, const void *data)
const void * imlImageGetAttribute (const imlImage *image, const char
                                   *attrib, int *data_type, int *count)
void imlImageGetAttributeList (const imlImage *image, char
                               **attrib, int *attrib_count)
void imlImageClear (imlImage *image)
int imlImageIsBitmap (const imlImage *image)
void imlImageSetPalette (imlImage *image, long *palette, int
                        palette_count)
void imlImageCopyAttributes (const imlImage *src_image,
                            imlImage *dst_image)
int imlImageMatchSize (const imlImage *image1, const
                      imlImage *image2)
int imlImageMatchColor (const imlImage *image1, const
                       imlImage *image2)
```

```
int imlImageMatchDataType (const imlImage *image1, const
                           imlImage *image2)
int imlImageMatchColorSpace (const imlImage *image1,
                            const imlImage *image2)
int imlImageMatch (const imlImage *image1, const imlImage
                  *image2)
imlImage * imFileLoadImage (imFile *ifile, int index, int *error)
imlImage * imFileLoadBitmap (imFile *ifile, int index, int *error)
           int imFileSaveImage (imFile *ifile, const imlImage *image)
imlImage * imlImageLoad (const char *file_name, int index, int *error)
imlImage * imlImageLoadBitmap (const char *file_name, int index, int
                               *error)
void imlImageSetBinary (imlImage *image)
void imlImageMakeBinary (imlImage *image)
```

Define Documentation

```
#define cdPutBitmap ( _image,  
                      _x,  
                      _y,  
                      _w,  
                      _h,  
                      _xmin,  
                      _xmax,  
                      _ymin,  
                      _ymax )
```

Value:

```
{  
    if (_image->color_space == IM_RGB)  
        cdPutImageRectRGB(_image->width, _image->height,  
                           (unsigned char*)_image->data[0],  
                           (unsigned char*)_image->data[1],  
                           (unsigned char*)_image->data[2],  
                           _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax)  
    else  
        cdPutImageRectMap(_image->width, _image->height,  
                           (unsigned char*)_image->data[0], _image->pa  
                           _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax)  
}
```

Utility macro to draw the image in a CD library canvas. Works only for data type IM_BYTE, and color spaces: IM_RGB, IM_MAP, IMGRAY and IM_BINARY.

Typedef Documentation

typedef struct imlImage imlImage

Image Structure Definition.

An image representation than supports all the color spaces, but no alpha channel, planes are always unpacked and the orientation is always bottom up.

Function Documentation

```
imlImage* imlImageCreate( int width,  
                         int height,  
                         int color_space,  
                         int data_type  
                       )
```

Creates a new image.

```
imlImage* imlImageInit( int width,  
                        int height,  
                        int color_space,  
                        int data_type,  
                        void * data_buffer,  
                        long * palette,  
                        int palette_count  
                      )
```

Initializes the image structure but does not allocates image data.

```
void imlImageDestroy( imlImage * image )
```

Destroys the image and frees the memory used. image data is destroyed only if it data[0] is not NULL.

```
void imlImageReshape( imlImage * image,  
                      int width,  
                      int height
```

)

Changes the buffer size. Reallocate internal buffers if they are larger than original.

void imlImageCopy(const imlImage * src_image,	
	imlImage * dst_image	
)		

Copy image data and attributes from one image to another.
Images must have the same size and type.

void imlImageCopyData(const imlImage * src_image,	
	imlImage * dst_image	
)		

Copy image data only one image to another.
Images must have the same size and type.

imlImage* imlImageDuplicate(const imlImage * image)

Creates a copy of the image.

imlImage* imlImageClone(const imlImage * image)

Creates a clone of the image. i.e. same attributes but ignore contents.

void imlImageSetAttribute(imlImage * image,	
	const char * attrib,	
	int data_type,	
	int count,	

	const void * <i>data</i>
)	

Changes an extended attribute.

The data will be internally duplicated.

If data is NULL the attribute is removed.

If count is -1 and data_type is IM_BYTE then data is zero terminated.

const void* imlImageGetAttribute(const imlImage * <i>image</i>,	
---	--

	const char * <i>attrib</i>,
--	------------------------------------

	int * <i>data_type</i>,
--	--------------------------------

	int * <i>count</i>
--	---------------------------

)

Returns an extended attribute.

Returns NULL if not found.

void imlImageGetAttributeList(const imlImage * <i>image</i>,	
--	--

	char ** <i>attrib</i>,
--	-------------------------------

	int * <i>attrib_count</i>
--	----------------------------------

)

Returns a list of the attribute names.

"attrib" must contain room enough for "attrib_count" names. Use

"attrib=NULL" to return only the count.

void imlImageClear(imlImage * <i>image</i>)	
--	--

Sets all image data to zero.

int imlImageIsBitmap(const imlImage * <i>image</i>)	
--	--

Indicates that the image can be viewed in common graphic devices. Data type must be IM_BYTE. Color mode can be IM_RGB, IM_MAP, IM_GRAY or IM_BINARY.

```
void imImageSetPalette( imImage * image,  
                        long * palette,  
                        int palette_count  
                      )
```

Changes the image palette. This will destroy the existing palette and replace it with the given palette buffer.

```
void imImageCopyAttributes( const imImage * src_image,  
                           imImage * dst_image  
                         )
```

Copies the image attributes from src to dst.

```
int imImageMatchSize( const imImage * image1,  
                      const imImage * image2  
                    )
```

Returns 1 if the images match width and height. Returns 0 otherwise.

```
int imImageMatchColor( const imImage * image1,  
                      const imImage * image2  
                    )
```

Returns 1 if the images match color mode and data type. Returns 0 otherwise.

int imImageMatchDataType (const imImage * image1,
	const imImage * image2
)	

Returns 1 if the images match width, height and data type. Returns 0 otherwise.

int imImageMatchColorSpace (const imImage * image1,
	const imImage * image2
)	

Returns 1 if the images match width, height and color space. Returns 0 otherwise.

int imImageMatch (const imImage * image1,
	const imImage * image2
)	

Returns 1 if the images match in width, height, data type and color space. Returns 0 otherwise.

imImage* imFileLoadImage (imFile * ifile,
	int index,
	int * error
)	

Loads an image from file. Returns NULL if failed. This will call imFileReadImageInfo and imFileReadImageData.

imImage* imFileLoadBitmap (imFile * ifile,
	int index,

	int *	error
)

Loads an image from file, but forces the image to be a bitmap.
Returns NULL if failed.

int imFileSaveImage (imFile *	<i>ifile,</i>
	const imImage *	<i>image</i>
)	

Saves the image to file. Returns error code.
This will call imFileWriteImageInfo and imFileWriteImageData.

imImage* imImageLoad (const char *	<i>file_name,</i>
	int	<i>index,</i>
	int *	error
)	

Loads an image from file.
Returns NULL if failed.

imImage* imImageLoadBitmap (const char *	<i>file_name,</i>
	int	<i>index,</i>
	int *	error
)	

Loads an image from file, but forces the image to be a bitmap.
Returns NULL if failed.

void imImageSetBinary (**imImage *** *image* **)**

Changes the image space from gray to binary by just changing color_space and the palette.

```
void imlImageMakeBinary( imlImage * image )
```

Changes a gray data into a binary data.

Image Conversion

[Image Representation]

Detailed Description

Converts one type of image into another. Can convert between color modes and between data types.

See [im_convert.h](#)

Enumerations

```
enum imComplex2Real { IM_CPX_REAL, IM_CPX_IMAG,
    IM_CPX_MAG, IM_CPX_PHASE }

enum imGammaFactor {
    IM_GAMMA_LINEAR = 0, IM_GAMMA_LOGLITE = -10,
    IM_GAMMA_LOGHEAVY = -1000, IM_GAMMA_EXPLITE = 2,
    IM_GAMMA_EXPHEAVY = 7
}
enum imCastMode { IM_CAST_MINMAX, IM_CAST_FIXED,
    IM_CAST_DIRECT }
```

Functions

```
int imConvertDataType (const imImage *src_image, imImage  
*dst_image, int cpx2real, float gamma, int abssolute, int  
cast_mode)  
int imConvertColorSpace (const imImage *src_image, imImage  
*dst_image)  
int imConvertToBitmap (const imImage *src_image, imImage  
*dst_image, int cpx2real, float gamma, int abssolute, int  
cast_mode)  
void imConvertPacking (const void *src_data, void *dst_data, int width,  
int height, int depth, int data_type, int src_is_packed)  
void imConvertMapToRGB (unsigned char *data, int count, int depth,  
int packed, long *palette, int palette_count)
```

Enumeration Type Documentation

enum imComplex2Real

Complex to real conversions

```
00030 {  
00031     IM_CPX_REAL,  
00032     IM_CPX_IMAG,  
00033     IM_CPX_MAG,  
00034     IM_CPX_PHASE  
00035 };
```

enum imGammaFactor

Predefined Gamma factors

```
00040 {  
00041     IM_GAMMA_LINEAR    = 0,  
00042     IM_GAMMA_LOGLITE   = -10,  
00043     IM_GAMMA_LOGHEAVY = -1000,  
00044     IM_GAMMA_EXPLITE  = 2,  
00045     IM_GAMMA_EXPHEAVY = 7  
00046 };
```

enum imCastMode

Predefined Cast Modes

Enumeration values:

IM_CAST_MINMAX scan for min and max values

IM_CAST_FIXED use predefined 0-max values, see [Color Manipulation](#).

IM_CAST_DIRECT direct type cast the value. Only byte and ushort.

```
00051 {  
00052     IM_CAST_MINMAX, /*< scan for min and max values */  
00053     IM_CAST_FIXED, /*< use predefined 0-max values, see \ref c  
00054     IM_CAST_DIRECT /*< direct type cast the value. Only byte  
00055 };
```

Function Documentation

<code>int imConvertDataType(const imImage * <i>src_image</i>,</code>		
<code> imImage * <i>dst_image</i>,</code>		
<code> int <i>cpx2real</i>,</code>		
<code> float <i>gamma</i>,</code>		
<code> int <i>absolute</i>,</code>		
<code> int <i>cast_mode</i></code>		
<code>)</code>		

Changes the image data type, using a complex2real conversion, a gamma factor, and an absolute mode (modulus).

When demoting the data type the function will scan for min/max values or use fixed values (*cast_mode*) to scale the result according to the destiny range.

Except complex to real that will use only the complex2real conversion.

Images must be of the same size and color mode.

Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.

<code>int imConvertColorSpace(const imImage * <i>src_image</i>,</code>		
<code> imImage * <i>dst_image</i></code>		
<code>)</code>		

Converts one color space to another. Images must be of the same size and data type.

CMYK can be converted to RGB only, and it is a very simple conversion.

All colors can be converted to Binary, the non zero gray values are converted to 1.

RGB to Map uses the median cut implementation from the free IJG JPEG software, copyright Thomas G. Lane.

All other color space conversions assume sRGB and CIE definitions.

Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.

int imConvertToBitmap (const imImage * src_image,
	imImage * dst_image,
	int cpx2real,
	float gamma,
	int abssolute,
	int cast_mode
)	

Converts the image to its bitmap equivalent, uses
imConvertColorSpace and **imConvertDataType**.

Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.

void imConvertPacking (const void * src_data,
	void * dst_data,
	int width,
	int height,
	int depth,
	int data_type,
	int src_is_packed
)	

Changes the packing of the data buffer.

void imConvertMapToRGB (unsigned char * data,
	int count,
	int depth,
	int packed,
	long * palette,
	int palette_count

)

Changes in-place a MAP data into a RGB data. The data must have room for the RGB image.

depth can be 3 or 4. count=width*height.

Very usefull for OpenGL applications.

Image Utilities

[Image Representation]

Detailed Description

See [**im_util.h**](#)

Functions

int **imImageContentSize** (int width, int height, int color_mode, int data_type)

int **imImageLineSize** (int width, int color_mode, int data_type)

int **imImageLineCount** (int width, int color_mode)

int **imImageCheckFormat** (int color_mode, int data_type)

Function Documentation

```
int imImageContentSize( int width,  
                        int height,  
                        int color_mode,  
                        int data_type  
)
```

Returns the size of the data buffer.

```
int imImageLineSize( int width,  
                     int color_mode,  
                     int data_type  
)
```

Returns the size of one line of the data buffer.

This depends if the components are packed. If packed includes all components, if not includes only one.

```
int imImageLineCount( int width,  
                      int color_mode  
)
```

Returns the number of elements of one line of the data buffer.

This depends if the components are packed. If packed includes all components, if not includes only one.

```
int imImageCheckFormat( int color_mode,  
                       int data_type
```

)
Check the combination color_mode+data_type.

Color Mode Utilities

[Image Representation]

Detailed Description

See [im_util.h](#)

Defines

```
#define imColorModeSpace(_cm) (_cm & 0xFF)
#define imColorModeMatch(_cm1,
    _cm2) (imColorModeSpace(_cm1) ==
    imColorModeSpace(_cm2))
#define imColorModeHasAlpha(_cm) (_cm & IM_ALPHA)
#define imColorModeIsPacked(_cm) (_cm & IM_PACKED)
#define imColorModeIsTopDown(_cm) (_cm & IM_TOPDOWN)
```

Functions

```
const char * imColorModeSpaceName (int color_mode)
    int imColorModeDepth (int color_mode)
    int imColorModeToBitmap (int color_mode)
    int imColorModeIsBitmap (int color_mode, int data_type)
```

Define Documentation

```
#define imColorModeSpace( _cm ) (_cm & 0xFF)
```

Returns the color space of the color mode.

```
#define imColorModeMatch( _cm1,  
                         _cm2 ) (imColorModeSpace(_cm1)
```

Check if the two color modes match. Only the color space is compared.

```
#define imColorModeHasAlpha( _cm ) (_cm & IM_ALPHA)
```

Check if the color mode has an alpha channel.

```
#define imColorModelsPacked( _cm ) (_cm & IM_PACKED)
```

Check if the color mode components are packed in one plane.

```
#define imColorModelsTopDown( _cm ) (_cm & IM_TOPDOWN)
```

Check if the color mode orients the image from top down to bottom.

Function Documentation

```
const char* imColorModeSpaceName( int color_mode )
```

Returns the color mode name.

```
int imColorModeDepth( int color_mode )
```

Returns the number of components of the color space including alpha.

```
int imColorModeToBitmap( int color_mode )
```

Returns the color mode of the equivalent display bitmap image.
Original packing and alpha are ignored. Returns IM_RGB, IM_GRAY,
IM_MAP or IM_BINARY.

```
int imColorModelsBitmap( int color_mode,
                         int data_type
                       )
```

Check if the color mode and data_type defines a display bitmap image.

Image Storage

Detailed Description

See [im.h](#)

Modules

group **File Format SDK**
group **Read Access**
group **Write Access**
group **File Formats**

Data Structures

class [imImageFile](#)

C++ Wrapper for the Image File Structure. [More...](#)

Enumerations

```
enum imErrorCodes {
    IM_ERR_NONE, IM_ERR_OPEN, IM_ERR_ACCESS,
    IM_ERR_FORMAT,
    IM_ERR_DATA, IM_ERR_COMPRESS, IM_ERR_MEM,
    IM_ERR_COUNTER
}
```

Functions

```
void imFileClose (imFile *ifile)
void * imFileHandle (imFile *ifile)
void imFileSetAttribute (imFile *ifile, const char *attrib, int
data_type, int count, const void *data)
const void * imFileGetAttribute (imFile *ifile, const char *attrib, int
*data_type, int *count)
void imFileGetAttributeList (imFile *ifile, char **attrib, int
*attrib_count)
```

Enumeration Type Documentation

enum imErrorCodes

File Access Error Codes

Enumeration values:

<i>IM_ERR_NONE</i>	No error.
<i>IM_ERR_OPEN</i>	Error while opening the file (read or write).
<i>IM_ERR_ACCESS</i>	Error while accessing the file (read or write).
<i>IM_ERR_FORMAT</i>	Invalid or unrecognized file format.
<i>IM_ERR_DATA</i>	Invalid or unsupported data.
<i>IM_ERR_COMPRESS</i>	Invalid or unsupported compression.
<i>IM_ERR_MEM</i>	Insufficient memory
<i>IM_ERR_COUNTER</i>	Interrupted by the counter

```
00061 {
00062     IM_ERR_NONE,      /**< No error. */
00063     IM_ERR_OPEN,      /**< Error while opening the file (read or
00064     IM_ERR_ACCESS,    /**< Error while accessing the file (read or
00065     IM_ERR_FORMAT,    /**< Invalid or unrecognized file format.
00066     IM_ERR_DATA,      /**< Invalid or unsupported data. */
00067     IM_ERR_COMPRESS,  /**< Invalid or unsupported compression. */
00068     IM_ERR_MEM,       /**< Insufficient memory */
00069     IM_ERR_COUNTER    /**< Interrupted by the counter */
00070 };
```

Function Documentation

```
void imFileClose( imFile * ifile )
```

Closes the file

```
void* imFileHandle( imFile * ifile )
```

Returns the internal handle. It is file format dependent.

```
void imFileSetAttribute( imFile * ifile,  
                        const char * attrib,  
                        int data_type,  
                        int count,  
                        const void * data  
)
```

Changes an extended attribute.

The data will be internally duplicated.

If data is NULL the attribute is removed.

```
const void* imFileGetAttribute( imFile * ifile,  
                               const char * attrib,  
                               int * data_type,  
                               int * count  
)
```

Returns an extended attribute.

Returns NULL if not found. *data_type* and *count* can be NULL.

```
void imFileGetAttributeList( imFile * ifile,  
                            char ** attrib,  
                            int * attrib_count  
                          )
```

Returns a list of the attribute names.

"attrib" must contain room enough for "attrib_count" names. Use "attrib=NULL" to return only the count.

Read Access

[Image Storage]

Detailed Description

See [im.h](#)

Functions

```
imFile * imFileOpen (const char *file_name, int *error)
void imFileGetInfo (imFile *ifile, char *format, char *compression,
int *image_count)
void imFileGetPalette (imFile *ifile, long *palette, int
*palette_count)
int imFileReadImageInfo (imFile *ifile, int index, int *width, int
*height, int *file_color_mode, int *file_data_type)
int imFileReadImageData (imFile *ifile, void *data, int
convert2bitmap, int color_mode_flags)
```

Function Documentation

<code>imFile*</code>	<code>imFileOpen(</code>	<code>const char *</code>	<code>file_name,</code>
		<code>int *</code>	<code>error</code>
	<code>)</code>		

Opens the file for reading. It must exists. Also reads file header.

<code>void imFileGetInfo(</code>	<code>imFile *</code>	<code>ifile,</code>
	<code>char *</code>	<code>format,</code>
	<code>char *</code>	<code>compression,</code>
	<code>int *</code>	<code>image_count</code>
	<code>)</code>	

Returns file information. `image_count` is the number of images in a stack or the number of frames in a video/animation or the depth of a volume data.

`compression` and `image_count` can be NULL.

<code>void imFileGetPalette(</code>	<code>imFile *</code>	<code>ifile,</code>
	<code>long *</code>	<code>palette,</code>
	<code>int *</code>	<code>palette_count</code>
	<code>)</code>	

Returns the palette if any.

"`palette`" must be a 256 colors alocated array.

Returns zero in "`palette_count`" if there is no palette. "`palette_count`" is `>0` and `<=256`.

<code>int imFileReadImageInfo(</code>	<code>imFile *</code>	<code>ifile,</code>
---------------------------------------	-----------------------	---------------------

	int	<i>index,</i>
	int *	<i>width,</i>
	int *	<i>height,</i>
	int *	<i>file_color_mode,</i>
	int *	<i>file_data_type</i>
)	

Reads the image header if any and returns image information.
 Reads also the extended image attributes, so other image attributes will be available only after calling this function.
 Returns an error code. index specifies the image number between 0 and `image_count-1`.
 Some drivers reads only in sequence, so "index" can be ignored by the format driver.
 Any parameters can be NULL. This function must be called at least once, check each format documentation.

int imFileReadImageData(imFile *	<i>ifile,</i>
	void *	<i>data,</i>
	int	<i>convert2bitmap,</i>
	int	<i>color_mode_flags</i>
)	

Reads the image data with or without conversion.
 The data can be converted to bitmap when reading. Data type conversion to byte will always scan for min-max then scale to 0-255, except integer values that min-max are already between 0-255.
 Complex to real conversions will use the magnitude.
 Color mode flags contains packed, alpha and top-bottom information. If flag is 0 means unpacked, no alpha and bottom up. If flag is -1 the file original flags are used.
 Returns an error code.

Write Access

[Image Storage]

Detailed Description

See [im.h](#)

Functions

```
imFile * imFileNew (const char *file_name, const char *format, int  
*error)  
void imFileSetInfo (imFile *ifile, const char *compression)  
void imFileSetPalette (imFile *ifile, long *palette, int palette_count)  
int imFileWriteImageInfo (imFile *ifile, int width, int height, int  
user_color_mode, int user_data_type)  
int imFileWriteImageData (imFile *ifile, void *data)
```

Function Documentation

imFile*	imFileNew	(const char *	<i>file_name</i> ,
			const char *	<i>format</i> ,
			int *	<i>error</i>
)		

Creates a new file for writing. If the file exists will be replaced. It will only initialize the format driver and create the file, no data is actually written.

void	imFileSetInfo	(imFile *	<i>ifile</i> ,
			const char *	<i>compression</i>
)		

Changes the write compression method.
If the compression is not supported will return an error code when writting.
Use NULL to set the default compression. You can use the imFileGetInfo to retrieve the actual compression but only after imFileWriteImageInfo. Only a few formats allow you to change the compression between frames.

void	imFileSetPalette	(imFile *	<i>ifile</i> ,
			long *	<i>palette</i> ,
			int	<i>palette_count</i>
)		

Changes the palette.
"palette_count" is >0 and <=256.

int imFileWriteImageInfo (imFile *	<i>ifile,</i>
	int	<i>width,</i>
	int	<i>height,</i>
	int	<i>user_color_mode,</i>
	int	<i>user_data_type</i>
)		

Writes the image header. Writes the file header at the first time it is called. Writes also the extended image attributes.

Must call imFileSetPalette and set other attributes before calling this function.

In some formats the color space will be converted to match file format specification.

Returns an error code. This function must be called at least once, check each format documentation.

int imFileWriteImageData (imFile *	<i>ifile,</i>
	void *	<i>data</i>
)		

Writes the image data.

Returns an error code.

File Formats

[Image Storage]

Detailed Description

See [im.h](#)

Internal Predefined File Formats:

- "BMP" - Windows Device Independent Bitmap
- "PCX" - ZSoft Picture
- "GIF" - Graphics Interchange Format
- "TIFF" - Tagged Image File Format
- "RAS" - Sun Raster File
- "SGI" - Silicon Graphics Image File Format
- "JPEG" - JPEG File Interchange Format
- "LED" - IUP image in LED
- "TGA" - Truevision Targa
- "RAW" - RAW File
- "PNM" - Netpbm Portable Image Map
- "ICO" - Windows Icon
- "PNG" - Portable Network Graphic Format

Other Supported File Formats:

- "JP2" - JPEG-2000 JP2 File Format
- "AVI" - Windows Audio-Video Interleaved RIFF
- "WMV" - Windows Media Video Format

Some Known Compressions:

- "NONE" - No Compression.
- "RLE" - Run Length Encoding.
- "LZW" - Lempel, Ziff and Welsh.
- "JPEG" - Joint Photographic Experts Group.
- "DEFLATE" - LZ77 variation (ZIP)

Modules

- group [**TIFF - Tagged Image File Format**](#)
- group [**JPEG - JPEG File Interchange Format**](#)
- group [**PNG - Portable Network Graphic Format**](#)
- group [**GIF - Graphics Interchange Format**](#)
- group [**BMP - Windows Device Independent Bitmap**](#)
- group [**RAS - Sun Raster File**](#)
- group [**LED - IUP image in LED**](#)
- group [**SGI - Silicon Graphics Image File Format**](#)
- group [**PCX - ZSoft Picture**](#)
- group [**TGA - Truevision Graphics Adapter File**](#)
- group [**PNM - Netpbm Portable Image Map**](#)
- group [**ICO - Windows Icon**](#)
- group [**KRN - IM Kernel File Format**](#)
- group [**AVI - Windows Audio-Video Interleaved RIFF**](#)
- group [**JP2 - JPEG-2000 JP2 File Format**](#)
- group [**RAW - RAW File**](#)
- group [**WMV - Windows Media Video Format**](#)

Functions

```
void imFormatList (char **format_list, int *format_count)
int imFormatInfo (const char *format, char *desc, char *ext, int
    *can_sequence)
int imFormatCompressions (const char *format, char **comp, int
    *comp_count, int color_mode, int data_type)
int imFormatCanWriteImage (const char *format, const char
    *compression, int color_mode, int data_type)
```

Function Documentation

void imFormatList(char **	<i>format_list,</i>
	int *	<i>format_count</i>
)		

Returns a list of the registered formats.

format_list is an array of format identifiers. Each format identifier is 10 chars max, maximum of 50 formats. You can use "char* *format_list*[50]".

int imFormatInfo(const char *	<i>format,</i>
	char *	<i>desc,</i>
	char *	<i>ext,</i>
	int *	<i>can_sequence</i>
)		

Returns the format description.

Format description is 50 chars max.

Extensions are separated like "*.tif;*.tiff;", 50 chars max.

Returns an error code. The parameters can be NULL, except *format*.

int imFormatCompressions(const char *	<i>format,</i>
	char **	<i>comp,</i>
	int *	<i>comp_count,</i>
	int	<i>color_mode,</i>
	int	<i>data_type</i>
)		

Returns the format compressions.

Compressions are 20 chars max each, maximum of 50 compressions.
You can use "char* comp[50]".

color_mode and data_type are optional, use -1 to ignore them.
If you use them they will select only the allowed compressions checked
like in [imFormatCanWriteImage](#).

Returns an error code.

int imFormatCanWriteImage (const char * <i>format</i>,
	const char * <i>compression</i>,
	int <i>color_mode</i>,
	int <i>data_type</i>
)	

Checks if the format suport the given image class at the given
compression.

Returns an error code.

RAW - RAW File

[File Formats]

Detailed Description

The file must be open/created with the functions **imFileOpenRaw** and **imFileNewRaw**.

Description

Internal Implementation.

Supports RAW binary images. You must know image parameters a priori. You must set the IM_INT attributes "Width", "Height", "ColorMode", "DataType" before the imFileReadImageInfo/imFileWriteImageInfo functions.

The data must be in binary form, but can start in an arbitrary offset from the begining of the file, use attribute "StartOffset". The default is at 0 offset.

Integer sign and double precision can be converted using attribute "SwitchType".

The conversions will be BYTE<->CHAR, USHORT<->SHORT, INT<->UINT, FLOAT<->DOUBLE.

Byte Order can be Little Endian (Intel=1) or Big Endian (Motorola=0), use the attribute "ByteOrder", the default is the current CPU.

The lines can be aligned to a BYTE (1), WORD (2) or DWORD (4) boundaries, ue attribute "Padding" with the respective value.

See [im_raw.h](#)

Features

Data Types: <all>

Color Spaces: all, except MAP.

Compressions:

NONE - no compression

Can have more than one image, depends on "StartOffset" attribute

Can have an alpha channel.

Components can be packed or not.

Lines arranged from top down to bottom or bottom up to top.

Handle() returns a imBinFile* pointer.

Attributes:

Width, Height, ColorMode, DataType IM_INT (1)

StartOffset, SwitchType, ByteOrder, Padding IM_INT (1)

Functions

imFile * imFileOpenRaw (const char *file_name, int *error)

imFile * imFileNewRaw (const char *file_name, int *error)

Function Documentation

imFile*	imFileOpenRaw(const char *	<i>file_name,</i>
		int *	<i>error</i>
)		

Opens a RAW image file.

imFile*	imFileNewRaw(const char *	<i>file_name,</i>
		int *	<i>error</i>
)		

Creates a RAW image file.

BMP - Windows Device Independent Bitmap [File Formats]

Description

Windows Copyright Microsoft Corporation.

Internal Implementation.

Features

Data Types: Byte

Color Spaces: RGB, MAP and Binary (Gray saved as MAP)

Compressions:

NONE - no compression [default]

RLE - Run Length Encoding (only for MAP and Gray)

Only one image.

Can have an alpha channel (only for RGB)

Internally the components are always packed.

Lines arranged from top down to bottom or bottom up to top. But Handle() returns imBinFile* pointer.

Attributes:

ResolutionUnit (string) ["DPC", "DPI"]

XResolution, YResolution IM_FLOAT (1)

Comments:

Reads OS2 1.x and Windows 3, but writes Windows 3 always.

Version 4 and 5 BMPs are not supported.

GIF - Graphics Interchange Format

[File Formats]

Description

Copyright (c) 1987,1988,1989,1990 CompuServe Incorporated.
GIF is a Service Mark property of CompuServe Incorporated.
Graphics Interchange Format Programming Reference, 1990.
LZW Copyright Unisys.

Patial Internal Implementation.
Decoding and encoding code were extracted from GIFLib 1.0.
Copyright (c) 1989 Gershon Elber.

Features

Data Types: Byte

Color Spaces: MAP only, (Gray and Binary saved as MAP)

Compressions:

LZW - Lempel-Ziv & Welch [default]

Can have more than one image.

No alpha channel.

Internally the lines are arranged from top down to bottom.

Handle() returns a imBinFile* pointer.

Attributes:

ScreenHeight, ScreenWidth IM_USHORT (1) screen size [default t

Interlaced IM_INT (1 | 0) default 0

Description (string)

TransparencyIndex IM_BYTE (1)

XScreen, YScreen IM_USHORT (1) screen position

UserInput IM_BYTE (1) [1, 0]

Disposal (string) [UNDEF, LEAVE, RBACK, RPREV]

Delay IM_USHORT (1)

Iterations IM_USHORT (1) (NETSCAPE2.0 Application Extension)

Comments:

Attributes after the last image are ignored.

Reads GIF87 and GIF89, but writes GIF89 always.

Ignored attributes: Background Color Index, Pixel Aspect Ratio
Plain Text Extensions, Application Extensi

ICO - Windows Icon

[File Formats]

Description

Windows Copyright Microsoft Corporation.

Internal Implementation.

Features

Data Types: Byte

Color Spaces: RGB, MAP and Binary (Gray saved as MAP)

Compressions:

NONE - no compression [default]

Can have more than one image. But writing is limited to 5 images
and all images must have different sizes and bpp.

No alpha channel.

Internally the components are always packed.

Internally the lines are arranged from bottom up to top.

Handle() returns imBinFile* pointer.

Attributes:

none

Comments:

If the user specifies an alpha channel, the AND mask is loaded
but the file color mode never contains the IM_ALPHA flag.

Although any size and bpp can be used is recommended to use the
16x16, 32x32, 48x48, 64x64 or 96x96
2 colors, 16 colors or 256 colors

JPEG - JPEG File Interchange Format

[File Formats]

Description

ISO/IEC 10918 (1994, 1995, 1997, 1999)

<http://www.jpeg.org/>

Access to the JPEG file format uses libJPEG version 6b.

<http://www.ijg.org>

Copyright (C) 1991-1998, Thomas G. Lane
from the Independent JPEG Group.

Access to the EXIF attributes uses libEXIF version 0.5.12.

<http://sourceforge.net/projects/libexif>

Copyright (C) 2001-2003, Lutz Müller

Features

Data Types: Byte
Color Spaces: Gray, RGB, CMYK and YCbCr (Binary Saved as Gray)
Compressions:
 JPEG - ISO JPEG [default]
Only one image.
No alpha channel.
Internally the components are always packed.
Internally the lines are arranged from top down to bottom.
Handle() returns jpeg_decompress_struct* when reading, and
 jpeg_compress_struct* when writing.

Attributes:

 JPEGQuality IM_INT (1) [0-100, default 75] (write only)
 ResolutionUnit (string) ["DPC", "DPI"]
 XResolution, YResolution IM_FLOAT (1)
 Interlaced (same as Progressive) IM_INT (1 | 0) default 0
 Description (string)
 (lots of Exif tags)

Changes to libJPEG:

 jdatadst.c - fflush and ferror replaced by macros JFFLUSH and
 jinclude.h - standard JFFLUSH and JFERROR definitions, and new
 jmorecfg.h - changed definition of INT32 to JINT32 for better
 new file created: jconfig.h

Changes to libEXIF:

 new file config.h
 changed "exif-tag.c" to add new function
 changed "exif-entry.c" to improve exif_entry_initialize

Comments:

 Other APPx markers are ignored.
 No thumbnail support.

KRN - IM Kernel File Format

[File Formats]

Description

Textual format to provied a simple way to create kernel convolution images.

Internal Implementation.

Features

Data Types: Byte, Int
Color Spaces: Gray
Compressions:
 NONE - no compression [default]
Only one image.
No alpha channel.
Internally the lines are arranged from top down to bottom.
Handle() returns imBinFile* pointer.

Attributes:
 Description (string)

Comments:
The format is very simple, inspired by PNM.
It was developed because PNM does not have support for INT and
Remember that usually convolution operations use kernel size an

Format Model:
IMKERNEL
Description up to 512 characters
width height
type (0 - IM_INT, 1 - IM_FLOAT)
data...

Example:
IMKERNEL
Gradian
3 3
0
0 -1 0
0 1 0
0 0 0

LED - IUP image in LED [File Formats]

Description

Copyright Tecgraf/PUC-Rio and PETROBRAS/CENPES.

Internal Implementation.

Features

Data Types: Byte

Color Spaces: MAP only (Gray and Binary saved as MAP)

Compressions:

NONE - no compression [default]

Only one image.

No alpha channel.

Internally the lines are arranged from top down to bottom.

Handle() returns imBinFile* pointer.

Attributes:

none

Comments:

LED file must start with "LEDImage = IMAGE[".

PCX - ZSoft Picture

[File Formats]

Description

Copyright ZSoft Corporation.
ZSoft (1988) PCX Technical Reference Manual.

Internal Implementation.

Features

Data Types: Byte

Color Spaces: RGB, MAP and Binary (Gray saved as MAP)

Compressions:

NONE - no compression

RLE - Run Length Encoding [default - since uncompressed PCX is only one image.]

No alpha channel.

Internally the components are always packed.

Internally the lines are arranged from top down to bottom.

Handle() returns imBinFile* pointer.

Attributes:

ResolutionUnit (string) ["DPC", "DPI"]

XResolution, YResolution IM_FLOAT (1)

XScreen, YScreen IM USHORT (1) screen position

Comments:

Reads Versions 0-5, but writes Version 5 always.

PNG - Portable Network Graphic Format

[File Formats]

Description

Access to the PNG file format uses libPNG version 1.2.5.

<http://www.libpng.org>

Copyright (C) 2000-2002 Glenn Randers-Pehrson

Deflate compression support uses zlib version 1.2.1.

<http://www.zlib.org>

Copyright (C) 1995-2003 Jean-loup Gailly and Mark Adler

Features

Data Types: Byte and UShort
Color Spaces: Gray, RGB, MAP and Binary
Compressions:
 DEFLATE - LZ77 variation (ZIP) [default]
Only one image.
Can have an alpha channel.
Internally the components are always packed.
Internally the lines are arranged from top down to bottom.
Handle() returns png_structp

Attributes:

```
ZIPQuality IM_INT (1) [1-9, default 6] (write only)
ResolutionUnit (string) ["DPC", "DPI"]
XResolution, YResolution IM_FLOAT (1)
Interlaced (same as Progressive) IM_INT (1 | 0) default 0
Gamma IM_FLOAT (1)
WhitePoint IMFLOAT (2)
PrimaryChromaticities IMFLOAT (6)
XPosition, YPosition IM_FLOAT (1)
sRGBIntent IM_INT (1) [0: Perceptual, 1: Relative colorimetric]
TransparencyIndex IM_BYTE (1 or N)
TransparentColor IM_BYTE (3)
CalibrationName, CalibrationUnits (string)
CalibrationLimits IM_INT (2)
CalibrationEquation IM_BYTE (1) [0-Linear,1-Exponential,2-Arbi
CalibrationParam (string) [params separated by '\\n']
Title, Author, Description, Copyright, DateTime (string)
Software, Disclaimer, Warning, Source, Comment, ... (str
DateTimeModified (string) [when writing uses the current syste
ICCProfile IM_BYTE (N)
ScaleUnit (string) ["meters", "radians"]
XScale, YScale IM_FLOAT (1)
```

Comments:

Attributes after the image are ignored.
Define PNG_NO_CONSOLE_IO to avoid printf.
We define PNG_TIME_RFC1123_SUPPORTED.
Add the following files to the makefile to optimize the library
pngvcrd.c - PNG_USE_PNGVCRD
 For Intel x86 CPU and Microsoft Visual C++ compil
pnggccrd.c - PNG_USE_PNGGCCRD
 For Intel x86 CPU (Pentium-MMX or later) and GNU

PNM - Netpbm Portable Image Map [File Formats]

Description

PNM formats Copyright Jef Poskanzer

Internal Implementation.

Features

Data Types: Byte and UShort

Color Spaces: Gray, RGB and Binary

Compressions:

NONE - no compression [default]

ASCII (textual data)

Can have more than one image, but sequential access only.

No alpha channel.

Internally the components are always packed.

Internally the lines are arranged from top down to bottom.

Handle() returns imBinFile* pointer.

Attributes:

Description (string)

Comments:

In fact ASCII is an expansion...

RAS - Sun Raster File

[File Formats]

Description

Copyright Sun Corporation.

Internal Implementation.

Features

Data Types: Byte

Color Spaces: Gray, RGB, MAP and Binary

Compressions:

NONE - no compression [default]

RLE - Run Length Encoding

Only one image.

Can have an alpha channel (only for IM_RGB)

Internally the components are always packed.

Internally the lines are arranged from top down to bottom.

Handle() returns imBinFile* pointer.

Attributes:

none

SGI - Silicon Graphics Image File Format

[File Formats]

Description

SGI is a trademark of Silicon Graphics, Inc.

Internal Implementation.

Features

Data Types: Byte and UShort

Color Spaces: Gray and RGB (Binary saved as Gray, MAP with fixed Compressions:

NONE - no compression [default]

RLE - Run Length Encoding

Only one image.

Can have an alpha channel (only for IM_RGB)

Internally the components are always packed.

Internally the lines are arranged from bottom up to top.

Handle() returns imBinFile* pointer.

Attributes:

Description (string)

TGA - Truevision Graphics Adapter File [File Formats]

Description

Truevision TGA File Format Specification Version 2.0
Technical Manual Version 2.2 January, 1991
Copyright 1989, 1990, 1991 Truevision, Inc.

Internal Implementation.

Features

Supports 8 bits per component only. Data type is always Byte.

Color Spaces: Gray, RGB and MAP (Binary saved as Gray)

Compressions:

NONE - no compression [default]

RLE - Run Length Encoding

Only one image.

No alpha channel.

Internally the components are always packed.

Internally the lines are arranged from bottom up to top or from Handle() returns imBinFile* pointer.

Attributes:

XScreen, YScreen IM_USHORT (1) screen position

Title, Author, Description, JobName, Software (string)

SoftwareVersion (read only) (string)

DateTimeModified (string) [when writing uses the current system date]

Gamma IM_FLOAT (1)

TIFF - Tagged Image File Format

[File Formats]

Description

Copyright (c) 1986-1988, 1992 by Adobe Systems Incorporated.

Originally created by a group of companies, the Aldus Corporation kept the copyright until Aldus was acquired by Adobe.

TIFF Revision 6.0 Final June 3, 1992

<http://www.adobe.com/Support/TechNotes.html>

Access to the TIFF file format uses libTIFF version 3.6.1

<http://www.libtiff.org>

Copyright (c) 1988-1997 Sam Leffler

Copyright (c) 1991-1997 Silicon Graphics, Inc.

Features

Data Types: <all>
Color Spaces: Gray, RGB, CMYK, YCbCr, Lab, XYZ, Map and Binary.
Compressions:
NONE - no compression [default for IEEE Floating Point Data]
CCITTRLE - CCITT modified Huffman RLE (binary only) [default f
CCITTFAX3 - CCITT Group 3 fax (binary only)
CCITTFAX4 - CCITT Group 4 fax (binary only)
LZW - Lempel-Ziv & Welch [default]
JPEG - ISO JPEG [default for YCBQR]
NEXT - NeXT 2-bit RLE (2 bpp only)
CCITTRLEW - CCITT modified Huffman RLE with word alignment (bi
RLE - Packbits (Macintosh RLE) [default for MAP]
THUNDERSCAN - ThunderScan 4-bit RLE (only for 2 or 4 bpp)
PIXARLOG - Pixar companded 11-bit ZIP (only byte, ushort and f
DEFLATE - LZ77 variation (ZIP)
ADOBEDFLATE - Adobe LZ77 variation
SGILOG - SGI Log Luminance RLE for L and Luv (only byte, ushort
SGILOG24 - SGI Log 24-bit packed for Luv (only byte, ushort an
Can have more than one image.
Can have an alpha channel.
Components can be packed or not.
Lines arranged from top down to bottom or bottom up to top.
Handle() returns a TIFF* of libTIFF.

Attributes:

Photometric IM_USHORT (1) (when writing this will complement t
ExtraSampleInfo IM_USHORT (1) (description of alpha channel: 0
JPEGQuality IM_INT (1) [0-100, default 75] (write only)
ZIPQuality IM_INT (1) [1-9, default 6] (write only)
ResolutionUnit (string) ["DPC", "DPI"]
XResolution, YResolution IM_FLOAT (1)
Description, Author, Copyright, DateTime, DocumentName,
PageName, TargetPrinter, Make, Model, Software, HostComputer (br/>InkNames (strings separated by '0's)
InkSet IM_USHORT (1)
NumberOfInks IM_USHORT (1)
DotRange IM_USHORT (2)
TransferFunction0, TransferFunction1, TransferFunction3 IM_USH
ReferenceBlackWhite IMFLOAT (6)
WhitePoint IMFLOAT (2)
PrimaryChromaticities IMFLOAT (6)
YCbCrCoefficients IM_FLOAT (3)
YCbCrSubSampling IM_USHORT (2)

```
YCbCrPositioning IM_USHORT (1)
pageNumber IM_USHORT (2)
StoNits IM_FLOAT (1)
XPosition, YPosition IM_FLOAT (1)
SMinSampleValue, SMaxSampleValue IM_FLOAT (1)
HalftoneHints IM_USHORT (2)
SubfileType IM_INT (1)
ICCProfile IM_BYTE (N)
GeoTiePoints, GeoTransMatrix, IntergraphMatrix, GeoPixelScale,
GeoASCIIParams (string)
(other attributes can be obtained by using libTIFF directly us
```

Comments:

LogLuv is in fact $Y' + CIE(u, v)$, so we choose to convert to XYZ.
SubIFD is not handled.
Since LZW patent expired, we use the libtiff-lzw-compression-k
LZW Copyright Unisys.
libGeoTIFF can be used without XTIFF initialization. Use Handl
Must define in the makefile: JPEG_SUPPORT, ZIP_SUPPORT, PIXARL
If your system does not have the definitions u_char, u_short,
you must define BSDTYPES in the makefile when compiling libT
Our include file "port.h" simply includes "tiffcomp.h".
Changed "tiff_jpeg.c" - commented "downsampled_output = TRUE"
New file tiff_binfile.c

AVI - Windows Audio-Video Interleaved RIFF [File Formats]

Detailed Description

Description

Windows Copyright Microsoft Corporation.

Internal Implementation, Windows Only.

You must link the application with "im_avi.lib" and you must call the function **imFormatRegisterAVI** once to register the format into the IM core library.

Depends also on the VFW library (vfw32.lib). When using the "im_avi.dll" this extra library is not necessary.

If using Cygwin or MingW must link with "vfw_ms32.a" and "vfw_avi32.a".

See [**im_format_avi.h**](#)

Features

Data Types: Byte

Color Spaces: RGB, MAP and Binary (Gray saved as MAP)

Compressions (installed in Windows XP by default):

NONE	- no compression [default]
RLE	- Microsoft RLE (8bpp only)
CINEPACK	- Cinepak Codec by Radius
MSVC	- Microsoft Video 1 (old)
M261	- Microsoft H.261 Video Codec
M263	- Microsoft H.263 Video Codec
I420	- Intel 4:2:0 Video Codec (same as M263)
IV32	- Intel Indeo Video Codec 3.2 (old)
IV41	- Intel Indeo Video Codec 4.5 (old)
IV50	- Intel Indeo Video 5.1
IYUV	- Intel IYUV Codec
MPG4	- Microsoft MPEG-4 Video Codec V1 (not MPEG-4 compliant)
MP42	- Microsoft MPEG-4 Video Codec V2 (not MPEG-4 compliant)
CUSTOM	- (show compression dialog)
DIVX	- DivX 5.0.4 Codec (DivX must be installed) (others, must be the 4 characters of the fourfcc code)

Can have more than one image.

Can have an alpha channel (only for RGB)

Internally the components are always packed.

Lines arranged from top down to bottom or bottom up to top. But Handle() returns PAVIFILE.

Attributes:

FPS IM_FLOAT (1) (should set when writing, default 15)
AVIQuality IM_INT (1) [1-10000, default -1] (write only)
KeyFrameRate IM_INT (1) (write only) [key frame frequency, if
DataRate IM_INT (1) (write only) [kilobits/second, default 240]

Comments:

Reads only the first video stream. Other streams are ignored.
All the images have the same size, you must call imFileReadImage at least once.

For codecs comparison and download go to:

<http://graphics.lcs.mit.edu/~tbuehler/video/codecs/>
<http://www.fourcc.org>

Functions

void **imFormatRegisterAVI** (void)

Function Documentation

void imFormatRegisterAVI(void)

Register the AVI Format

JP2 - JPEG-2000 JP2 File Format

[File Formats]

Detailed Description

Description

ISO/IEC 15444 (2000, 2003)

<http://www.jpeg.org/>

You must link the application with "im_jp2.lib" and you must call the function **imFormatRegisterJP2** once to register the format into the IM core library.

Access to the JPEG2000 file format uses libJasper version 1.700.5.

<http://www.ece.uvic.ca/~mdadams/jasper>

Copyright (c) 2001-2003 Michael David Adams.

See **im_format_jp2.h**

Features

Data Types: Byte and USHORT

Color Spaces: Binary, Gray, RGB, YCbCr, Lab and XYZ

Compressions:

JPEG-2000 - ISO JPEG 2000 [default]

Only one image.

Can have an alpha channel.

Internally the components are always unpacked.

Internally the lines are arranged from top down to bottom.

Handle() returns jas_image_t*

Attributes:

CompressionRatio IM_FLOAT (1) [write only, example: Ratio=7 ju

Comments:

We read code stream syntax and JP2, but write as JP2 always.

Used definitions EXCLUDE_JPG_SUPPORT, EXCLUDE_MIF_SUPPORT,
EXCLUDE_PNM_SUPPORT, EXCLUDE_RAS_SUPPORT,
EXCLUDE_BMP_SUPPORT, EXCLUDE_PGX_SUPPORT

Changed jas_config.h to match our needs.

New file jas_binfile.c

Changed jas_stream.c to export 2 functions.

Changed jp2_dec.c and jpc_cs.c to remove unit and ulong defini

Functions

void **imFormatRegisterJP2** (void)

Function Documentation

```
void imFormatRegisterJP2( void )
```

Register the JP2 Format

WMV - Windows Media Video Format [File Formats]

Detailed Description

Description

Advanced Systems Format (ASF)
Windows Copyright Microsoft Corporation.

Internal Implementation, Windows Only.

You must link the application with "im_wmv.lib" and you must call the function **imFormatRegisterWMV** once to register the format into the IM core library.

Depends also on the WMF SDK (wmvcore.lib). When using the "im_wmv.dll" this extra library is not necessary.

The application users should have the WMV codec 9 installed:

<http://www.microsoft.com/windows/windowsmedia/format/codecdownload>.

You must agree with the WMF SDK EULA to use the SDK.

<http://wmlicense.smdisp.net/v9sdk/>

For more information:

<http://www.microsoft.com/windows/windowsmedia/9series/sdk.aspx>

[http://msdn.microsoft.com/library/en-](http://msdn.microsoft.com/library/en-us/wmform/htm/introducingwindowsmediaformat.asp)

[us/wmform/htm/introducingwindowsmediaformat.asp](http://msdn.microsoft.com/library/en-us/wmform/htm/introducingwindowsmediaformat.asp)

See **im_format_wmv.h**

Features

Data Types: Byte

Color Spaces: RGB and MAP (Gray and Binary saved as MAP)

Compressions (installed in Windows XP by default):

NONE	- no compression
MPEG-4v3	- Windows Media MPEG-4 Video V3
MPEG-4v1	- ISO MPEG-4 Video V1
WMV7	- Windows Media Video V7
WMV7Screen	- Windows Media Screen V7
WMV8	- Windows Media Video V8
WMV9Screen	- Windows Media Video 9 Screen
WMV9	- Windows Media Video 9 [default]
Unknown	- Others

Can have more than one image.

Can have an alpha channel (only for RGB) ?

Internally the components are always packed.

Lines arranged from top down to bottom or bottom up to top.

Handle() returns IWMSyncReader* when reading, IWMWriter* when wr

Attributes:

FPS IM_FLOAT (1) (should set when writing, default 15)
WMFQuality IM_INT (1) [0-100, default 50] (write only)
MaxKeyFrameTime IM_INT (1) (write only) [maximum key frame int
DataRate IM_INT (1) (write only) [kilobits/second, default 240
VBR IM_INT (1) [0, 1] (write only) [0 - Constant Bit Rate (def
(and several others from the file-level attributes) For ex:
Title, Author, Copyright, Description (string)
Duration IM_INT [100-nanosecond units]
Seekable, HasAudio, HasVideo, Is_Protected, Is_Trusted, IsVB
NumberOfFrames IM_INT (1)

Comments:

IMPORTANT - The "image_count" and the "FPS" attribute may not we try to estimate from the duration and from the average ti We do not handle DRM protected files (Digital Rights Management Reads only the first video stream. Other streams are ignored. All the images have the same size, you must call imFileReadIma at least once.

For optimal random reading, the file should be indexed previous If not indexed by frame, random positioning may not be precise Sequencial reading will always be precise.

When writing we use a custom profile and time indexing only.

We do not support multipass encoding.

Since the driver uses COM, CoInitialize(NULL) and CoUninitiali

Functions

void **imFormatRegisterWMV** (void)

Function Documentation

```
void imFormatRegisterWMV( void )
```

Register the WMF Format

File Format SDK

[Image Storage]

Detailed Description

All the file formats are based on theses structures. Use them to create new file formats.

The LineBuffer functions will help transfer image from format buffer to application buffer and vice-versa.

See [im_file.h](#)

Data Structures

struct [_imFile](#)

Image File Format Base (SDK Use Only). [More...](#)

class [imFormat](#)

Image File Format Driver (SDK Use Only). [More...](#)

Typedefs

```
typedef imFormat *(* imFormatFunc )()
```

Functions

```
int imFileLineBufferCount (imFile *ifile)
void imFileLineBufferInc (imFile *ifile, int *row, int *plane)
void imFileLineBufferRead (imFile *ifile, void *data, int line, int plane)
void imFileLineBufferWrite (imFile *ifile, const void *data, int line, int
    plane)
int imFileLineSizeAligned (int width, int bpp, int align)
void imFormatRegister (imFormatFunc format_init)
```

Typedef Documentation

typedef imFormat*(* imFormatFunc)()

Format function initialization definition.

Function Documentation

```
int imFileLineBufferCount( imFile * ifile )
```

Number of lines to be accessed.

```
void imFileLineBufferInc( imFile * ifile,  
                          int * row,  
                          int * plane  
)
```

Increments the row and plane counters.

```
void imFileLineBufferRead( imFile * ifile,  
                           void * data,  
                           int line,  
                           int plane  
)
```

Converts from FILE color mode to USER color mode.

```
void imFileLineBufferWrite( imFile * ifile,  
                           const void * data,  
                           int line,  
                           int plane  
)
```

Converts from USER color mode to FILE color mode.

```
int imFileLineSizeAligned( int width,  
                           int bpp,  
                           int align  
                         )
```

Utility to calculate the line size in byte with a specified alignment.
"align" can be 1, 2 or 4.

```
void imFormatRegister( imFormatFunc format_init )
```

Register a format driver.

Image Processing

Detailed Description

Several image processing functions based on the **imImage** structure.

You must link the application with "im_process.lib/.a/.so".

Some complex operations use the **Counter**.

There is no check on the input/output image properties, check each function documentation before using it.

See **im_process.h**

Modules

group [**Image Statistics Calculations**](#)
group [**Image Analysis**](#)
group [**Domain Transform Operations**](#)
group [**Image Resize**](#)
group [**Geometric Operations**](#)
group [**Morphology Operations for Gray Images**](#)
group [**Morphology Operations for Binary Images**](#)
group [**Rank Convolution Operations**](#)
group [**Convolution Operations**](#)
group [**Arithmetic Operations**](#)
group [**Additional Image Quantization Operations**](#)
group [**Histogram Based Operations**](#)
group [**Color Processing Operations**](#)
group [**Logical Arithmetic Operations**](#)
group [**Synthetic Image Render**](#)
group [**Tone Gamut Operations**](#)
group [**Threshold Operations**](#)
group [**Special Effects**](#)

Synthetic Image Render

[Image Processing]

Detailed Description

Renders some 2D mathematical functions as images. All the functions operates in place and supports all data types except IM_COMPLEX.

See [im_process_pon.h](#)

Typedefs

```
typedef float(* imRenderFunc )(int x, int y, int d, float *param)  
typedef float(* imRenderCondFunc )(int x, int y, int d, int *cond, float  
*param)
```

Functions

```
int imProcessRenderOp (imImage *image, imRenderFunc  
    render_func, char *render_name, float *param, int plus)  
int imProcessRenderCondOp (imImage *image, imRenderCondFunc  
    render_func, char *render_name, float *param)  
int imProcessRenderAddSpeckleNoise (const imImage *src_image,  
    imImage *dst_image, float percent)  
int imProcessRenderAddGaussianNoise (const imImage *src_image,  
    imImage *dst_image, float mean, float stddev)  
int imProcessRenderAddUniformNoise (const imImage *src_image,  
    imImage *dst_image, float mean, float stddev)  
int imProcessRenderRandomNoise (imImage *image)  
int imProcessRenderConstant (imImage *image, float *value)  
int imProcessRenderWheel (imImage *image, int int_radius, int  
    ext_radius)  
int imProcessRenderCone (imImage *image, int radius)  
int imProcessRenderTent (imImage *image, int width, int height)  
int imProcessRenderRamp (imImage *image, int start, int end, int dir)  
int imProcessRenderBox (imImage *image, int width, int height)  
int imProcessRenderSinc (imImage *image, float xperiod, float  
    yperiod)  
int imProcessRenderGaussian (imImage *image, float stddev)  
int imProcessRenderLapOfGaussian (imImage *image, float stddev)  
int imProcessRenderCosine (imImage *image, float xperiod, float  
    yperiod)  
int imProcessRenderGrid (imImage *image, int x_space, int y_space)  
int imProcessRenderChessboard (imImage *image, int x_space, int  
    y_space)
```

Typedef Documentation

```
typedef float(* imRenderFunc)(int x, int y, int d, float *param)
```

Render Function.

```
typedef float(* imRenderCondFunc)(int x, int y, int d, int *cond, float
```

Render Conditional Function.

Function Documentation

int imProcessRenderOp (imImage *	<i>image,</i>
	imRenderFunc	<i>render_func,</i>
	char *	<i>render_name,</i>
	float *	<i>param,</i>
	int	<i>plus</i>
)		

Render a synthetic image using a render function.

plus will make the render be added to the current image data, or else all data will be replaced. All the render functions use this or the conditional function.

Returns zero if the counter aborted.

int imProcessRenderCondOp (imImage *	<i>image,</i>
	imRenderCondFunc	<i>render_func,</i>
	char *	<i>render_name,</i>
	float *	<i>param</i>
)		

Render a sintetic image using a conditional render function.

Data will be rendered only if the condional param is true.

Returns zero if the counter aborted.

int imProcessRenderAddSpeckleNoise (const imImage *	<i>src_image</i>
	imImage *	<i>dst_image</i>
	float	<i>percent</i>
)		

Render speckle noise on existing data. Can be done in place.

```
int imProcessRenderAddGaussianNoise ( const imlImage * src_image,
                                      imlImage * dst_image,
                                      float mean,
                                      float stddev
                                    )
```

Render gaussian noise on existing data. Can be done in place.

```
int imProcessRenderAddUniformNoise ( const imlImage * src_image,
                                      imlImage * dst_image,
                                      float mean,
                                      float stddev
                                    )
```

Render uniform noise on existing data. Can be done in place.

```
int imProcessRenderRandomNoise ( imlImage * image )
```

Render random noise.

```
int imProcessRenderConstant ( imlImage * image,
                             float * value
                           )
```

Render a constant. The number of values must match the depth of the image.

```
int imProcessRenderWheel ( imlImage * image,
```

	int	<i>int_radius,</i>
	int	<i>ext_radius</i>
)	

Render a centered wheel.

int imProcessRenderCone(imImage *	<i>image,</i>
	int	<i>radius</i>
)	

Render a centered cone.

int imProcessRenderTent(imImage *	<i>image,</i>
	int	<i>width,</i>
	int	<i>height</i>
)	

Render a centered tent.

int imProcessRenderRamp(imImage *	<i>image,</i>
	int	<i>start,</i>
	int	<i>end,</i>
	int	<i>dir</i>
)	

Render a ramp. Direction can be vertical (1) or horizontal (0).

int imProcessRenderBox(imImage *	<i>image,</i>
	int	<i>width,</i>
	int	<i>height</i>
)	

Render a centered box.

int imProcessRenderSinc (imImage * image,	
	float	xperiod,
	float	yperiod
)		

Render a centered sinc.

int imProcessRenderGaussian (imImage * image,	
	float	stddev
)		

Render a centered gaussian.

int imProcessRenderLapOfGaussian (imImage * image,	
	float	stddev
)		

Render the laplacian of a centered gaussian.

int imProcessRenderCosine (imImage * image,	
	float	xperiod,
	float	yperiod
)		

Render a centered cosine.

int imProcessRenderGrid (imImage * image,	
----------------------------------	-------------------------	--

	int	<i>x_space,</i>
	int	<i>y_space</i>
)		

Render a centered grid.

int imProcessRenderChessboard (imImage *	<i>image,</i>
	int	<i>x_space,</i>
	int	<i>y_space</i>
)		

Render a centered chessboard.

Image Resize

[Image Processing]

Detailed Description

Operations to change the image size.

See [im_process_loc.h](#)

Functions

```
int imProcessReduce (const imlImage *src_image, imlImage  
*dst_image, int order)  
int imProcessResize (const imlImage *src_image, imlImage  
*dst_image, int order)  
void imProcessReduceBy4 (const imlImage *src_image, imlImage  
*dst_image)  
void imProcessCrop (const imlImage *src_image, imlImage  
*dst_image, int xmin, int ymin)  
void imProcessAddMargins (const imlImage *src_image, imlImage  
*dst_image, int xmin, int ymin)
```

Function Documentation

```
int imProcessReduce( const imlImage * src_image,  
                     imlImage * dst_image,  
                     int order  
)
```

Only reduce the image size using the given decimation order.

Supported interpolation orders:

- 0 - zero order (mean)
- 1 - first order (bilinear decimation) Images must be of the same type.

Returns zero if the counter aborted.

```
int imProcessResize( const imlImage * src_image,  
                     imlImage * dst_image,  
                     int order  
)
```

Change the image size using the given interpolation order.

Supported interpolation orders:

- 0 - zero order (near neighborhood)
- 1 - first order (bilinear interpolation)
- 3 - third order (bicubic interpolation) Images must be of the same type.

Returns zero if the counter aborted.

```
void imProcessReduceBy4( const imlImage * src_image,  
                        imlImage * dst_image
```

)

Reduze the image area by 4 (w/2,h/2).

Images must be of the same type. Destiny image size must be source image width/2, height/2.

void imProcessCrop (const imImage * src_image,	
	imImage * dst_image,	
	int xmin,	
	int ymin	
)		

Reduze the image size by removing pixels.

Images must be of the same type. Destiny image size must be smaller than source image width-xmin, height-ymin.

void imProcessAddMargins (const imImage * src_image,	
	imImage * dst_image,	
	int xmin,	
	int ymin	
)		

Increase the image size by adding pixels with zero value.

Images must be of the same type. Destiny image size must be greater than source image width+xmin, height+ymin.

Geometric Operations

[Image Processing]

Detailed Description

Operations to change the shape of the image.

See [im_process_loc.h](#)

Functions

```
void imProcessCalcRotateSize (int width, int height, int *new_width, int  
    *new_height, double cos0, double sin0)  
int imProcessRotate (const imImage *src_image, imImage  
    *dst_image, double cos0, double sin0, int order)  
void imProcessRotate90 (const imImage *src_image, imImage  
    *dst_image, int dir)  
void imProcessRotate180 (const imImage *src_image, imImage  
    *dst_image)  
void imProcessMirror (const imImage *src_image, imImage  
    *dst_image)  
void imProcessFlip (const imImage *src_image, imImage *dst_image)  
int imProcessRadial (const imImage *src_image, imImage  
    *dst_image, float k1, int order)
```

Function Documentation

```
void imProcessCalcRotateSize( int width,  
                             int height,  
                             int * new_width,  
                             int * new_height,  
                             double cos0,  
                             double sin0  
 )
```

Calculates the size of the new image after rotation.

```
int imProcessRotate( const imlImage * src_image,  
                     imlImage * dst_image,  
                     double cos0,  
                     double sin0,  
                     int order  
 )
```

Rotates the image using the given interpolation order (see imProcessResize).

Images must be of the same type. The destiny size can be calculated using [imProcessCalcRotateSize](#).

Returns zero if the counter aborted.

```
void imProcessRotate90( const imlImage * src_image,  
                      imlImage * dst_image,  
                      int dir  
 )
```

Rotate the image in 90 degrees counterclockwise or clockwise. Swap columns by lines.

Images must be of the same type. Destiny width and height must be source height and width.

void imProcessRotate180 (const imImage * src_image,	
	imImage * dst_image	
)		

Rotate the image in 180 degrees. Swap columns and swap lines.

Images must be of the same type and size.

void imProcessMirror (const imImage * src_image,	
	imImage * dst_image	
)		

Mirrors the image in a horizontal flip. Swap columns.

Images must be of the same type and size.

void imProcessFlip (const imImage * src_image,	
	imImage * dst_image	
)		

Apply a vertical flip. Swap lines.

Images must be of the same type and size.

int imProcessRadial (const imImage * src_image,	
	imImage * dst_image,	
	float k1,	
	int order	
)		

Apply a radial distortion using the given interpolation order (see imProcessResize).
Images must be of the same type and size. Returns zero if the counter aborted.

Additional Image Quantization Operations

[Image Processing]

Detailed Description

Additionally operations to the **imConvertColorSpace** function.

See [im_process_pon.h](#)

Functions

```
void imProcessQuantizeRGBUniform (const imImage *src_image,  
                                imImage *dst_image, int dither)  
void imProcessQuantizeGrayUniform (const imImage *src_image,  
                                    imImage *dst_image, int grays)
```

Function Documentation

void imProcessQuantizeRGBUniform (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>dither</i>
)		

Converts a RGB image to a MAP image using uniform quantization with an optional 8x8 ordered dither. The RGB image must have data type IM_BYTE.

void imProcessQuantizeGrayUniform (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>grays</i>
)		

Quantizes a gray scale image in less than 256 grays using uniform quantization.

Both images must be IM_BYTE/IM_GRAY. Can be done in place.

Color Processing Operations

[Image Processing]

Detailed Description

Operations to change the color components configuration.

See [im_process_pon.h](#)

Functions

```
void imProcessSplitYChroma (const imImage *src_image, imImage
    *y_image, imImage *chroma_image)
void imProcessSplitHSI (const imImage *src_image, imImage
    *h_image, imImage *s_image, imImage *i_image)
void imProcessMergeHSI (const imImage *h_image, const imImage
    *s_image, const imImage *i_image3, imImage *dst_image)
void imProcessSplitComponents (const imImage *src_image,
    imImage **dst_image)
void imProcessMergeComponents (const imImage **src_image_list,
    imImage *dst_image)
void imProcessNormalizeComponents (const imImage *src_image,
    imImage *dst_image)
void imProcessReplaceColor (const imImage *src_image, imImage
    *dst_image, float *src_color, float *dst_color)
```

Function Documentation

```
void imProcessSplitYChroma( const imlImage * src_image,  
                           imlImage * y_image,  
                           imlImage * chroma_image  
                         )
```

Split a RGB image into luma and chroma.

Chroma is calculated as R-Y,G-Y,B-Y. Source image must be IM_RGB/IM_BYTE.

Luma image is IM_GRAY/IM_BYTE and chroma is IM_RGB/IM_BYTE.
Source and destiny have the same size.

```
void imProcessSplitHSI( const imlImage * src_image,  
                       imlImage * h_image,  
                       imlImage * s_image,  
                       imlImage * i_image  
                     )
```

Split a RGB image into HSI planes.

Source image must be IM_RGB/IM_BYTE. Destiny images are all IM_GRAY/IM_BYTE.

Source and destiny have the same size. See [HSI Color Coordinate System Conversions](#).

```
void imProcessMergeHSI( const imlImage * h_image,  
                       const imlImage * s_image,  
                       const imlImage * i_image3,  
                       imlImage * dst_image  
                     )
```

Merge HSI planes into a RGB image.

Source images must be IM_GRAY/IM_BYTE. Destiny image is all IM_RGB/IM_BYTE.

Source and destiny have the same size. See [HSI Color Coordinate System Conversions](#).

void imProcessSplitComponents(const imImage * src_image,	
	imImage ** dst_image	
)		

Split a multicomponent image into separate components.

Destiny images must be IM_GRAY. Size and data types must be all the same.

The number of destiny images must match the depth of the source image.

void imProcessMergeComponents(const imImage ** src_image_list,	
	imImage * dst_image	
)		

Merges separate components into a multicomponent image.

Source images must be IM_GRAY. Size and data types must be all the same.

The number of source images must match the depth of the destiny image.

void imProcessNormalizeComponents(const imImage * src_image,	
	imImage * dst_image	
)		

Normalize the color components by their sum. Example: $c1 = c1/(c1+c2+c3)$.

Destiny image must be IM_FLOAT.

```
void imProcessReplaceColor( const imImage * src_image,  
                           imImage * dst_image,  
                           float * src_color,  
                           float * dst_color  
                         )
```

Replaces the source color by the destiny color.

The color will be type casted to the image data type.

The colors must have the same number of components of the images.

Supports all color spaces and all data types except IM_COMPLEX.

Histogram Based Operations

[Image Processing]

Detailed Description

See [im_process_pon.h](#)

Functions

```
void imProcessExpandHistogram (const imImage *src_image,  
    imImage *dst_image, float percent)  
void imProcessEqualizeHistogram (const imImage *src_image,  
    imImage *dst_image)
```

Function Documentation

void imProcessExpandHistogram(const imlImage * src_image,
	imlImage * dst_image,
	float percent
)	

Performs an histogram expansion.

Percentage defines an amount of pixels to include at start and end. If its is zero only empty counts of the histogram will be considered.

Images must be IM_BYTE/(IM_RGB or IM_GRAY). Can be done in place.

void imProcessEqualizeHistogram(const imlImage * src_image,
	imlImage * dst_image
)	

Performs an histogram equalization.

Images must be IM_BYTE/(IM_RGB or IM_GRAY). Can be done in place.

Threshold Operations

[Image Processing]

Detailed Description

Operations that converts a usually IM_GRAY/IM_BYTE image into a IM_BINARY image using several threshold techniques.

See [im_process_pon.h](#)

Functions

```
int imProcessRangeContrastThreshold (const imImage  
    *src_image, imImage *dst_image, int kernel_size, int min_range)  
int imProcessLocalMaxThreshold (const imImage *src_image,  
    imImage *dst_image, int kernel_size, int min_thres)  
void imProcessThreshold (const imImage *src_image, imImage  
    *dst_image, int level, int value)  
void imProcessThresholdByDiff (const imImage *src_image1, const  
    imImage *src_image2, imImage *dst_image)  
void imProcessHysteresisThreshold (const imImage *src_image,  
    imImage *dst_image, int low_thres, int high_thres)  
void imProcessHysteresisThresEstimate (const imImage  
    *src_image, int *low_thres, int *high_thres)  
int imProcessUniformErrThreshold (const imImage *src_image,  
    imImage *dst_image)  
void imProcessDifusionErrThreshold (const imImage *src_image,  
    imImage *dst_image, int level)  
int imProcessPercentThreshold (const imImage *src_image,  
    imImage *dst_image, float percent)  
int imProcessOtsuThreshold (const imImage *src_image, imImage  
    *dst_image)  
int imProcessMinMaxThreshold (const imImage *src_image,  
    imImage *dst_image)  
void imProcessLocaMaxThresEstimate (const imImage *src_image,  
    int *thres)  
void imProcessSliceThreshold (const imImage *src_image, imImage  
    *dst_image, int start_level, int end_level)
```

Function Documentation

int imProcessRangeContrastThreshold(const imImage *	<i>src_image</i>
	imImage *	<i>dst_image</i>
	int	<i>kernel_size</i>
	int	<i>min_range</i>
)		

Threshold using a rank convolution with a range contrast function.
Supports all integer IM_GRAY images as source, and IM_BINARY as destiny.
Local variable threshold by the method of Bernsen.
Extracted from XITE, Copyright 1991, Blab, UiO
<http://www.ifi.uio.no/~blab/Software/Xite/>

Reference:

Bernsen, J: "Dynamic thresholding of grey-level images"
Proc. of the 8th ICPR, Paris, Oct 1986, 1251-1255.

Author: Oivind Due Trier

Returns zero if the counter aborted.

int imProcessLocalMaxThreshold(const imImage *	<i>src_image</i> ,
	imImage *	<i>dst_image</i> ,
	int	<i>kernel_size</i> ,
	int	<i>min_threshold</i>
)		

Threshold using a rank convolution with a local max function.
Returns zero if the counter aborted.
Supports all integer IM_GRAY images as source, and IM_BINARY as destiny.

void imProcessThreshold (const imImage * src_image,	
	imImage * dst_image,	
	int level,	
	int value	
)		

Apply a manual threshold.

$$\text{threshold} = \text{a} \leq \text{level} ? 0 : \text{value}$$

Normal value is 1 but another common value is 255. Can be done in place for IM_BYTE source.

Supports all integer IM_GRAY images as source, and IM_BINARY as destiny.

void imProcessThresholdByDiff (const imImage * src_image1,	
	const imImage * src_image2,	
	imImage * dst_image	
)		

Apply a threshold by the difference of two images.

$$\text{threshold} = \text{a}_1 \leq \text{a}_2 ? 0 : 1$$

Can be done in place.

void imProcessHysteresisThreshold (const imImage * src_image,	
	imImage * dst_image,	
	int low_thres,	
	int high_thres	
)		

Apply a threshold by the Hysteresis method.

Hysteresis thersholding of edge pixels. Starting at pixels with a value greater than the HIGH threshold, trace a connected sequence of pixels that have a value greater than the LOW threhsold.

Note: could not find the original source code author name.

void imProcessHysteresisThresEstimate(const imlImage * src_image,	
	int * low_threshold,	
	int * high_threshold,	
)	

Estimates hysteresis low and high threshold levels.

int imProcessUniformErrThreshold(const imlImage * src_image,	
	imlImage * dst_image	
)	

Calculates the threshold level for manual threshold using an uniform error approach.

Extracted from XITE, Copyright 1991, Blab, UiO

<http://www.ifi.uio.no/~blab/Software/Xite/>

Reference:

S. M. Dunn & D. Harwood & L. S. Davis:
 "Local Estimation of the Uniform Error Threshold"
 IEEE Trans. on PAMI, Vol PAMI-6, No 6, Nov 1984.

Comments: It only works well on images with large objects.

Author: Olav Borgli, BLAB, ifi, UiO

Image processing lab, Department of Informatics, University of Oslo

Returns the used level.

void imProcessDifusionErrThreshold(const imlImage * src_image,	
	imlImage * dst_image,	
	int level	
)	

Apply a dithering on each image channel by using a diffusion error method.

It can be applied on any IM_BYTE images. It will "threshold" each

channel individually, so source and destiny must be of the same depth.

int imProcessPercentThreshold (const imlImage * src_image,
	imlImage * dst_image,
	float percent
)	

Calculates the threshold level for manual threshold using a percentage of pixels that should stay below the threshold.

Returns the used level.

int imProcessOtsuThreshold (const imlImage * src_image,
	imlImage * dst_image
)	

Calculates the threshold level for manual threshold using the Otsu approach.

Returns the used level.

Original implementation by Flavio Szenberg.

int imProcessMinMaxThreshold (const imlImage * src_image,
	imlImage * dst_image
)	

Calculates the threshold level for manual threshold using (max-min)/2.
Returns the used level.

Supports all integer IM_GRAY images as source, and IM_BINARY as destiny.

void imProcessLocaMaxThresEstimate (const imlImage * src_image,
	int * thres
)	

Estimates Local Max threshold level for IM_BYTEx images.

void imProcessSliceThreshold(const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>start_level,</i>
	int	<i>end_level</i>
)		

Apply a manual threshold using an interval.

threshold = start_level <= a <= end_level ? 1: 0

Normal value is 1 but another common value is 255. Can be done in place for IM_BYTEx source.

Supports all integer IM_GRAY images as source, and IM_BINARY as destiny.

Arithmetic Operations

[Image Processing]

Detailed Description

Simple math operations for images.

See [im_process_pon.h](#)

Enumerations

```
enum imUnaryOp {
    IM_UN_EQL, IM_UN_ABS, IM_UN_LESS, IM_UN_INC,
    IM_UN_INV, IM_UN_SQR, IM_UN_SQRT, IM_UN_LOG,
    IM_UN_EXP, IM_UN_SIN, IM_UN_COS, IM_UN_CONJ,
    IM_UN_CPNORM
}
enum imBinaryOp {
    IM_BIN_ADD, IM_BIN_SUB, IM_BIN_MUL, IM_BIN_DIV,
    IM_BIN_DIFF, IM_BIN_POW, IM_BIN_MIN, IM_BIN_MAX
}
```

Functions

```
void imProcessUnArithmeticOp (const imlImage *src_image,  
    imlImage *dst_image, int op)  
void imProcessArithmeticOp (const imlImage *src_image1, const  
    imlImage *src_image2, imlImage *dst_image, int op)  
void imProcessArithmeticConstOp (const imlImage *src_image, float  
    src_const, imlImage *dst_image, int op)  
void imProcessBlend (const imlImage *src_image1, imlImage  
    *src_image2, imlImage *dst_image, float alpha)  
void imProcessSplitComplex (const imlImage *src_image, imlImage  
    *dst_image1, imlImage *dst_image2, int polar)  
void imProcessMergeComplex (const imlImage *src_image1, const  
    imlImage *src_image2, imlImage *dst_image, int polar)  
void imProcessMultipleMean (const imlImage **src_image_list, int  
    src_image_count, imlImage *dst_image)  
void imProcessMultipleStdDev (const imlImage **src_image_list, int  
    src_image_count, const imlImage *mean_image, imlImage  
    *dst_image)  
int imProcessAutoCovariance (const imlImage *src_image, const  
    imlImage *mean_image, imlImage *dst_image)  
void imProcessMultiplyConj (const imlImage *src_image1, const  
    imlImage *src_image2, imlImage *dst_image)
```

Enumeration Type Documentation

enum imUnaryOp

Unary Arithmetic Operations. Inverse and log may lead to math exception.

Enumeration values:

<i>IM_UN_EQL</i>	equal = a
<i>IM_UN_ABS</i>	abssolute = a
<i>IM_UN_LESS</i>	less = -a
<i>IM_UN_INC</i>	increment += a
<i>IM_UN_INV</i>	invert = 1/a (#)
<i>IM_UN_SQR</i>	square = a*a
<i>IM_UN_SQRT</i>	square root = a^(1/2)
<i>IM_UN_LOG</i>	natural logarithm = ln(a) (#)
<i>IM_UN_EXP</i>	exponential = exp(a)
<i>IM_UN_SIN</i>	sine = sin(a)
<i>IM_UN_COS</i>	cosine = cos(a)
<i>IM_UN_CONJ</i>	complex conjugate = ar - ai*i
<i>IM_UN_CPNORM</i>	complex normalization by magnitude = a / cpxmag(a)

```
00029      {
00030      IM_UN_EQL,    /**< equal           =     a      */
00031      IM_UN_ABS,   /**< abssolute       =     |a|      */
00032      IM_UN_LESS,  /**< less            =     -a      */
00033      IM_UN_INC,   /**< increment        +=     a      */
00034      IM_UN_INV,   /**< invert           =     1/a      (#) */
00035      IM_UN_SQR,   /**< square           =     a*a      */
00036      IM_UN_SQRT,  /**< square root      =     a^(1/2) */
00037      IM_UN_LOG,   /**< natural logarithm =   ln(a)      (#) */
00038      IM_UN_EXP,   /**< exponential      =   exp(a)      */
00039      IM_UN_SIN,   /**< sine             =   sin(a)      */
00040      IM_UN_COS,   /**< cosine           =   cos(a)      */
00041      IM_UN_CONJ,  /**< complex conjugate =   ar - ai*i
```

```
00042     IM_UN_CPNORM /**< complex normalization by magnitude = a ,  
00043 };
```

enum imBinaryOp

Binary Arithmetic Operations. Inverse and log may lead to math exceptions.

Enumeration values:

IM_BIN_ADD add = a+b
IM_BIN_SUB subtract = a-b
IM_BIN_MUL multiply = a*b
IM_BIN_DIV divide = a/b (#)
IM_BIN_DIFF difference = |a-b|
IM_BIN_POW power = a^b
IM_BIN_MIN minimum = (a < b)? a: b
IM_BIN_MAX maximum = (a > b)? a: b

```
00053             {  
00054     IM_BIN_ADD,    /**< add      = a+b      */  
00055     IM_BIN_SUB,   /**< subtract = a-b      */  
00056     IM_BIN_MUL,   /**< multiply = a*b      */  
00057     IM_BIN_DIV,   /**< divide    = a/b      (#) */  
00058     IM_BIN_DIFF,  /**< difference = |a-b| */  
00059     IM_BIN_POW,   /**< power     = a^b      */  
00060     IM_BIN_MIN,   /**< minimum   = (a < b)? a: b */  
00061     IM_BIN_MAX,   /**< maximum   = (a > b)? a: b */  
00062 };
```

Function Documentation

```
void imProcessUnArithmeticOp( const imlImage * src_image,
                               imlImage * dst_image,
                               int op
)
```

Apply an arithmetic unary operation.

Can be done in place, images must match size, does not need to match type.

```
void imProcessArithmeticOp( const imlImage * src_image1,
                            const imlImage * src_image2,
                            imlImage * dst_image,
                            int op
)
```

Apply a binary arithmetic operation.

Can be done in place, images must match size.

Source images must match type, destiny image can be several types depending on source:

- byte -> byte, ushort, int, float
- ushort -> ushort, int, float
- int -> int, float
- float -> float
- complex -> complex One exception is that you can combine complex with float resulting complex.

```
void imProcessArithmeticConstOp( const imlImage * src_image,
                                 float src_const,
```

	imlImage *	<i>dst_image,</i>
	int	<i>op</i>
)		

Apply a binary arithmetic operation with a constant value.

Can be done in place, images must match size.

Destiny image can be several types depending on source:

- byte -> byte, ushort, int, float
- ushort -> byte, ushort, int, float
- int -> byte, ushort, int, float
- float -> float
- complex -> complex The constant value is type casted to an appropriate type before the operation.

void imProcessBlend (const imlImage *	<i>src_image1,</i>
	imlImage *	<i>src_image2,</i>
	imlImage *	<i>dst_image,</i>
	float	<i>alpha</i>
)		

Blend two images using an alpha value = [a * alpha + b * (1 - alpha)].

Can be done in place, images must match size and type.

void imProcessSplitComplex (const imlImage *	<i>src_image,</i>
	imlImage *	<i>dst_image1,</i>
	imlImage *	<i>dst_image2,</i>
	int	<i>polar</i>
)		

Split a complex image into two images with real and imaginary parts or magnitude and phase parts (polar = 1).

Source image must be IM_COMPLEX, destiny images must be

IM_FLOAT.

void imProcessMergeComplex(const imlImage * src_image1,
	const imlImage * src_image2,
	imlImage * dst_image,
	int polar
)	

Merges two images as the real and imaginary parts of a complex image,

or as magnitude and phase parts (polar = 1).

Source images must be IM_FLOAT, destiny image must be IM_COMPLEX.

void imProcessMultipleMean(const imlImage ** src_image_list,
	int src_image_count,
	imlImage * dst_image
)	

Calculates the mean of multiple images.

Images must match size and type.

void imProcessMultipleStdDev(const imlImage ** src_image_list,
	int src_image_count,
	const imlImage * mean_image,
	imlImage * dst_image
)	

Calculates the standard deviation of multiple images.

Images must match size and type.

int imProcessAutoCovariance (const imlImage * src_image,
	const imlImage * mean_image,
	imlImage * dst_image
)	

Calculates the auto-covariance of an image with the mean of a set of images.

Images must match size and type. Returns zero if the counter aborted.

void imProcessMultiplyConj (const imlImage * src_image1,
	const imlImage * src_image2,
	imlImage * dst_image
)	

Multiplies the conjugate of one complex image with another complex image.

Images must match size. Conj(img1) * img2

Can be done in-place.

Logical Arithmetic Operations

[Image Processing]

Detailed Description

Logical binary math operations for images.

See [im_process_pon.h](#)

Enumerations

```
enum imLogicOp { IM_BIT_AND, IM_BIT_OR, IM_BIT_XOR }
```

Functions

Enumeration Type Documentation

enum imLogicOp

Logical Operations.

Enumeration values:

IM_BIT_AND and = a & b

IM_BIT_OR or = a | b

IM_BIT_XOR xor = ~(a | b)

```
00227          {  
00228    IM_BIT_AND,   /**< and   =   a & b   */  
00229    IM_BIT_OR,   /**< or    =   a | b   */  
00230    IM_BIT_XOR   /**< xor   = ~(a | b) */  
00231 };
```

Function Documentation

```
void imProcessBitwiseOp( const imlImage * src_image1,  
                         const imlImage * src_image2,  
                         imlImage * dst_image,  
                         int op  
 )
```

Apply a logical operation.

Images must have data type IM_BYTE, IM USHORT or IM_INT. Can be done in place.

```
void imProcessBitwiseNot( const imlImage * src_image,  
                          imlImage * dst_image  
 )
```

Apply a logical NOT operation.

Images must have data type IM_BYTE, IM USHORT or IM_INT. Can be done in place.

```
void imProcessBitMask( const imlImage * src_image,  
                      imlImage * dst_image,  
                      unsigned char mask,  
                      int op  
 )
```

Apply a bit mask.

The same as imProcessBitwiseOp but the second image is replaced by a fixed mask.

Images must have data type IM_BYTE. It is valid only for AND, OR and

XOR. Can be done in place.

```
void imProcessBitPlane( const imImage * src_image,  
                        imImage * dst_image,  
                        int plane,  
                        int reset  
)
```

Extract or Reset a bit plane. For ex: 000X0000 or XXX0XXXX
(plane=3).

Images must have data type IM_BYTE. Can be done in place.

Tone Gamut Operations

[Image Processing]

Detailed Description

Operations that try to preserve the min-max interval in the output (the dynamic range).

See [im_process_pon.h](#)

Enumerations

```
enum imToneGamut {
    IM_GAMUT_NORMALIZE, IM_GAMUT_POW,
    IM_GAMUT_LOG, IM_GAMUT_EXP,
    IM_GAMUT_INVERT, IM_GAMUT_ZEROSTART,
    IM_GAMUT_SOLARIZE, IM_GAMUT_SLICE,
    IM_GAMUT_EXPAND, IM_GAMUT_CROP,
    IM_GAMUT_BRIGHTCONT
}
```

Functions

```
void imProcessToneGamut (const imImage *src_image, imImage
    *dst_image, int op, float *param)
void imProcessUnNormalize (const imImage *src_image, imImage
    *dst_image)
void imProcessDirectConv (const imImage *src_image, imImage
    *dst_image)
void imProcessNegative (const imImage *src_image, imImage
    *dst_image)
```

Enumeration Type Documentation

enum imToneGamut

Tone Gamut Operations.

Enumeration values:

<i>IM_GAMUT_NORMALIZE</i>	normalize = (a-min) / (max-min) (destination)
<i>IM_GAMUT_POW</i>	pow = ((a-min) / (max-min))^gamma * param[0]=gamma
<i>IM_GAMUT_LOG</i>	log = log(K * (a-min) / (max-min) + 1)) param[0]=K (K>0)
<i>IM_GAMUT_EXP</i>	exp = (exp(K * (a-min) / (max-min)) - 1 param[0]=K
<i>IM_GAMUT_INVERT</i>	invert = max - (a-min)
<i>IM_GAMUT_ZEROSTART</i>	zerostart = a - min
<i>IM_GAMUT_SOLARIZE</i>	solarize = a < level ? a: (level * (max-r param[0]=level percentage (0-100) relative to photography solarization effect.
<i>IM_GAMUT_SLICE</i>	slice = start < a a > end ? min: binar param[0]=start, param[1]=end, param[2]=min
<i>IM_GAMUT_EXPAND</i>	expand = a < start ? min: a > end ? max: (a - start) * (end - start) param[0]=start, param[1]=end
<i>IM_GAMUT_CROP</i>	crop = a < start ? start: a > end ? end: (a - start) * (end - start) param[0]=start, param[1]=end
<i>IM_GAMUT_BRIGHTCONT</i>	brightcont = a < min ? min: a > max ? max: (a - min) * (max - min) param[0]=bright_shift (-100%..+100%) change brightness and contrast simultaneously

```
00361          {
00362      IM_GAMUT_NORMALIZE,    /**< normalize = (a-min) / (max-min)
00363      IM_GAMUT_POW,        /**< pow       = ((a-min) / (max-min))^gamma
00364                                param[0]=gamma
00365      IM_GAMUT_LOG,        /**< log       = log(K * (a-min) / (max-min) + 1))
```

```
00366     param[0]=K      (K>0)
00367     IM_GAMUT_EXP,    /**< exp      = (exp(K * (a-min) / (max(a,0)-min(a,0)) * K) - 1) * 255 + 128;
00368                                         param[0]=K
00369     IM_GAMUT_INVERT, /**< invert    = max - (a-min)
00370     IM_GAMUT_ZEROSTART, /**< zerostart = a - min
00371     IM_GAMUT_SOLARIZE, /**< solarize = a < level ? a: (level - a) * 255 / (level - min);
00372                                         param[0]=level percent
00373                                         photography solarizat:
00374     IM_GAMUT_SLICE,   /**< slice     = start < a || a > end ? a: (start + end) / 2;
00375                                         param[0]=start, param[1]=end
00376     IM_GAMUT_EXPAND,  /**< expand    = a < start ? min: a > end ? max: a;
00377                                         param[0]=start, param[1]=end
00378     IM_GAMUT_CROP,   /**< crop      = a < start ? start: a > end ? end: a;
00379                                         param[0]=start, param[1]=end
00380     IM_GAMUT_BRIGHTCONT /**< brightcont = a < min ? min: a > max ? max: a;
00381                                         param[0]=bright_shift
00382                                         change brightness and contrast
00383 };
```

Function Documentation

void imProcessToneGamut(const imImage * src_image,	
	imImage * dst_image,	
	int op,	
	float * param	
)		

Apply a gamut operation with arguments.

Supports all data types except IM_COMPLEX.

The linear operation do a special conversion when min > 0 and max < 1, it forces min=0 and max=1.

IM_BYTE images have min=0 and max=255 always.

Can be done in place. When there is no extra params use NULL.

void imProcessUnNormalize(const imImage * src_image,	
	imImage * dst_image	
)		

Converts from (0-1) to (0-255), crop out of bounds values.

Source image must be IM_FLOAT, and destiny image must be IM_BYTE.

void imProcessDirectConv(const imImage * src_image,	
	imImage * dst_image	
)		

Directly converts IM_USHORT, IM_INT and IM_FLOAT into IM_BYTE images.

This can also be done using [imConvertDataType](#) with IM_CAST_DIRECT.

```
void imProcessNegative( const imImage * src_image,  
                        imImage *          dst_image  
                      )
```

A negative effect. Uses **imProcessToneGamut** with
IM_GAMUT_INVERT for non MAP images.
Supports all color spaces and all data types except **IM_COMPLEX**.

Convolution Operations

[Image Processing]

Detailed Description

See [im_process_loc.h](#)

Functions

```
int imProcessConvolve (const imlImage *src_image, imlImage  
    *dst_image, const imlImage *kernel)  
int imProcessConvolveRep (const imlImage *src_image, imlImage  
    *dst_image, const imlImage *kernel, int count)  
int imProcessCompassConvolve (const imlImage *src_image,  
    imlImage *dst_image, imlImage *kernel)  
void imProcessRotateKernel (imlImage *kernel)  
int imProcessDiffOfGaussianConvolve (const imlImage *src_image,  
    imlImage *dst_image, float stddev1, float stddev2)  
int imProcessDiffOfGaussianConvolveRep (const imlImage  
    *src_image, imlImage *dst_image, float stddev1, float stddev2)  
int imProcessLapOfGaussianConvolve (const imlImage *src_image,  
    imlImage *dst_image, float stddev)  
int imProcessMeanConvolve (const imlImage *src_image, imlImage  
    *dst_image, int kernel_size)  
int imProcessGaussianConvolveRep (const imlImage *src_image,  
    imlImage *dst_image, float stddev)  
int imProcessGaussianConvolve (const imlImage *src_image,  
    imlImage *dst_image, float stddev)  
int imProcessSobelConvolve (const imlImage *src_image, imlImage  
    *dst_image)  
void imProcessZeroCrossing (const imlImage *src_image, imlImage  
    *dst_image)  
void imProcessCanny (const imlImage *src_image, imlImage  
    *dst_image, float stddev)  
int imGaussianStdDev2Repetitions (float stddev)  
int imGaussianStdDev2KernelSize (float stddev)
```

Function Documentation

```
int imProcessConvolve( const imlImage * src_image,  
                      imlImage * dst_image,  
                      const imlImage * kernel  
                      )
```

Base Convolution with a kernel.

Kernel can be IM_INT or IM_FLOAT, but always IM_GRAY. Use kernel size odd for better results.

Supports all data types. The border is mirrored.

Returns zero if the counter aborted. Most of the convolutions use this function.

If the kernel image attribute "Description" exists it is used by the counter.

```
int imProcessConvolveRep( const imlImage * src_image,  
                         imlImage * dst_image,  
                         const imlImage * kernel,  
                         int count  
                         )
```

Repeats the convolution a number of times.

Returns zero if the counter aborted.

If the kernel image attribute "Description" exists it is used by the counter.

```
int imProcessCompassConvolve( const imlImage * src_image,  
                             imlImage * dst_image,  
                             imlImage * kernel  
                             )
```

Convolve with a kernel rotating it 8 times and getting the absolute maximum value.

Kernel must be square.

The rotation is implemented only for kernel sizes 3x3, 5x5 and 7x7.

Supports all data types except IM_COMPLEX. Returns zero if the counter aborted.

If the kernel image attribute "Description" exists it is used by the counter.

void imProcessRotateKernel (*imImage* * *kernel*)

Utility function to rotate a kernel one time.

int imProcessDiffOfGaussianConvolve (const <i>imImage</i> * <i>src_image</i>	
	<i>imImage</i> * <i>dst_image</i>	
	float <i>stddev1</i>,	
	float <i>stddev2</i>	
)		

Difference(Gaussian1, Gaussian2).

Supports all data types, but if source is IM_BYTE or IM USHORT destiny image must be of type IM_INT.

int imProcessDiffOfGaussianConvolveRep (const <i>imImage</i> * <i>src_image</i>	
	<i>imImage</i> * <i>dst_image</i>	
	float <i>stddev1</i>,	
	float <i>stddev2</i>	
)		

Difference(Gaussian1, Gaussian2) using gaussian repetitions.

Supports all data types, but if source is IM_BYTE or IM USHORT

destiny image must be of type IM_INT.

int imProcessLapOfGaussianConvolve (const imImage * src_image,	
	imImage * dst_image,	
	float stddev	
)		

Convolution with a laplacian of a gaussian kernel.

Supports all data types, but if source is IM_BYTE or IM USHORT
destiny image must be of type IM_INT.

int imProcessMeanConvolve (const imImage * src_image,	
	imImage * dst_image,	
	int kernel_size	
)		

Convolution with a kernel full of "1"s inside a circle.

Supports all data types.

int imProcessGaussianConvolveRep (const imImage * src_image,	
	imImage * dst_image,	
	float stddev	
)		

Convolution with a gaussian kernel. The gaussian is obtained by repetition of a base 3x3 IM_INT kernel.

Supports all data types.

int imProcessGaussianConvolve (const imImage * src_image,	
	imImage * dst_image,	
	float stddev	
)		

)

Convolution with a float gaussian kernel.
Supports all data types.

int imProcessSobelConvolve (const imlImage * <i>src_image</i>,		
	imlImage * <i>dst_image</i>	
)

Magnitude of the sobel convolution.
Supports all data types except IM_COMPLEX.

void imProcessZeroCrossing (const imlImage * <i>src_image</i>,		
	imlImage * <i>dst_image</i>	
)

Finds the zero crossings of IM_INT and IM_FLOAT images. Crossings are marked with non zero values indicating the intensity of the edge. It is usually used after a second derivative, laplace.

Extracted from XITE, Copyright 1991, Blab, UiO

<http://www.ifi.uio.no/~blab/Software/Xite/>

void imProcessCanny (const imlImage * <i>src_image</i>,		
	imlImage * <i>dst_image</i>,	
	float <i>stddev</i>	
)

First part of the Canny edge detector. Includes the gaussian filtering and the nonmax suppression.

After using this you could apply a Hysteresis Threshold, see
imProcessHysteresisThreshold.

Image must be IM_BYTE/IM_GRAY.

Implementation from the book:

J. R. Parker
"Algoritms for Image Processing and Computer Vision"
WILEY

```
int imGaussianStdDev2Repetitions( float stddev )
```

Calculates the number of 3x3 gaussian repetitions given the standard deviation.

```
int imGaussianStdDev2KernelSize( float stddev )
```

Calculates the kernel size given the standard deviation.

Rank Convolution Operations

[Image Processing]

Detailed Description

All the rank convolution use the same base function. Near the border the base function includes only the real image pixels in the rank. No border extensions are used.

See [im_process_loc.h](#)

Functions

```
int imProcessMedianConvolve (const imlImage *src_image, imlImage  
    *dst_image, int kernel_size)  
int imProcessRangeConvolve (const imlImage *src_image, imlImage  
    *dst_image, int kernel_size)  
int imProcessRankClosestConvolve (const imlImage *src_image,  
    imlImage *dst_image, int kernel_size)  
int imProcessRankMaxConvolve (const imlImage *src_image,  
    imlImage *dst_image, int kernel_size)  
int imProcessRankMinConvolve (const imlImage *src_image,  
    imlImage *dst_image, int kernel_size)
```

Function Documentation

<code>int imProcessMedianConvolve (</code>	<code>const imImage * src_image,</code>	
	<code>imImage * dst_image,</code>	
	<code>int kernel_size</code>	
<code>)</code>		

Rank convolution using the median value.

Returns zero if the counter aborted.

Supports all data types except IM_COMPLEX. Can be applied on color images.

<code>int imProcessRangeConvolve (</code>	<code>const imImage * src_image,</code>	
	<code>imImage * dst_image,</code>	
	<code>int kernel_size</code>	
<code>)</code>		

Rank convolution using (maximum-minimum) value.

Returns zero if the counter aborted.

Supports all data types except IM_COMPLEX. Can be applied on color images.

<code>int imProcessRankClosestConvolve (</code>	<code>const imImage * src_image,</code>	
	<code>imImage * dst_image,</code>	
	<code>int kernel_size</code>	
<code>)</code>		

Rank convolution using the closest maximum or minimum value.

Returns zero if the counter aborted.

Supports all data types except IM_COMPLEX. Can be applied on color

images.

int imProcessRankMaxConvolve (const imImage * src_image,	
	imImage * dst_image,	
	int kernel_size	
)		

Rank convolution using the maximum value.

Returns zero if the counter aborted.

Supports all data types except IM_COMPLEX. Can be applied on color images.

int imProcessRankMinConvolve (const imImage * src_image,	
	imImage * dst_image,	
	int kernel_size	
)		

Rank convolution using the minimum value.

Returns zero if the counter aborted.

Supports all data types except IM_COMPLEX. Can be applied on color images.

Morphology Operations for Binary Images

[Image Processing]

Detailed Description

See [im_process_loc.h](#)

Functions

```
int imProcessBinMorphConvolve (const imlImage *src_image,  
    imlImage *dst_image, const imlImage *kernel, int hit_white, int iter)  
int imProcessBinMorphErode (const imlImage *src_image, imlImage  
    *dst_image, int kernel_size, int iter)  
int imProcessBinMorphDilate (const imlImage *src_image, imlImage  
    *dst_image, int kernel_size, int iter)  
int imProcessBinMorphOpen (const imlImage *src_image, imlImage  
    *dst_image, int kernel_size, int iter)  
int imProcessBinMorphClose (const imlImage *src_image, imlImage  
    *dst_image, int kernel_size, int iter)  
int imProcessBinMorphOutline (const imlImage *src_image,  
    imlImage *dst_image, int kernel_size, int iter)  
void imProcessBinMorphThin (const imlImage *src_image, imlImage  
    *dst_image)
```

Function Documentation

<code>int imProcessBinMorphConvolve(</code>	<code>const imlImage * src_image,</code>
	<code>imlImage * dst_image,</code>
	<code>const imlImage * kernel,</code>
	<code>int hit_white,</code>
	<code>int iter</code>
<code>)</code>	

Base binary morphology convolution.

Images are all IM_BINARY. Kernel is IM_INT. Use kernel size odd for better results.

Hit white means hit=1 and miss=0, or else hit=0 and miss=1.

Use -1 for don't care positions in kernel.

The operation can be repeated by a number of iterations. The border is zero extended.

Almost all the binary morphology operations use this function.

If the kernel image attribute "Description" exists it is used by the counter.

<code>int imProcessBinMorphErode(</code>	<code>const imlImage * src_image,</code>
	<code>imlImage * dst_image,</code>
	<code>int kernel_size,</code>
	<code>int iter</code>
<code>)</code>	

Binary morphology convolution with a kernel full of "1"s and hit white.

<code>int imProcessBinMorphDilate(</code>	<code>const imlImage * src_image,</code>
	<code>imlImage * dst_image,</code>

	int	<i>kernel_size,</i>
	int	<i>iter</i>
)		

Binary morphology convolution with a kernel full of "0"s and hit black.

int imProcessBinMorphOpen (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>kernel_size,</i>
	int	<i>iter</i>
)		

Erode+Dilate. When iteration is more than one it means
Erode+Erode+Erode+...+Dilate+Dilate+Dilate+...

int imProcessBinMorphClose (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>kernel_size,</i>
	int	<i>iter</i>
)		

Dilate+Erode.

int imProcessBinMorphOutline (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>kernel_size,</i>
	int	<i>iter</i>
)		

Erode+Difference.

The difference from the source image is applied only once.

```
void imProcessBinMorphThin( const imImage * src_image,  
                           imImage * dst_image  
                         )
```

Thins the supplied binary image using Rosenfeld's parallel thinning algorithm.

Reference:

"Efficient Binary Image Thinning using Neighborhood Maps"
by Joseph M. Cychosz, 3ksnn64@ecn.purdue.edu
in "Graphics Gems IV", Academic Press, 1994

Morphology Operations for Gray Images

[Image Processing]

Detailed Description

See [im_process_loc.h](#)

Functions

```
int imProcessGrayMorphConvolve (const imlImage *src_image,  
                                imlImage *dst_image, const imlImage *kernel, int ismax)  
int imProcessGrayMorphErode (const imlImage *src_image, imlImage  
                            *dst_image, int kernel_size)  
int imProcessGrayMorphDilate (const imlImage *src_image, imlImage  
                            *dst_image, int kernel_size)  
int imProcessGrayMorphOpen (const imlImage *src_image, imlImage  
                            *dst_image, int kernel_size)  
int imProcessGrayMorphClose (const imlImage *src_image, imlImage  
                            *dst_image, int kernel_size)  
int imProcessGrayMorphTopHat (const imlImage *src_image,  
                               imlImage *dst_image, int kernel_size)  
int imProcessGrayMorphWell (const imlImage *src_image, imlImage  
                            *dst_image, int kernel_size)  
int imProcessGrayMorphGradient (const imlImage *src_image,  
                                imlImage *dst_image, int kernel_size)
```

Function Documentation

<code>int imProcessGrayMorphConvolve(</code>	<code>const imlImage * src_image,</code>
	<code>imlImage * dst_image,</code>
	<code>const imlImage * kernel,</code>
	<code>int ismax</code>
<code>)</code>	

Base gray morphology convolution.

Supports all data types except IM_COMPLEX. Can be applied on color images.

Kernel is always IM_INT. Use kernel size odd for better results.

You can use the maximum value or else the minimum value.

No border extensions are used. All the gray morphology operations use this function.

If the kernel image attribute "Description" exists it is used by the counter.

<code>int imProcessGrayMorphErode(</code>	<code>const imlImage * src_image,</code>
	<code>imlImage * dst_image,</code>
	<code>int kernel_size</code>
<code>)</code>	

Gray morphology convolution with a kernel full of "0"s and use minimum value.

<code>int imProcessGrayMorphDilate(</code>	<code>const imlImage * src_image,</code>
	<code>imlImage * dst_image,</code>
	<code>int kernel_size</code>
<code>)</code>	

Gray morphology convolution with a kernel full of "0"s and use maximum value.

int imProcessGrayMorphOpen (const imlImage * src_image,
	imlImage * dst_image,
	int kernel_size
)	

Erode+Dilate.

int imProcessGrayMorphClose (const imlImage * src_image,
	imlImage * dst_image,
	int kernel_size
)	

Dilate+Erode.

int imProcessGrayMorphTopHat (const imlImage * src_image,
	imlImage * dst_image,
	int kernel_size
)	

Open+Difference.

int imProcessGrayMorphWell (const imlImage * src_image,
	imlImage * dst_image,
	int kernel_size
)	

Close+Difference.

```
int imProcessGrayMorphGradient( const imlImage * src_image,  
                                imlImage * dst_image,  
                                int kernel_size  
                                )
```

Difference(Erode, Dilate).

Domain Transform Operations

[Image Processing]

Detailed Description

FFT, Wavelts, Hough, Distance.

FFTW Copyright Matteo Frigo, Steven G. Johnson and the MIT.

<http://www.fftw.org>

See fftw.h or fftw3.h

Must link with "im_fftw.lib" for FFTW version 2.1.5, and "im_fftw3.lib" for version 3.0.1.

Both libraries are available because version 3 was not that fast from version 2, and its file size is 3x bigger than version 2. But version 3 was not compiled using hardware optimizations.

The FFTW lib has a GPL license. The license of the "im_fftw.lib" library is automatically the GPL. So you cannot use it for commercial applications without contacting the authors.

See [im_process_glo.h](#)

Functions

```
void imProcessFFT (const imlImage *src_image, imlImage *dst_image)
void imProcessIFFT (const imlImage *src_image, imlImage
    *dst_image)
void imProcessFFTRaw (imlImage *src_image, int inverse, int center, int
    normalize)
void imProcessSwapQuadrants (imlImage *src_image, int
    center2origin)
int imProcessHoughLines (const imlImage *src_image, imlImage
    *dst_image)
int imProcessHoughLinesDraw (const imlImage *src_image, const
    imlImage *hough_points, imlImage *dst_image)
void imProcessCrossCorrelation (const imlImage *src_image1, const
    imlImage *src_image2, imlImage *dst_image)
void imProcessAutoCorrelation (const imlImage *src_image,
    imlImage *dst_image)
void imProcessDistanceTransform (const imlImage *src_image,
    imlImage *dst_image)
void imProcessRegionalMaximum (const imlImage *src_image,
    imlImage *dst_image)
```

Function Documentation

void imProcessFFT(const imImage * src_image,	
	imImage * dst_image	
)		

Forward FFT.

The result has its lowest frequency at the center of the image.

This is an unnormalized fft.

Images must be of the same size. Destiny image must be of type complex.

void imProcessIFFT(const imImage * src_image,	
	imImage * dst_image	
)		

Inverse FFT.

The image has its lowest frequency restored to the origin before the transform.

The result is normalized by (width*height).

Images must be of the same size and both must be of type complex.

void imProcessFFDraw(imImage * src_image,	
	int inverse,	
	int center,	
	int normalize	
)		

Raw in-place FFT (forward or inverse).

The lowest frequency can be centered after forward, or can be restored to the origin before inverse.

The result can be normalized after the transform by $\text{sqrt}(w \cdot h)$ [1] or by $(w \cdot h)$ [2], or left unnormalized [0].

Images must be of the same size and both must be of type complex.

void imProcessSwapQuadrants(imImage * src_image,
	int center2origin
)	

Auxiliary function for the raw FFT.

This is the function used internally to change the lowest frequency position in the image.

If the image size has even dimensions the flag "center2origin" is useless. But if it is odd, you must specify if its from center to origin (usually used before inverse) or from origin to center (usually used after forward).

Notice that this function is used for images in the the frequency domain.
Image type must be complex.

int imProcessHoughLines(const imImage * src_image,
	imImage * dst_image
)	

Hough Lines Transform.

It will detect white lines in a black background. So the source image must be a IM_BINARY image with the white lines of interest enhanced. The better the threshold with the white lines the better the line detection.

The destiny image must have IM_GRAY, IM_INT, width=180, height=2*rmax+1, where rmax is the image diagonal/2.

The houfh transform defines " $\cos(\theta) \cdot X + \sin(\theta) \cdot Y = \rho$ " and the parameters are in the interval:

theta = "0 .. 179", rho = "-height/2 .. height/2" .

Returns zero if the counter aborted.

Inspired from ideas in XITE, Copyright 1991, Blab, UiO

<http://www.ifi.uio.no/~blab/Software/Xite/>

int imProcessHoughLinesDraw (const imImage * src_image,
	const imImage * hough_points,
	imImage * dst_image
)	

Draw detected hough lines.

The source image must be IM_GRAY and IM_BYTE. The destiny image can be a clone of the source image or it can be the source image for in place processing.

The hough points image is a hough transform image that was thresholded to a IM_BINARY image, usually using a Local Max threshold operation. Again the better the threshold the better the results.

The destiny image will be set to IM_MAP, and the detected lines will be drawn using a red color.

Returns the number of detected lines.

void imProcessCrossCorrelation (const imImage * src_image1,
	const imImage * src_image2,
	imImage * dst_image
)	

Calculates the Cross Correlation in the frequency domain.

$$\text{CrossCorr}(a,b) = \text{IFFT}(\text{Conj}(\text{FFT}(a)) * \text{FFT}(b))$$

Images must be of the same size and only destiny image must be of type complex.

void imProcessAutoCorrelation (const imImage * src_image,
	imImage * dst_image
)	

Calculates the Auto Correlation in the frequency domain.
Uses the cross correlation. Images must be of the same size and only
destiny image must be of type complex.

```
void imProcessDistanceTransform( const imlImage * src_image,  
                                imlImage *          dst_image  
                                )
```

Calculates the Distance Transform of a binary image using an
approximation of the euclidian distance.
Each white pixel in the binary image is assigned a value equal to its
distance from the nearest black pixel.
Uses a two-pass algorithm incrementally calculating the distance.
Source image must be IM_BINARY, destiny must be IM_FLOAT.

```
void imProcessRegionalMaximum( const imlImage * src_image,  
                               imlImage *          dst_image  
                               )
```

Marks all the regional maximum of the distance transform.
source is IMGRAY/IM_FLOAT destiny in IM_BINARY.
We consider maximum all connected pixel values that have smaller
pixel values around it.

Special Effects

[Image Processing]

Detailed Description

Operations to change image appearance.

See [im_process_pon.h](#)

Functions

```
void imProcessPixelate (const imImage *src_image, imImage
    *dst_image, int box_size)
void imProcessPosterize (const imImage *src_image, imImage
    *dst_image, int level)
```

Function Documentation

```
void imProcessPixelate( const imImage * src_image,  
                        imImage * dst_image,  
                        int box_size  
                      )
```

Generates a zoom in effect averaging colors inside a square region.
Operates only on IM_BYTE images.

```
void imProcessPosterize( const imImage * src_image,  
                        imImage * dst_image,  
                        int level  
                      )
```

A simple Posterize effect. It reduces the number of colors in the image
eliminating less significant bit planes. Can have 1 to 7 levels. See
[imProcessBitMask](#).

Image data type must be integer.

Image Statistics Calculations

[Image Processing]

Detailed Description

Operations to calculate some statistics over images.

See [im_process_ana.h](#)

Data Structures

```
struct _imStats
```

Typedefs

```
typedef _imStats imStats
```

Functions

```
float imCalcRMSError (const imImage *image1, const  
                      imImage *image2)  
float imCalcSNR (const imImage *src_image, const imImage  
                  *noise_image)  
unsigned long imCalcCountColors (const imImage *src_image)  
void imCalcHistogram (const unsigned char *data, int count,  
                      unsigned long *histo, int accum)  
void imCalcUShortHistogram (const unsigned short *data,  
                           int count, unsigned long *histo, int accum)  
void imCalcGrayHistogram (const imImage *src_image,  
                          unsigned long *histo, int accum)  
void imCalcImageStatistics (const imImage *src_image,  
                           imStats *stats)  
void imCalcHistogramStatistics (const imImage  
                               *src_image, imStats *stats)  
void imCalcHistoImageStatistics (const imImage  
                                 *src_image, int *median, int *mode)
```

Typedef Documentation

typedef struct _imStats imStats

Numerical Statistics Structure

Function Documentation

```
float imCalcRMSError( const imImage * image1,  
                      const imImage * image2  
                    )
```

Calculates the RMS error between 2 images (Root Mean Square Error).

```
float imCalcSNR( const imImage * src_image,  
                  const imImage * noise_image  
                )
```

Calculates the SNR of an image and its noise (Signal Noise Ratio).

```
unsigned long imCalcCountColors( const imImage * src_image )
```

Count the number of different colors in an image.
Image must be IM_BYTE, but all color spaces except IM_CMYK.

```
void imCalcHistogram( const unsigned char * data,  
                     int count,  
                     unsigned long * histo,  
                     int accum  
                   )
```

Calculates the histogram of a IM_BYTE data.
Histogram is always 256 positions long.
When accum is different from zero it calculates the accumulative histogram.

void imCalcUShortHistogram (const unsigned short *	<i>data,</i>
	int	<i>count,</i>
	unsigned long *	<i>histo,</i>
	int	<i>accum</i>
)		

Calculates the histogram of a IM USHORT data.

Histogram is always 65535 positions long.

When accum is different from zero it calculates the accumulative histogram.

void imCalcGrayHistogram (const imImage *	<i>src_image,</i>
	unsigned long *	<i>histo,</i>
	int	<i>accum</i>
)		

Calculates the gray histogram of an image.

Image must be IM_BYTE/(IM_RGB, IM_GRAY, IM_BINARY or IM_MAP).

If the image is IM_RGB then the histogram of the luma component is calculated.

Histogram is always 256 positions long.

When accum is different from zero it calculates the accumulative histogram.

void imCalcImageStatistics (const imImage *	<i>src_image,</i>
	imStats *	<i>stats</i>
)		

Calculates the statistics about the image data.

There is one stats for each depth plane. For ex: stats[0]=red stats,

stats[0]=green stats, ...
Supports all data types except IM_COMPLEX.

void imCalcHistogramStatistics(const imImage * src_image,	
	imStats * stats	
)		

Calculates the statistics about the image histogram data.
There is one stats for each depth plane. For ex: stats[0]=red stats,
stats[0]=green stats, ...
Only IM_BYTE images are supported.

void imCalcHistolImageStatistics(const imImage * src_image,	
	int * median,	
	int * mode	
)		

Calculates some extra statistics about the image histogram data.
There is one stats for each depth plane.
Only IM_BYTE images are supported.
mode will be -1 if more than one max is found.

Image Analysis

[Image Processing]

Detailed Description

See [im_process_ana.h](#)

Functions

```
int imAnalyzeFindRegions (const imImage *src_image, imImage
    *dst_image, int connect)
void imAnalyzeMeasureArea (const imImage *image, int *area)
void imAnalyzeMeasurePerimArea (const imImage *image, float
    *perimarea)
void imAnalyzeMeasureCentroid (const imImage *image, const int
    *area, int region_count, float *cx, float *cy)
void imAnalyzeMeasurePrincipalAxis (const imImage *image, const
    int *data_area, const float *cx, const float *cy, const int
    region_count, float *major_slope, float *major_length, float
    *minor_slope, float *minor_length)
void imAnalyzeMeasureHoles (const imImage *image, int connect, int
    *holes_count, int *area, float *perim)
void imAnalyzeMeasurePerimeter (const imImage *image, float
    *perim)
void imProcessPerimeterLine (const imImage *src_image, imImage
    *dst_image)
void imProcessPrune (const imImage *src_image, imImage
    *dst_image, int connect, int start_size, int end_size)
void imProcessFillHoles (const imImage *src_image, imImage
    *dst_image, int connect)
```

Function Documentation

<code>int imAnalyzeFindRegions(</code>	<code>const imImage * src_image,</code>	
	<code>imImage * dst_image,</code>	
	<code>int connect</code>	
<code>)</code>		

Find white regions in binary image.

Result is IM USHORT type. Regions can be 4 connected or 8 connected.

Returns the number of regions found. Background is marked as 0.

<code>void imAnalyzeMeasureArea(</code>	<code>const imImage * image,</code>	
	<code>int * area</code>	
<code>)</code>		

Measure the actual area of all regions. Holes are not included.

This is the number of pixels of each region.

Source image is IM USHORT type (the result of [imAnalyzeFindRegions](#)).

data has size the number of regions.

<code>void imAnalyzeMeasurePerimArea(</code>	<code>const imImage * image,</code>	
	<code>float * perimarea</code>	
<code>)</code>		

Measure the polygonal area limited by the perimeter line of all regions.

Holes are not included.

Notice that some regions may have polygonal area zero.

Source image is IM USHORT type (the result of [imAnalyzeFindRegions](#)).

data has size the number of regions.

void imAnalyzeMeasureCentroid (const imImage * image,	
	const int * area,	
	int region_count,	
	float * cx,	
	float * cy	
)		

Calculate the centroid position of all regions. Holes are not included.

Source image is IM USHORT type (the result of
[imAnalyzeFindRegions](#)).

data has size the number of regions. If area is NULL will be internally calculated.

void imAnalyzeMeasurePrincipalAxis (const imImage * image,	
	const int * data_area,	
	const float * cx,	
	const float * cy,	
	const int region_cou	
	float * major_slop	
	float * major_leng	
	float * minor_slop	
	float * minor_leng	
)		

Calculate the principal major axis slope of all regions.

Source image is IM USHORT type (the result of
[imAnalyzeFindRegions](#)).

data has size the number of regions. If area or centroid are NULL will be internally calculated.

Principal (major and minor) axes are defined to be those axes that pass through the centroid, about which the moment of inertia of the region is,

respectively maximal or minimal.

void imAnalyzeMeasureHoles (const imImage * image,	
	int	connect,
	int *	holes_count,
	int *	area,
	float *	perim
)		

Measure the number and area of holes of all regions.

Source image is IM_USHORT type (the result of
[imAnalyzeFindRegions](#)).

data has size the number of regions. If some data is NULL it will be not calculated.

void imAnalyzeMeasurePerimeter (const imImage * image,	
	float *	perim
)		

Measure the total perimeter of all regions (external and internal).

Source image is IM_USHORT type (the result of
[imAnalyzeFindRegions](#)).

It uses a half-pixel inter distance for 8 neigboors in a perimeter of a 4 connected region.

This function can also be used to measure line lenght.

data has size the number of regions.

void imProcessPerimeterLine (const imImage * src_image,	
	imImage * dst_image	
)		

Isolates the perimeter line of gray integer images. Background is

defined as being black (0).

It just checks if at least one of the 4 connected neighbors is non zero.

void imProcessPrune (const imImage * src_image,
	imImage * dst_image,
	int connect,
	int start_size,
	int end_size
)	

Eliminates regions that have size outside the given interval.

Source and destiny are binary images. Regions can be 4 connected or 8 connected.

Regions touching the image border will also be eliminated.

Can be done in-place. end_size can be zero to ignore big objects.

void imProcessFillHoles (const imImage * src_image,
	imImage * dst_image,
	int connect
)	

Fill holes inside white regions.

Source and destiny are binary images. Regions can be 4 connected or 8 connected.

Can be done in-place.

Utilities

Detailed Description

See [im_util.h](#)

Modules

```
group Binary File Access
group Color Manipulation
group Complex Numbers
group Counter
group Windows DIB
group Math Utilities
group Palette Generators
group String Utilities
group Color Utilities
group Data Type Utilities
group Binary Data Utilities
group Data Compression Utilities
```

Data Structures

class **imAttribTable**

Attributes Table. [More...](#)

class **imAttribArray**

Attributes Table. [More...](#)

Defines

```
#define IM_MIN(_a, _b) (_a < _b? _a: _b)
#define IM_MAX(_a, _b) (_a > _b? _a: _b)
```

Binary Data Utilities

[Utilities]

Detailed Description

See [im_util.h](#)

Enumerations

```
enum imByteOrder { IM_LITTLEENDIAN, IM_BIGENDIAN }
```

Functions

```
int imBinCPUByteOrder (void)
void imBinSwapBytes (void *data, int count, int size)
void imBinSwapBytes2 (void *data, int count)
void imBinSwapBytes4 (void *data, int count)
void imBinSwapBytes8 (void *data, int count)
```

Enumeration Type Documentation

enum imByteOrder

CPU Byte Orders.

Enumeration values:

IM_LITTLEENDIAN Little Endian - The most significant byte is on the least significant address.
IM_BIGENDIAN Big Endian - The most significant byte is on the most significant address.

```
00174 {  
00175     IM_LITTLEENDIAN, /*< Little Endian - The most significant  
00176     IM_BIGENDIAN      /*< Big Endian - The most significant byt  
00177 };
```

Function Documentation

```
int imBinCPUByteOrder( void )
```

Returns the current CPU byte order.

```
void imBinSwapBytes( void * data,  
                     int    count,  
                     int    size  
)
```

Changes the byte order of an array of 2, 4 or 8 byte values.

```
void imBinSwapBytes2( void * data,  
                     int    count  
)
```

Changes the byte order of an array of 2 byte values.

```
void imBinSwapBytes4( void * data,  
                     int    count  
)
```

Inverts the byte order of the 4 byte values

```
void imBinSwapBytes8( void * data,  
                     int    count  
)
```

Inverts the byte order of the 8 byte values

Binary File Access

[Utilities]

Detailed Description

These functions are very usefull for reading/writing binary files that have headers or data that have to be converted depending on the current CPU byte order. It can invert 2, 4 or 8 bytes numbers to/from little/big-endian orders.

It will process the data only if the file format is different from the current CPU.

Can read from disk or memory. In case of a memory buffer, the file name must be the **imBinMemoryFileName** structure.

See [im_binfile.h](#)

Data Structures

struct [_imBinMemoryFileName](#)
Memory File I/O Filename. [More...](#)

Typedefs

```
typedef _imBinMemoryFileName imBinMemoryFileName
```

Enumerations

```
enum imBinFileModule {  
    IM_RAWFILE, IM_STREAM, IM_MEMFILE, IM_SUBFILE,  
    IM_IOCUSTOM0  
}
```

Functions

```
imBinFile * imBinFileOpen (const char *pFileName)
imBinFile * imBinFileNew (const char *pFileName)
    void imBinFileClose (imBinFile *bfile)
        int imBinFileError (imBinFile *bfile)
unsigned long imBinFileSize (imBinFile *bfile)
    int imBinFileByteOrder (imBinFile *bfile, int pByteOrder)
unsigned long imBinFileRead (imBinFile *bfile, void *pValues, unsigned
    long pCount, int pSizeOf)
unsigned long imBinFileWrite (imBinFile *bfile, void *pValues, unsigned
    long pCount, int pSizeOf)
unsigned long imBinFilePrintf (imBinFile *bfile, char *format,...)
    void imBinFileSeekTo (imBinFile *bfile, unsigned long
        pOffset)
    void imBinFileSeekOffset (imBinFile *bfile, long pOffset)
    void imBinFileSeekFrom (imBinFile *bfile, long pOffset)
unsigned long imBinFileTell (imBinFile *bfile)
    int imBinFileEndOfFile (imBinFile *bfile)
    int imBinFileSetCurrentModule (int pModule)
```

Typedef Documentation

typedef struct imBinMemoryFileName imBinMemoryFileName

Memory File I/O Filename.

Fake file name for the memory I/O module.

Enumeration Type Documentation

enum **imBinFileModule**

Predefined I/O Modules.

Enumeration values:

- IM_RAWFILE* System dependent file I/O Rotines.
- IM_STREAM* Standard Ansi C Stream I/O Rotines.
- IM_MEMFILE* Uses a memory buffer.
- IM_SUBFILE* It is a sub file. FileName is a imBinFile* pointer from module.
- IM_IOCUSTOM0* Other registered modules starts from here.

```
00103 {  
00104     IM_RAWFILE,    /**< System dependent file I/O Rotines. */  
00105     IM_STREAM,    /**< Standard Ansi C Stream I/O Rotines. */  
00106     IM_MEMFILE,    /**< Uses a memory buffer. */  
00107     IM_SUBFILE,    /**< It is a sub file. FileName is a imBinFile* pointer from module. */  
00108     IM_IOCUSTOM0  /**< Other registered modules starts from here. */  
00109 };
```

Function Documentation

`imBinFile* imBinFileOpen(const char * pFileName)`

Opens an existant binary file for reading. The default file byte order is the CPU byte order. Returns NULL if failed.

`imBinFile* imBinFileNew(const char * pFileName)`

Creates a new binary file for writing. The default file byte order is the CPU byte order. Returns NULL if failed.

`void imBinFileClose(imBinFile * bfile)`

Closes the file.

`int imBinFileError(imBinFile * bfile)`

Indicates that was an error on the last operation.

`unsigned long imBinFileSize(imBinFile * bfile)`

Returns the file size in bytes.

<code>int imBinFileByteOrder(imBinFile * <i>bfile</i>,</code>		
	<code>int</code>	<code><i>pByteOrder</i></code>
	<code>)</code>	

Changes the file byte order. Returns the old one.

unsigned long imBinFileRead (imBinFile *	<i>bfile,</i>
	void *	<i>pValues,</i>
	unsigned long	<i>pCount,</i>
	int	<i>pSizeOf</i>
)		

Reads an array of count values with byte sizes: 1, 2, 4, or 8. And invert the byte order if necessary after read.

unsigned long imBinFileWrite (imBinFile *	<i>bfile,</i>
	void *	<i>pValues,</i>
	unsigned long	<i>pCount,</i>
	int	<i>pSizeOf</i>
)		

Writes an array of values with sizes: 1, 2, 4, or 8. And invert the byte order if necessary before write.

ATENTION: The function will not make a temporary copy of the values to invert the byte order.

So after the call the values will be invalid, if the file byte order is different from the CPU byte order.

unsigned long imBinFilePrintf (imBinFile *	<i>bfile,</i>
	char *	<i>format,</i>
		<i>...</i>
)		

Writes a string without the NULL terminator. The function uses sprintf to compose the string.

The internal buffer is fixed at 4096 bytes.

```
void imBinFileSeekTo ( imBinFile * bfile,  
                      unsigned long pOffset  
)
```

Moves the file pointer from the begining of the file.

When writing to a file seeking can go beyond the end of the file.

```
void imBinFileSeekOffset ( imBinFile * bfile,  
                         long pOffset  
)
```

Moves the file pointer from current position.

If the offset is a negative value the pointer moves backwards.

```
void imBinFileSeekFrom ( imBinFile * bfile,  
                        long pOffset  
)
```

Moves the file pointer from the end of the file.

The offset is usually a negative value.

```
unsigned long imBinFileTell ( imBinFile * bfile )
```

Returns the current offset position.

```
int imBinFileEndOfFile ( imBinFile * bfile )
```

Indicates that the file pointer is at the end of the file.

```
int imBinFileSetCurrentModule ( int pModule )
```

Sets the current I/O module.

Returns:

the previous function set, or -1 if failed.

Color Utilities

[Utilities]

Detailed Description

See [im_util.h](#)

Function Documentation

```
long imColorEncode( unsigned char red,  
                     unsigned char green,  
                     unsigned char blue  
)
```

Encode RGB components in a long for palette usage.
"long" definition is compatible with the CD library definition.

```
void imColorDecode( unsigned char * red,  
                     unsigned char * green,  
                     unsigned char * blue,  
                     long color  
)
```

Decode RGB components from a long for palette usage.
"long" definition is compatible with the CD library definition.

Color Manipulation

[Utilities]

Detailed Description

Functions to convert from one color space to another, and color gammut utilities.

See [im_color.h](#)

Some Color Science

Y is luminance, a linear-light quantity. It is directly proportional to physical intensity weighted by the spectral sensitivity of human vision.

L* is lightness, a nonlinear luminance that approximates the perception of brightness. It is nearly perceptual uniform. It has a range of 0 to 100.

Y' is luma, a nonlinear luminance that approximates lightness.

Brightness is a visual sensation according to which an area appears to exhibit more or less light. It is a subjective quantity and can not be measured.

One unit of euclidian distance in CIE L*u*v* or CIE L*a*b* corresponds roughly to a just-noticeable difference (JND) of color.

```
ChromaUV = sqrt(u*u + v*v)
HueUV = atan2(v, u)
SaturationUV = ChromaUV / L      (called psychometric saturation
(the same can be calculated for Lab)
```

IEC 61966-2.1 Default RGB colour space - sRGB

- ITU-R Recommendation BT.709 (D65 white point).
- D65 White Point (X,Y,Z) = (0.9505 1.0000 1.0890)

Documentation extracted from Charles Poynton - Digital Video and HDTV
- Morgan Kaufmann - 2003.

Links

- www.color.org - ICC
- www.srgb.com - sRGB
- www.poynton.com - Charles Poynton
- www.littlecms.com - A free Color Management System (use this if you need precise color conversions)

Color Component Intervals

All the color components are stored in the 0-max interval, even the signed ones.

Here are the pre-defined intervals for each data type. These values are used for standard color conversion. You should normalize data before converting between color spaces.

byte	[0, 255]	or [-128, +127]	(1 byte)
ushort	[0, 65535]	or [-32768, +32767]	(2 bytes)
int	[0, 16777215]	or [-8388608, +8388607]	(3 bytes)
float	[0, 1]	or [-0.5, +0.5]	(4 bytes)

Modules

group [HSI Color Coordinate System Conversions](#)

Functions

```
float imColorZero (int data_type)
    int imColorMax (int data_type)
template<class T>
    T imColorQuantize (const float &value, const T &max)
template<class T>
float imColorReconstruct (const T &value, const T &max)
template<class T>
void imColorYCbCr2RGB (const T Y, const T Cb, const T Cr, T &R, T
    &G, T &B, const T &zero, const T &max)
template<class T>
void imColorRGB2YCbCr (const T R, const T G, const T B, T &Y, T
    &Cb, T &Cr, const T &zero)
template<class T>
void imColorCMYK2RGB (const T C, const T M, const T Y, const T K,
    T &R, T &G, T &B, const T &max)
template<class T>
void imColorXYZ2RGB (const T X, const T Y, const T Z, T &R, T &G, T
    &B, const T &max)
template<class T>
void imColorRGB2XYZ (const T R, const T G, const T B, T &X, T &Y, T
    &Z)
void imColorXYZ2Lab (const float X, const float Y, const float Z, float
    &L, float &a, float &b)
void imColorLab2XYZ (const float L, const float a, const float b, float
    &X, float &Y, float &Z)
void imColorXYZ2Luv (const float X, const float Y, const float Z, float
    &L, float &u, float &v)
void imColorLuv2XYZ (const float L, const float u, const float v, float
```

```
&X, float &Y, float &Z)
float imColorTransfer2Linear (const float &nonlinear_value)
float imColorTransfer2Nonlinear (const float &value)
void imColorRGB2RGBNonlinear (const float RL, const float GL, const
    float BL, float &R, float &G, float &B)
template<class T>
    T imColorRGB2Luma (const T R, const T G, const T B)
float imColorLuminance2Lightness (const float &Y)
float imColorLightness2Luminance (const float &L)
```

Function Documentation

float imColorZero(int *data_type*) [inline]

Returns the zero value for color conversion porpoises.

This is a value to be compensated when the *data_type* is unsigned and component is signed.

```
00078 {  
00079     float zero[] = {128.0f, 32768.0f, 8388608.0f, 0.5f};  
00080     return zero[data_type];  
00081 }
```

int imColorMax(int *data_type*) [inline]

Returns the maximum value for color conversion porpoises.

```
00086 {  
00087     int max[] = {255, 65535, 16777215, 1};  
00088     return max[data_type];  
00089 }
```

template<class T>

T imColorQuantize(const float &	<i>value</i> ,
	const T &	<i>max</i>
)	[inline]	

Quantize 0-1 values into 0-max.

$$q = r * (\max + 1)$$

Divide by the size of each interval $1/(\max+1)$, then the value is rounded down in the typecast.

But 0 is mapped to 0, and 1 is mapped to max.

```
00099 {
```

```

00100     if (max == 1) return (T)value; // to allow a dummy quantizer
00101     if (value >= 1) return max;
00102     if (value <= 0) return 0;
00103     return (T)(value*(max + 1));
00104 }
```

template<class T>	
float imColorReconstruct(const T & value,	
	const T & max
) [inline]

Reconstruct 0-max values into 0-1.

$$r = (q + 0.5)/(max + 1)$$

Add 0.5 to set the same origin, then multiply by the size of each interval $1/(max+1)$.

But 0 is mapped to 0, and max is mapped to 1.

```

00113 {
00114     if (max == 1) return (float)value; // to allow a dummy recon
00115     if (value <= 0) return 0;
00116     if (value >= max) return 1;
00117     return (((float)value + 0.5f)/((float)max + 1.0f));
00118 }
```

template<class T>	
void imColorYCbCr2RGB(const T & Y,	
	const T & Cb,
	const T & Cr,
	T & R,
	T & G,
	T & B,
	const T & zero,
	const T & max
) [inline]

Converts Y'CbCr to R'G'B' (all nonlinear).

ITU-R Recommendation 601-1 with no headroom/footroom.

```
0 <= Y <= 1 ; -0.5 <= CbCr <= 0.5 ; 0 <= RGB <= 1

R' = Y' + 0.000 *Cb + 1.402 *Cr
G' = Y' - 0.344 *Cb - 0.714 *Cr
B' = Y' + 1.772 *Cb + 0.000 *Cr

00134 {
00135     float r = float(Y
00136         + 1.402f * (Cr - zero));
00137     float g = float(Y - 0.344f * (Cb - zero) - 0.714f * (Cr - zero));
00138     float b = float(Y + 1.772f * (Cb - zero));
00139     // now we should enforce 0<= rgb <= max
00140
00141     R = (T)IM_CROPMAX(r, max);
00142     G = (T)IM_CROPMAX(g, max);
00143     B = (T)IM_CROPMAX(b, max);
00144 }
```

template<class T>

void imColorRGB2YCbCr(const T	R,
	const T	G,
	const T	B,
	T &	Y,
	T &	Cb,
	T &	Cr,
	const T &	zero
)	[inline]	

Converts R'G'B' to Y'CbCr (all nonlinear).

ITU-R Recommendation 601-1 with no headroom/footroom.

```
0 <= Y <= 1 ; -0.5 <= CbCr <= 0.5 ; 0 <= RGB <= 1

Y' = 0.299 *R' + 0.587 *G' + 0.114 *B'
Cb = -0.169 *R' - 0.331 *G' + 0.500 *B'
Cr = 0.500 *R' - 0.419 *G' - 0.081 *B'
```

```

00160 {
00161     Y = (T)( 0.299f *R + 0.587f *G + 0.114f *B);
00162     Cb = (T)(-0.169f *R - 0.331f *G + 0.500f *B + (float)zero);
00163     Cr = (T)( 0.500f *R - 0.419f *G - 0.081f *B + (float)zero);
00164
00165     // there is no need for cropping here, YCrCr is already at
00166 }
```

template<class T>

void imColorCMYK2RGB (const T	C,
	const T	M,
	const T	Y,
	const T	K,
	T &	R,
	T &	G,
	T &	B,
	const T &	max
)	[inline]

Converts C'M'Y'K' to R'G'B' (all nonlinear).

This is a poor conversion that works for a simple visualization.

```

0 <= CMYK <= 1 ; 0 <= RGB <= 1

R = (1 - K) * (1 - C)
G = (1 - K) * (1 - M)
B = (1 - K) * (1 - Y)
```

```

00181 {
00182     T W = max - K;
00183     R = (T)((W * (max - C)) / max);
00184     G = (T)((W * (max - M)) / max);
00185     B = (T)((W * (max - Y)) / max);
00186
00187     // there is no need for cropping here, RGB is already at th
00188 }
```

template<class T>

```

void imColorXYZ2RGB( const T X,
const T Y,
const T Z,
T & R,
T & G,
T & B,
const T & max
) [inline]

```

Converts CIE XYZ to Rec 709 RGB (all linear).
ITU-R Recommendation BT.709 (D65 white point).

```

0 <= XYZ <= 1 ; 0 <= RGB <= 1

R = 3.2406 *X - 1.5372 *Y - 0.4986 *Z
G = -0.9689 *X + 1.8758 *Y + 0.0415 *Z
B = 0.0557 *X - 0.2040 *Y + 1.0570 *Z

00203 {
00204     float r = 3.2406f *X - 1.5372f *Y - 0.4986f *Z;
00205     float g = -0.9689f *X + 1.8758f *Y + 0.0415f *Z;
00206     float b = 0.0557f *X - 0.2040f *Y + 1.0570f *Z;
00207
00208     // we need to crop because not all XYZ colors are visible
00209
00210     R = (T)IM_CROPMAX(r, max);
00211     G = (T)IM_CROPMAX(g, max);
00212     B = (T)IM_CROPMAX(b, max);
00213 }

```

```

template<class T>
void imColorRGB2XYZ( const T R,
const T G,
const T B,
T & X,
T & Y,
T & Z
) [inline]

```

)

Converts Rec 709 RGB to CIE XYZ (all linear).
ITU-R Recommendation BT.709 (D65 white point).

```
0 <= XYZ <= 1 ; 0 <= RGB <= 1

X = 0.4124 *R + 0.3576 *G + 0.1805 *B
Y = 0.2126 *R + 0.7152 *G + 0.0722 *B
Z = 0.0193 *R + 0.1192 *G + 0.9505 *B

00228 {
00229     X = (T)(0.4124f *R + 0.3576f *G + 0.1805f *B);
00230     Y = (T)(0.2126f *R + 0.7152f *G + 0.0722f *B);
00231     Z = (T)(0.0193f *R + 0.1192f *G + 0.9505f *B);
00232
00233 // there is no need for cropping here, XYZ is already at the
00234 }
```

```
void imColorXYZ2Lab( const float X,
                      const float Y,
                      const float Z,
                      float & L,
                      float & a,
                      float & b
) [inline]
```

Converts CIE XYZ (linear) to CIE L*a*b* (nonlinear).
The white point is D65.

```
0 <= L <= 1 ; -0.5 <= ab <= +0.5 ; 0 <= XYZ <= 1

if (t > 0.008856)
    f(t) = pow(t, 1/3)
else
    f(t) = 7.787*t + 16/116

fX = f(X / Xn)      fY = f(Y / Yn)      fZ = f(Z / Zn)

L = 1.16 * fY - 0.16
```

```

a = 2.5 * (fX - fY)
b = (fY - fZ)

```

```

00260 {
00261     float fX = X / 0.9505f; // white point D65
00262     float fY = Y / 1.0f;
00263     float fZ = Z / 1.0890f;
00264
00265     fX = IM_FWLAB(fX);
00266     fY = IM_FWLAB(fY);
00267     fZ = IM_FWLAB(fZ);
00268
00269     L = 1.16f * fY - 0.16f;
00270     a = 2.5f * (fX - fY);
00271     b = (fY - fZ);
00272 }

```

void imColorLab2XYZ(const float	<i>L</i> ,
	const float	<i>a</i> ,
	const float	<i>b</i> ,
	float &	<i>X</i> ,
	float &	<i>Y</i> ,
	float &	<i>Z</i>
)	[inline]	

Converts CIE L*a*b* (nonlinear) to CIE XYZ (linear).

The white point is D65.

$0 \leq L \leq 1 ; -0.5 \leq ab \leq +0.5 ; 0 \leq XYZ \leq 1$

```

00285 {
00286     float fY = (L + 0.16f) / 1.16f;
00287     float gY = IM_GWLAB(fY);
00288
00289     float fgY = IM_FWLAB(gY);
00290     float gX = fgY + a / 2.5f;
00291     float gZ = fgY - b;
00292     gX = IM_GWLAB(gX);
00293     gZ = IM_GWLAB(gZ);
00294
00295     X = gX * 0.9505f; // white point D65

```

```

00296     Y = gY * 1.0f;
00297     Z = gZ * 1.0890f;
00298 }
```

void imColorXYZ2Luv(const float	X,
	const float	Y,
	const float	Z,
	float &	L,
	float &	u,
	float &	v
)	[inline]	

Converts CIE XYZ (linear) to CIE L*u*v* (nonlinear).
The white point is D65.

```

0 <= L <= 1 ; -1 <= uv <= +1 ; 0 <= XYZ <= 1

Y = Y / 1.0      (for D65)
if (Y > 0.008856)
    fY = pow(Y, 1/3)
else
    fY = 7.787 * Y + 0.16/1.16
L = 1.16 * fY - 0.16

U(x, y, z) = (4 * x)/(x + 15 * y + 3 * z)
V(x, y, z) = (9 * x)/(x + 15 * y + 3 * z)
un = U(Xn, Yn, Zn) = 0.1978      (for D65)
vn = V(Xn, Yn, Zn) = 0.4683      (for D65)
fu = U(X, Y, Z)
fv = V(X, Y, Z)

u = 13 * L * (fu - un)
v = 13 * L * (fv - vn)
```

```

00325 {
00326     float XYZ = (float)(X + 15 * Y + 3 * Z);
00327     float fY = Y / 1.0f;
00328
00329     if (XYZ != 0)
00330     {
00331         L = 1.16f * IM_FWLAB(fY) - 0.16f;
00332         u = 6.5f * L * ((4 * X)/XYZ - 0.1978f);
```

```

00333     v = 6.5f * L * ((9 * Y)/XYZ - 0.4683f);
00334 }
00335 else
00336 {
00337     L = u = v = 0;
00338 }
00339 }
```

void imColorLuv2XYZ(const float	<i>L</i> ,
	const float	<i>u</i> ,
	const float	<i>v</i> ,
	float &	<i>X</i> ,
	float &	<i>Y</i> ,
	float &	<i>Z</i>
)	[inline]	

Converts CIE L*u*v* (nonlinear) to CIE XYZ (linear).

The white point is D65. $0 \leq L \leq 1$; $-0.5 \leq uv \leq +0.5$; $0 \leq XYZ \leq 1$

```

00348 {
00349     float fY = (L + 0.16f) / 1.16f;
00350     Y = IM_GWLAB(fY) * 1.0f;
00351
00352     float ul = 0.1978f, vl = 0.4683f;
00353     if (L != 0)
00354     {
00355         ul = u / (6.5f * L) + 0.1978f;
00356         vl = v / (6.5f * L) + 0.4683f;
00357     }
00358
00359     X = ((9 * ul) / (4 * vl)) * Y;
00360     Z = ((12 - 3 * ul - 20 * vl) / (4 * vl)) * Y;
00361 }
```

float imColorTransfer2Linear(const float &	<i>nonlinear_value</i>)	[inl
--------------------------------------	--------------------------	------------------------	----------	-------------

Converts nonlinear values to linear values.

We use the sRGB transfer function. sRGB uses ITU-R 709 primaries and D65 white point.

```
0 <= l <= 1 ; 0 <= v <= 1

if (v < 0.03928)
    l = v / 12.92
else
    l = pow((v + 0.055) / 1.055, 2.4)

00375 {
00376     if (nonlinear_value < 0.03928f)
00377         return nonlinear_value / 12.92f;
00378     else
00379         return powf((nonlinear_value + 0.055f) / 1.055f, 2.4f);
00380 }
```

float imColorTransfer2Nonlinear(const float & **value**) [inline]

Converts linear values to nonlinear values.

We use the sRGB transfer function. sRGB uses ITU-R 709 primaries and D65 white point.

```
0 <= l <= 1 ; 0 <= v <= 1

if (l < 0.0031308)
    v = 12.92 * l
else
    v = 1.055 * pow(l, 1/2.4) - 0.055

00394 {
00395     if (value < 0.0031308f)
00396         return 12.92f * value;
00397     else
00398         return 1.055f * powf(value, 1.0f/2.4f) - 0.055f;
00399 }
```

void imColorRGB2RGBNonlinear(const float **RL**,

const float **GL**,

const float **BL**,

float & **R**,

	float &	G,
	float &	B
)	[inline]

Converts RGB (linear) to R'G'B' (nonlinear).

```
00405 {
00406     R = imColorTransfer2Nonlinear(RL);
00407     G = imColorTransfer2Nonlinear(GL);
00408     B = imColorTransfer2Nonlinear(BL);
00409 }
```

template<class T>		
T	imColorRGB2Luma (const T R,
		const T G,
		const T B
)	[inline]

Converts R'G'B' to Y' (all nonlinear).

```
Y' = 0.299 *R' + 0.587 *G' + 0.114 *B'

00418 {
00419     return (T)((299 * R + 587 * G + 114 * B) / 1000);
00420 }
```

float imColorLuminance2Lightness (const float & Y) [inline]

Converts Luminance (CIE Y) to Lightness (CIE L*) (all linear).
The white point is D65.

```
0 <= Y <= 1 ; 0 <= L* <= 1

Y = Y / 1.0      (for D65)
if (Y > 0.008856)
    fY = pow(Y, 1/3)
else
    fY = 7.787 * Y + 0.16/1.16
```

```
L = 1.16 * fY - 0.16

00436 {
00437     return 1.16f * IM_FWLAB(Y) - 0.16f;
00438 }
```

float imColorLightness2Luminance(const float & L) [inline]

Converts Lightness (CIE L*) to Luminance (CIE Y) (all linear).
The white point is D65.

```
0 <= Y <= 1 ; 0 <= L* <= 1

fY = (L + 0.16)/1.16
if (fY > 0.20689)
    Y = pow(fY, 3)
else
    Y = 0.1284 * (fY - 0.16/1.16)
Y = Y * 1.0          (for D65)

00454 {
00455     float fY = (L + 0.16f) / 1.16f;
00456     return IM_GWLAB(fY);
00457 }
```

HSI Color Coordinate System Conversions [Color Manipulation]

Detailed Description

HSI is just the RGB color space written in a different coordinate system.

"I" is the cube diagonal. HS is a polar coordinates of a plane normal to "I".
"S" is the distance from the diagonal. "H" is the angle from red vector.

This is not a new color space, this is exactly the same gamut as RGB.
Since it is still a cube, Smax depends on H.

See [im_colorhsi.h](#)

Functions

```
float imColorHSI_Smax (float h, double cosh, double sinh, float i)
float imColorHSI_ImaxS (float h, double cosh, double sinh)
void imColorRGB2HSI (float r, float g, float b, float *h, float *s, float *i)
void imColorRGB2HSIbyte (unsigned char r, unsigned char g,
    unsigned char b, float *h, float *s, float *i)
void imColorHSI2RGB (float h, float s, float i, float *r, float *g, float *b)
void imColorHSI2RGBbyte (float h, float s, float i, unsigned char *r,
    unsigned char *g, unsigned char *b)
```

Function Documentation

```
float imColorHSI_Smax( float h,  
                        double cosh,  
                        double sinh,  
                        float i  
)
```

Returns the maximum S for H (in radians) and I.

```
float imColorHSI_ImaxS( float h,  
                        double cosh,  
                        double sinh  
)
```

Returns I where S is maximum given H (in radians).

```
void imColorRGB2HSI( float r,  
                      float g,  
                      float b,  
                      float * h,  
                      float * s,  
                      float * i  
)
```

Converts from RGB to HSI.

```
void imColorRGB2HSIbyte( unsigned char r,  
                         unsigned char g,
```

	unsigned char	<i>b</i> ,
	float *	<i>h</i> ,
	float *	<i>s</i> ,
	float *	<i>i</i>
)

Converts from RGB (byte) to HSI.

void imColorHSI2RGB (float	<i>h</i> ,
	float	<i>s</i> ,
	float	<i>i</i> ,
	float *	<i>r</i> ,
	float *	<i>g</i> ,
	float *	<i>b</i>
)

Converts from HSI to RGB.

void imColorHSI2RGBbyte (float	<i>h</i> ,
	float	<i>s</i> ,
	float	<i>i</i> ,
	unsigned char *	<i>r</i> ,
	unsigned char *	<i>g</i> ,
	unsigned char *	<i>b</i>
)

Converts from HSI to RGB (byte).

Complex Numbers

[Utilities]

Detailed Description

See [im_complex.h](#)

Complex numbers operators.

Data Structures

class **imcfloat**

Complex Float Data Type. [More...](#)

Functions

```
<=> int operator<= (const imcfloat &C1, const imcfloat &C2)
<=> int operator<= (const imcfloat &C, const float &F)
imcfloat operator+ (const imcfloat &C1, const imcfloat &C2)
imcfloat operator+= (const imcfloat &C1, const imcfloat &C2)
imcfloat operator- (const imcfloat &C1, const imcfloat &C2)
imcfloat operator * (const imcfloat &C1, const imcfloat &C2)
imcfloat operator/ (const imcfloat &C1, const imcfloat &C2)
imcfloat operator/ (const imcfloat &C, const float &R)
imcfloat operator/= (const imcfloat &C, const float &R)
imcfloat operator * (const imcfloat &C, const float &R)

    int operator== (const imcfloat &C1, const imcfloat &C2)
    float cpxreal (const imcfloat &C)
    float cpximag (const imcfloat &C)
    float cpxmag (const imcfloat &C)
    float cpxphase (const imcfloat &C)
imcfloat cpxconj (const imcfloat &C)
imcfloat log (const imcfloat &C)
imcfloat exp (const imcfloat &C)
imcfloat pow (const imcfloat &C1, const imcfloat &C2)
imcfloat sqrt (const imcfloat &C)
imcfloat cpxpolar (const float &mag, const float &phase)
```

Counter

[Utilities]

Detailed Description

Used to notify the application that a step in the loading, saving or processing operation has been performed.

See [**im_counter.h**](#)

Typedefs

```
typedef int(* imCounterCallback )(int counter, void *user_data, const  
char *text, int progress)
```

Functions

```
imCounterCallback imCounterSetCallback (void *user_data,
                                         imCounterCallback counter_func)
    int imCounterBegin (const char *title)
    void imCounterEnd (int counter)
    int imCounterInc (int counter)
    void imCounterTotal (int counter, int total, const char
                         *message)
```

Typedef Documentation

```
typedef int(* imCounterCallback)(int counter, void *user_data, const
```

Counter callback, informs the progress of the operation to the client. Text contains a constant string that is NULL during normal counting, a title in the beginning of a sequence and a message in the beginning of a count. Counter id identifies different counters.

Progress in a count reports a value from 0 to 1000. If -1 indicates the start of a sequence of operations, 1001 ends the sequence.

If returns 0 the client should abort the operation.

If the counter is aborted, the callback will be called one last time at 1001.

Function Documentation

imCounterCallback	imCounterSetCallback(void * imCounterCallback cou)	use
--------------------------	---	-----

Changes the counter callback. Returns old callback.
User data is changed only if not NULL.

int imCounterBegin(const char * <i>title</i>)
--

Begins a new count, or a partial-count in a sequence.
Calls the callback with "-1" and text=title, if it is at the top level.
This is to be used by the operations. Returns a counter Id.

void imCounterEnd(int <i>counter</i>)
--

Ends a count, or a partial-count in a sequence.
Calls the callback with "1001", text=null, and releases the counter if it is
at top level count.

int imCounterInc(int <i>counter</i>)

Increments a count. Must set the total first.
Calls the callback, text=message if it is the first increment for the count.
Returns 0 if the callback aborted, 1 if returns normally.

void imCounterTotal(int <i>counter</i>, int <i>total</i>,

	const char * <i>message</i>
)	

Sets the total increments of a count.

Data Type Utilities

[Utilities]

Detailed Description

See [im_util.h](#)

Defines

```
#define IM_BYTECROP(_v) (_v < 0? 0: _v > 255? 255: _v)
#define IM_CROPMAX(_v, _max) (_v < 0? 0: _v > _max? _max: _v)
```

TypeDefs

```
typedef unsigned char imbyte  
typedef unsigned short imushort
```

Functions

```
int imDataTypeSize (int data_type)
const char * imDataTypeName (int data_type)
unsigned long imDataTypeIntMax (int data_type)
    long imDataTypeIntMin (int data_type)
```

Function Documentation

```
int imDataTypeSize( int data_type )
```

Returns the size in bytes of a specified numeric data type.

```
const char* imDataTypeName( int data_type )
```

Returns the numeric data type name given its identifier.

```
unsigned long imDataTypeIntMax( int data_type )
```

Returns the maximum value of an integer data type. For floating point returns 0.

```
long imDataTypeIntMin( int data_type )
```

Returns the minimum value of an integer data type. For floating point returns 0.

Data Compression Utilities

[Utilities]

Detailed Description

See [im_util.h](#)

Functions

```
int imCompressDataZ (const void *src_data, int src_size, void  
                      *dst_data, int dst_size, int zip_quality)  
int imCompressDataUnZ (const void *src_data, int src_size, void  
                      *dst_data, int dst_size)
```

Function Documentation

```
int imCompressDataZ( const void * src_data,  
                      int           src_size,  
                      void *        dst_data,  
                      int           dst_size,  
                      int           zip_quality  
)
```

Compresses the data using the ZLIB Deflate compression.
The destination buffer must be at least 0.1% larger than source_size
plus 12 bytes.
It compresses raw byte data. zip_quality can be 1 to 9.
Returns the size of the compressed buffer.

```
int imCompressDataUnZ( const void * src_data,  
                      int           src_size,  
                      void *        dst_data,  
                      int           dst_size  
)
```

Uncompresses the data compressed with the ZLIB Deflate
compression.

Math Utilities

[Utilities]

Detailed Description

See [im_color.h](#)

Functions

```
template<class T, class TU>
T imZeroOrderDecimation (int width, int height, T *map, float xl,
float yl, float box_width, float box_height, TU Dummy)
template<class T, class TU>
T imBilinearDecimation (int width, int height, T *map, float xl, float
yl, float box_width, float box_height, TU Dummy)
template<class T>
T imZeroOrderInterpolation (int width, int height, T *map, float xl,
float yl)
template<class T>
T imBilinearInterpolation (int width, int height, T *map, float xl, float
yl)
template<class T, class TU>
T imBicubicInterpolation (int width, int height, T *map, float xl, float
yl, TU Dummy)
template<class T>
void imMinMax (const T *map, int count, T &min, T &max)
```

Function Documentation

```
template<class T, class TU>
T imZeroOrderDecimation( int width,
                         int height,
                         T * map,
                         float xl,
                         float yl,
                         float box_width,
                         float box_height,
                         TU Dummy
) [inline]
```

Does Zero Order Decimation (Mean).

```
00046 {
00047     int x0,x1,y0,y1;
00048     (void)Dummy;
00049
00050     x0 = (int)floor(xl - box_width/2.0 - 0.5) + 1;
00051     y0 = (int)floor(yl - box_height/2.0 - 0.5) + 1;
00052     x1 = (int)floor(xl + box_width/2.0 - 0.5);
00053     y1 = (int)floor(yl + box_height/2.0 - 0.5);
00054
00055     if (x0 == x1) x1++;
00056     if (y0 == y1) y1++;
00057
00058     x0 = x0<0? 0: x0>width-1? width-1: x0;
00059     y0 = y0<0? 0: y0>height-1? height-1: y0;
00060     x1 = x1<0? 0: x1>width-1? width-1: x1;
00061     y1 = y1<0? 0: y1>height-1? height-1: y1;
00062
00063     TU Value;
00064     int Count = 0;
00065
00066     Value = 0;
00067
00068     for (int y = y0; y <= y1; y++)
```

```

00069  {
00070      for (int x = x0; x <= x1; x++)
00071      {
00072          Value += map[y*width+x];
00073          Count++;
00074      }
00075  }
00076
00077  if (Count == 0)
00078  {
00079      Value = 0;
00080      return (T)Value;
00081  }
00082
00083  return (T)(Value/(float)Count);
00084 }
```

template<class T, class TU>

T	imBilinearDecimation	(int	<i>width,</i>
			int	<i>height,</i>
			T *	<i>map,</i>
			float	<i>xl,</i>
			float	<i>yl,</i>
			float	<i>box_width,</i>
			float	<i>box_height,</i>
			TU	<i>Dummy</i>
)	[inline]	

Does Bilinear Decimation.

```

00090 {
00091     int x0,x1,y0,y1;
00092     (void)Dummy;
00093
00094     x0 = (int)floor(xl - box_width/2.0 - 0.5) + 1;
00095     y0 = (int)floor(yl - box_height/2.0 - 0.5) + 1;
00096     x1 = (int)floor(xl + box_width/2.0 - 0.5);
00097     y1 = (int)floor(yl + box_height/2.0 - 0.5);
00098
00099     if (x0 == x1) x1++;
```

```

00100    if (y0 == y1) y1++;
00101
00102    x0 = x0<0? 0: x0>width-1? width-1: x0;
00103    y0 = y0<0? 0: y0>height-1? height-1: y0;
00104    x1 = x1<0? 0: x1>width-1? width-1: x1;
00105    y1 = y1<0? 0: y1>height-1? height-1: y1;
00106
00107    TU Value, LineValue;
00108    float LineNorm, Norm, dxr, dyr;
00109
00110    Value = 0;
00111    Norm = 0;
00112
00113    for (int y = y0; y <= y1; y++)
00114    {
00115        dyr = y1 - (y+0.5f);
00116        if (dyr < 0) dyr *= -1;
00117
00118        LineValue = 0;
00119        LineNorm = 0;
00120
00121        for (int x = x0; x <= x1; x++)
00122        {
00123            dxr = x1 - (x+0.5f);
00124            if (dxr < 0) dxr *= -1;
00125
00126            LineValue += map[y*width+x] * dxr;
00127            LineNorm += dxr;
00128        }
00129
00130        Value += LineValue * dyr;
00131        Norm += dyr * LineNorm;
00132    }
00133
00134    if (Norm == 0)
00135    {
00136        Value = 0;
00137        return (T)Value;
00138    }
00139
00140    return (T)(Value/Norm);
00141 }
```

template<class T>

T imZeroOrderInterpolation(int *width*,

	int	<i>height,</i>
	T *	<i>map,</i>
	float	<i>xl,</i>
	float	<i>yl</i>
)	[inline]

Does Zero Order Interpolation (Nearest Neighborhood).

```

00147 {
00148     int x0 = (int)(xl-0.5f);
00149     int y0 = (int)(yl-0.5f);
00150     x0 = x0<0? 0: x0>width-1? width-1: x0;
00151     y0 = y0<0? 0: y0>height-1? height-1: y0;
00152     return map[y0*width + x0];
00153 }
```

template<class T>

T	imBilinearInterpolation (int	<i>width,</i>
		int	<i>height,</i>
		T *	<i>map,</i>
		float	<i>xl,</i>
		float	<i>yl</i>
)	[inline]	

Does Bilinear Interpolation.

```

00159 {
00160     int x0 = (int)(xl-0.5f);
00161     int y0 = (int)(yl-0.5f);
00162     int x1 = x0+1;
00163     int y1 = y0+1;
00164
00165     float t = xl - (x0+0.5f);
00166     float u = yl - (y0+0.5f);
00167
00168     x0 = x0<0? 0: x0>width-1? width-1: x0;
00169     y0 = y0<0? 0: y0>height-1? height-1: y0;
00170     x1 = x1<0? 0: x1>width-1? width-1: x1;
00171     y1 = y1<0? 0: y1>height-1? height-1: y1;
```

```

00172     T fll = map[y0*width + x0];
00173     T fhl = map[y0*width + x1];
00174     T flh = map[y1*width + x0];
00175     T fhh = map[y1*width + x1];
00176
00177
00178     return (T)((fhh - flh - fhl + fll) * u * t +
00179                  (fhl - fll) * t +
00180                  (flh - fll) * u +
00181                  fll);
00182 }
```

template<class T, class TU>

T	imBicubicInterpolation	(int	<i>width</i> ,
			int	<i>height</i> ,
			T *	<i>map</i> ,
			float	<i>xl</i> ,
			float	<i>yl</i> ,
			TU	<i>Dummy</i>
)	[inline]	

Does Bicubic Interpolation.

```

00188 {
00189     (void)Dummy;
00190
00191     int x0 = (int)(xl-0.5f);
00192     int y0 = (int)(yl-0.5f);
00193     int x1 = x0-1;
00194     int x2 = x0+2;
00195     int y1 = y0-1;
00196     int y2 = y0+2;
00197
00198     float t = xl - (x0+0.5f);
00199     float u = yl - (y0+0.5f);
00200
00201     x1 = x1<0? 0: x1>width-1? width-1: x1;
00202     y1 = y1<0? 0: y1>height-1? height-1: y1;
00203     x2 = x2<0? 0: x2>width-1? width-1: x2;
00204     y2 = y2<0? 0: y2>height-1? height-1: y2;
00205 }
```

```

00206     float CX[4], CY[4];
00207
00208     // Optimize calculations
00209     {
00210         float x, x2, x3;
00211
00212 #define C0 (-x3 + 2.0f*x2 - x)
00213 #define C1 ( x3 - 2.0f*x2 + 1.0f)
00214 #define C2 (-x3 + x2 + x)
00215 #define C3 ( x3 - x2)
00216
00217     x = t;
00218     x2 = x*x; x3 = x2*x;
00219     CX[0] = C0; CX[1] = C1; CX[2] = C2; CX[3] = C3;
00220
00221     x = u;
00222     x2 = x*x; x3 = x2*x;
00223     CY[0] = C0; CY[1] = C1; CY[2] = C2; CY[3] = C3;
00224 }
00225
00226 #undef C0
00227 #undef C1
00228 #undef C2
00229 #undef C3
00230
00231     TU LineValue, Value;
00232     float LineNorm, Norm;
00233
00234     Value = 0;
00235     Norm = 0;
00236
00237     for (int y = y1; y <= y2; y++)
00238     {
00239         LineValue = 0;
00240         LineNorm = 0;
00241
00242         for (int x = x1; x <= x2; x++)
00243         {
00244             LineValue += map[y*width+x] * CX[x-x1];
00245             LineNorm += CX[x-x1];
00246         }
00247
00248         Value += LineValue * CY[y-y1];
00249         Norm += CY[y-y1] * LineNorm;
00250     }
00251
00252     if (Norm == 0)

```

```

00253     {
00254         Value = 0;
00255         return (T)Value;
00256     }
00257
00258     Value = (Value/Norm);
00259
00260     int size = sizeof(T);
00261     if (size == 1)
00262         return (T)(Value<=0? 0: Value<=255? Value: 255);
00263     else
00264         return (T)(Value);
00265 }
```

template<class T>			
void imMinMax(const T * map,		
	int count,		
	T & min,		
	T & max		
) [inline]			

Calculates minimum and maximum values.

```

00271 {
00272     min = *map++;
00273     max = min;
00274     for (int i = 1; i < count; i++)
00275     {
00276         T value = *map++;
00277
00278         if (value > max)
00279             max = value;
00280         else if (value < min)
00281             min = value;
00282     }
00283 }
```

Palette Generators

[Utilities]

Detailed Description

Creates several standard palettes

See [im_palette.h](#)

Functions

```
int imPaletteFindNearest (const long *palette, int palette_count,  
                           long color)  
int imPaletteFindColor (const long *palette, int palette_count, long  
                        color, unsigned char tol)  
long * imPaletteGray (void)  
long * imPaletteRed (void)  
long * imPaletteGreen (void)  
long * imPaletteBlue (void)  
long * imPaletteYellow (void)  
long * imPaletteMagenta (void)  
long * imPaletteCian (void)  
long * imPaletteRainbow (void)  
long * imPaletteHues (void)  
long * imPaletteBlueIce (void)  
long * imPaletteHotIron (void)  
long * imPaletteBlackBody (void)  
long * imPaletteHighContrast (void)  
long * imPaletteUniform (void)  
    int imPaletteUniformIndex (long color)  
    int imPaletteUniformIndexHalftoned (long color, int x, int y)
```

Function Documentation

int imPaletteFindNearest (const long * palette,	
	int palette_count,	
	long color	
)		

Searches for the nearest color on the table and returns the color index if successful. It looks in all palette entries and finds the minimum euclidian square distance. If the color matches the given color it returns immediately.

int imPaletteFindColor (const long * palette,	
	int palette_count,	
	long color,	
	unsigned char tol	
)		

Searches for the color on the table and returns the color index if successful. If the tolerance is 0 search for the exact match in the palette else search for the first color that fits in the tolerance range.

long* imPaletteGray (void)

Creates a palette of gray scale values. The colors are arranged from black to white.

long* imPaletteRed (void)

Creates a palette of a gradient of red colors. The colors are arranged

from black to pure red.

long* imPaletteGreen(void)

Creates a palette of a gradient of green colors. The colors are arranged from black to pure green.

long* imPaletteBlue(void)

Creates a palette of a gradient of blue colors. The colors are arranged from black to pure blue.

long* imPaletteYellow(void)

Creates a palette of a gradient of yellow colors. The colors are arranged from black to pure yellow.

long* imPaletteMagenta(void)

Creates a palette of a gradient of magenta colors. The colors are arranged from black to pure magenta.

long* imPaletteCian(void)

Creates a palette of a gradient of cian colors. The colors are arranged from black to pure cian.

long* imPaletteRainbow(void)

Creates a palette of rainbow colors. The colors are arranged in the light wave length spectrum order (starting from purple).

```
long* imPaletteHues( void )
```

Creates a palette of hues with maximum saturation.

```
long* imPaletteBlueIce( void )
```

Creates a palette of a gradient of blue colors. The colors are arranged from pure blue to white.

```
long* imPaletteHotIron( void )
```

Creates a palette of a gradient from black to white passing through red and orange.

```
long* imPaletteBlackBody( void )
```

Creates a palette of a gradient from black to white passing through red and yellow.

```
long* imPaletteHighContrast( void )
```

Creates a palette with high contrast colors.

```
long* imPaletteUniform( void )
```

Creates a palette of an uniform range of colors from black to white. This is a $2^{(2.6)}$ bits per pixel palette.

```
int imPaletteUniformIndex( long color )
```

Returns the index of the correspondent RGB color of an uniform palette.

int	imPaletteUniformIndexHalftoned	(long	color ,
			int	x ,
			int	y
)		

Returns the index of the correspondent RGB color of an uniform palette. Uses an 8x8 ordered dither to lookup the index in a halftone matrix. The spatial position used by the halftone method.

String Utilities

[Utilities]

Detailed Description

See [im_util.h](#)

Functions

int **imStrEqual** (const char *str1, const char *str2)

int **imStrNLen** (const char *str, int max_len)

int **imStrCheck** (const void *data, int count)

Function Documentation

```
int imStrEqual( const char * str1,  
                 const char * str2  
               )
```

Check if the two strings are equal.

```
int imStrNLen( const char * str,  
                int          max_len  
              )
```

Calculate the size of the string but limited to max_len.

```
int imStrCheck( const void * data,  
                int          count  
              )
```

Check if the data is a string.

Windows DIB

[Utilities]

Detailed Description

Windows DIBs in memory are handled just like a BMP file without the file header.

These functions will work only in Windows. They are usefull for interchanging data with the clipboard, with capture drivers, with the AVI and WMF file formats and others.

Supported DIB aspects:

- bpp must be 1, 4, 8, 16, 24, or 32.
- BITMAPV4HEADER or BITMAPV5HEADER are handled but ignored.
- BITMAPCOREHEADER is not handled .
- BI_JPEG and BI_PNG compressions are not handled.
- biHeight can be negative, compression can be RLE only if created from imDibCreateReference, imDibPasteClipboard, imDibLoadFile.
- can not encode/decode Images to/from RLE compressed Dibs.
- if working with RLE Dibs bits_size is greater than used.
- the resolution of a new Dib is taken from the screen.
- SetDIBitsToDevice(start_scan is 0, scan_lines is dib->bmih->biHeight).
- StretchDIBits(use always DIB_RGB_COLORS).
- CreateDIBPatternBrushPt(packed_dib is dib->dib).

Must include <windows.h> before using these functions.

Check <wingdi.h> for structures and definitions.

See [im_dib.h](#)

Data Structures

struct [_imDib](#)

Windows DIB Structure. [More...](#)

TypeDefs

Functions

```
imDib * imDibCreate (int width, int height, int bpp)
imDib * imDibCreateCopy (const imDib *dib)
imDib * imDibCreateReference (BYTE *bmi, BYTE *bits)
imDib * imDibCreateSection (HDC hDC, HBITMAP
                           *image, int width, int height, int bpp)
void imDibDestroy (imDib *dib)

imDibLineGetPixel imDibLineGetPixelFunc (int bpp)
imDibLineSetPixel imDibLineSetPixelFunc (int bpp)
    imDib * imDibFromHBitmap (const HBITMAP image,
                               const HPALETTE hPalette)
HBITMAP imDibToHBitmap (const imDib *dib)
HPALETTE imDibLogicalPalette (const imDib *dib)
    imDib * imDibCaptureScreen (int x, int y, int width, int
                                 height)
void imDibCopyClipboard (imDib *dib)
imDib * imDibPasteClipboard (void)
    int imDibIsClipboardAvailable (void)
    int imDibSaveFile (const imDib *dib, const char
                       *filename)
imDib * imDibLoadFile (const char *filename)
    void imDibDecodeToRGBA (const imDib *dib,
                            unsigned char *red, unsigned char *green,
                            unsigned char *blue, unsigned char *alpha)
    void imDibDecodeToMap (const imDib *dib, unsigned
                           char *map, long *palette)
    void imDibEncodeFromRGBA (imDib *dib, const
                               unsigned char *red, const unsigned char *green,
                               const unsigned char *blue, const unsigned char
                               *alpha)
```

```
void imDibEncodeFromMap (imDib *dib, const  
    unsigned char *map, const long *palette, int  
    palette_count)  
void imDibEncodeFromBitmap (imDib *dib, const  
    unsigned char *data)  
void imDibDecodeToBitmap (const imDib *dib,  
    unsigned char *data)
```

Typedef Documentation

typedef struct _imDib imDib

Windows DIB Structure.

Handles a DIB in memory.

The DIB is stored in only one buffer. The secondary members are pointers to the main buffer.

typedef unsigned long(* imDibLineGetPixel)(unsigned char *line, int

DIB GetPixel function definition.

the ulong is a raw copy of the bits, use (unsigned char*)&pixel

typedef void(* imDibLineSetPixel)(unsigned char *line, int col, unsig

DIB SetPixel function definition

Function Documentation

```
imDib* imDibCreate( int width,  
                     int height,  
                     int bpp  
)
```

Creates a new DIB.
use bpp=-16/-32 to allocate space for BITFLIEDS.

```
imDib* imDibCreateCopy( const imDib * dib )
```

Duplicates the DIB contents in a new DIB.

```
imDib* imDibCreateReference( BYTE * bmi,  
                            BYTE * bits  
)
```

Creates a DIB using an already allocated memory.
"bmi" must be a pointer to BITMAPINFOHEADER.
"bits" can be NULL if it is inside "bmi" after the palette.

```
imDib* imDibCreateSection( HDC hDC,  
                         HBITMAP * image,  
                         int width,  
                         int height,  
                         int bpp  
)
```

Creates a DIB section for drawing porposes.
Returns the image handle also created.

void imDibDestroy (imDib * *dib*)

Destroy the DIB

imDibLineGetPixel imDibLineGetPixelFunc (int *bpp*)

Returns a function to read pixels from a DIB line.

imDibLineSetPixel imDibLineSetPixelFunc (int *bpp*)

Returns a function to write pixels into a DIB line.

imDib*	imDibFromHBitmap (const HBITMAP <i>image</i>,
	const HPALETTE <i>hPalette</i>
)

Creates a DIB from a image handle and a palette handle.

HBITMAP imDibToHBitmap (const imDib * *dib*)

Creates a image handle from a DIB.

HPALETTE imDibLogicalPalette (const imDib * *dib*)

Returns a Logical palette from the DIB palette.
DIB bpp must be <=8.

```
imDib* imDibCaptureScreen ( int x,  
                            int y,  
                            int width,  
                            int height  
                            )
```

Captures the screen into a DIB.

```
void imDibCopyClipboard ( imDib * dib )
```

Transfer the DIB to the clipboard.

"dib" pointer can not be used after, or use

imDibCopyClipboard(imDibCreateCopy(dib)). Warning: Clipboard functions in C++ can fail with Visual C++ /EHsc (Enable C++ Exceptions)

```
imDib* imDibPasteClipboard ( void )
```

Creates a reference for the DIB in the clipboard if any. Returns NULL otherwise. Warning: Clipboard functions in C++ can fail with Visual C++ /EHsc (Enable C++ Exceptions)

```
int imDibIsClipboardAvailable ( void )
```

Checks if there is a dib at the clipboard.

```
int imDibSaveFile ( const imDib * dib,  
                     const char * filename  
                     )
```

Saves the DIB into a file ".bmp".

```
imDib* imDibLoadFile ( const char * filename )
```

Creates a DIB from a file ".bmp".

```
void imDibDecodeToRGBA ( const imDib * dib,
                           unsigned char * red,
                           unsigned char * green,
                           unsigned char * blue,
                           unsigned char * alpha
)
```

Converts a DIB into an RGBA image. alpha is optional. bpp must be >8. alpha is used only when bpp=32.

```
void imDibDecodeToMap ( const imDib * dib,
                           unsigned char * map,
                           long * palette
)
```

Converts a DIB into an indexed image. bpp must be <=8. colors must have room for at least 256 colors. colors is rgb packed (RGBRGBRGB...)

```
void imDibEncodeFromRGBA ( imDib * dib,
                           const unsigned char * red,
                           const unsigned char * green,
                           const unsigned char * blue,
                           const unsigned char * alpha
)
```

Converts an RGBA image into a DIB. alpha is optional. bpp must be >8.

alpha is used only when bpp=32.

void imDibEncodeFromMap (imDib *	<i>dib,</i>
	const unsigned char *	<i>map,</i>
	const long *	<i>palette,</i>
	int	<i>palette_count</i>
)		

Converts an indexed image into a DIB. bpp must be <=8.
colors is rgb packed (RGBRGBRGB...)

void imDibEncodeFromBitmap (imDib *	<i>dib,</i>
	const unsigned char *	<i>data</i>
)		

Converts a IM_RGB packed image, with or without alpha, into a DIB.

void imDibDecodeToBitmap (const imDib *	<i>dib,</i>
	unsigned char *	<i>data</i>
)		

Converts a DIB into IM_RGB packed image, with or without alpha.

IM Data Structures

Here are the data structures with brief descriptions:

_imBinMemoryFileName	<i>Memory File I/O Filename</i>
_imDib	<i>Windows DIB Structure</i>
_imFile	<i>Image File Format Base (SDK Use Only)</i>
_imImage	<i>Image Structure Definition</i>
_imStats	
imAttribArray	<i>Attributes Table</i>
imAttribTable	<i>Attributes Table</i>
imcfloat	<i>Complex Float Data Type</i>
imFormat	<i>Image File Format Driver (SDK Use Only)</i>
imImageFile	<i>C++ Wrapper for the Image File Structure</i>

imAttribTable Class Reference

[Utilities]

Detailed Description

All the attributes have a name, a type, a count and the data.
Names are usually strings with less than 30 chars.

Attributes are stored in a hash table for fast access.
We use the hash function described in "The Practice of Programming" of
Kernighan & Pike.

Public Member Functions

```
imAttribTable (int hash_size)
~imAttribTable ()
int Count () const
void RemoveAll ()
void CopyFrom (const imAttribTable &table)
void Set (const char *name, int data_type, int count, const void
          *data)
void UnSet (const char *name)
const void * Get (const char *name, int *data_type=0, int *count=0)
               const
void ForEach (void *user_data, imAttribTableCallback
             attrib_func) const
```

Constructor & Destructor Documentation

imAttribTable::imAttribTable(int *hash_size*) [inline]

Creates an empty table. If size is zero the default size of 101 is used. Size must be a prime number. Other common values are 67, 599 and 1499.

```
00031     { ptable = imAttribTableCreate(hash_size); }
```

imAttribTable::~imAttribTable() [inline]

Destroys the table and all the attributes.

```
00035     { imAttribTableDestroy(ptable); ptable = 0; }
```

Member Function Documentation

int imAttribTable::Count() const [inline]

Returns the number of elements in the table.

```
00039     { return imAttribTableCount(ptable); }
```

void imAttribTable::RemoveAll() [inline]

Removes all the attributes in the table

```
00043     { imAttribTableRemoveAll(ptable); }
```

void imAttribTable::CopyFrom(const imAttribTable & table) [inli

Copies the contents of the given table into this table.

```
00047     { imAttribTableCopyFrom(ptable, table.ptable); }
```

void imAttribTable::Set(const char * name,		
	int	data_type,
	int	count,
	const void * data	
) [inline]

Inserts an attribute into the table.

Data is duplicated if not NULL, else data is initialized with zeros.

```
00052     { imAttribTableSet(ptable, name, data_type, count, data),
```

```
void imAttribTable::UnSet( const char * name ) [inline]
```

Removes an attribute from the table given its name.

```
00056 { imAttribTableUnSet(ptable, name); }
```

const void* imAttribTable::Get(const char * <i>name</i>,		
	int * <i>data_type</i> = 0,	
	int * <i>count</i> = 0	
) const [inline]	

Finds an attribute in the table. Returns the attribute if found, NULL otherwise.

```
00061 { return imAttribTableGet(ptable, name, data_type, count); }
```

void imAttribTable::ForEach(void * <i>user_data</i>,		
	imAttribTableCallback <i>attrib_func</i>	
) const [inline]	

For each attribute calls the user callback. If the callback returns 0 the function returns.

```
00065 { imAttribTableForEach(ptable, user_data, attrib_func); }
```

The documentation for this class was generated from the following file:

- [im_attrib.h](#)

_imBinMemoryFileName Struct

Reference

[Binary File Access]

Detailed Description

Fake file name for the memory I/O module.

Data Fields

```
unsigned char * buffer  
    int size  
    float reallocate
```

Field Documentation

unsigned char* _imBinMemoryFileName::buffer

The memory buffer. If you are reading the buffer must exists. If you are writing the buffer can be internally allocated to the given size. The buffer is never free.

int _imBinMemoryFileName::size

Size of the buffer.

float _imBinMemoryFileName::reallocate

Reallocate factor for the memory buffer when writing. size += reallocate*size.

The documentation for this struct was generated from the following file:

- [im_binfile.h](#)

_imDib Struct Reference

[Windows DIB]

Detailed Description

Handles a DIB in memory.

The DIB is stored in only one buffer. The secondary members are pointers to the main buffer.

Data Fields

```
HGLOBAL handle
    BYTE * dib
        int size
    BITMAPINFO * bmi
BITMAPINFOHEADER * bmih
    RGBQUAD * bmic
        BYTE * bits
            int palette_count
            int bits_size
            int line_size
            int pad_size
            int is_reference
```

Field Documentation

HGLOBAL _imDib::handle

The windows memory handle

BYTE* _imDib::dib

The DIB as it is defined in memory

int _imDib::size

Full size in memory

BITMAPINFO* _imDib::bmi

Bitmap Info = Bitmap Info Header + Palette

BITMAPINFOHEADER* _imDib::bmih

Bitmap Info Header

RGBQUAD* _imDib::bmic

Bitmap Info Colors = Palette

BYTE* _imDib::bits

Bitmap Bits

int _imDib::palette_count

number of colors in the palette

int _imDib::bits_size

size in bytes of the Bitmap Bits

int _imDib::line_size

size in bytes of one line, includes padding

int _imDib::pad_size

number of bytes remaining in the line, lines are in a word boundary

int _imDib::is_reference

only a reference, do not free pointer

The documentation for this struct was generated from the following file:

- [im_dib.h](#)

[_imFile Struct Reference](#) [\[File Format SDK\]](#)

Inherited by [imFormat](#).

Detailed Description

Base container to hold format independent state variables.

Data Fields

```
int is_new
void * attrib_table
void * line_buffer
    int line_buffer_size
    int line_buffer_extra
    int line_buffer_alloc
    int counter
    int convert_bpp
    int switch_type
long palette [256]
    int palette_count
    int user_color_mode
    int user_data_type
    int file_color_mode
    int file_data_type
char compression [10]
    int image_count
    int width
    int height
```

Field Documentation

`void* _imFile::attrib_table`

in fact is a `imAttribTable`, but we hide this here

`void* _imFile::line_buffer`

used for line conversion, contains all components if packed, or only one if not

`int _imFile::line_buffer_extra`

extra bytes to be allocated

`int _imFile::line_buffer_alloc`

total allocated so far

`int _imFile::convert_bpp`

number of bpp to expand or compact to/from 8bpp

`int _imFile::switch_type`

flag to switch the original data type: char-byte, short-ushort, uint-int, double-float

The documentation for this struct was generated from the following file:

- [**im_file.h**](#)

imFormat Class Reference

[File Format SDK]

Inherits [imFile](#).

Detailed Description

Virtual Base class for file formats. All file formats inherit from this class.

Public Member Functions

```
imFormat (const char *_format, const char *_desc,  
          const char *_ext, const char **_comp, int _comp_count,  
          int _can_sequence)  
imAttribTable * AttribTable ()  
    virtual int Open (const char *file_name)=0  
    virtual int New (const char *file_name)=0  
    virtual void Close ()=0  
    virtual void * Handle ()=0  
    virtual int ReadImageInfo (int index)=0  
    virtual int ReadImageData (void *data)=0  
    virtual int WriteImageInfo ()=0  
    virtual int WriteImageData (void *data)=0  
    virtual int CanWrite (const char *compression, int color_mode, int  
                      data_type) const =0
```

Data Fields

```
const char * format
const char * desc
const char * ext
const char ** comp
    int comp_count
    int can_sequence
```

The documentation for this class was generated from the following file:

- [**im_format.h**](#)

imcfloat Class Reference

[Complex Numbers]

Detailed Description

Complex class using two floats, one for real part, one for the imaginary part.

It is not a complete complex class, we just implement constructors inside the class. All the other operators and functions are external to the class.

Public Member Functions

imcfloat ()

imcfloat (const float &r, const float &i)

imcfloat (const float &r)

Data Fields

float **real**

float **imag**

Constructor & Destructor Documentation

imcfloat::imcfloat() [inline]

Default Constructor (0,0).

```
00034 :real(0), imag(0) {}
```

**imcfloat::imcfloat(const float & *r*,
 const float & *i*
) [inline]**

Constructor from (real, imag).

```
00037 :real(r), imag(i) {}
```

imcfloat::imcfloat(const float & *r*) [inline]

Constructor from (real).

```
00040 :real(r), imag(0) {}
```

Field Documentation

float imcfloat::real

Real part.

float imcfloat::imag

Imaginary part.

The documentation for this class was generated from the following file:

- **im_complex.h**

[_imImage Struct Reference](#) [\[Image Structure\]](#)

Detailed Description

An image representation than supports all the color spaces, but no alpha channel, planes are always unpacked and the orientation is always bottom up.

Data Fields

```
int width
int height
int color_space
int data_type
int depth
int line_size
int plane_size
int size
int count
void ** data
long * palette
    int palette_count
void * attrib_table
```

Field Documentation

int _imImage::width

Number of columns

int _imImage::height

Number of lines.

int _imImage::color_space

Color space descriptor.

int _imImage::data_type

Data type descriptor.

int _imImage::depth

Number of planes (ColorSpaceDepth)

int _imImage::line_size

Number of bytes per line in one plane (width * DataTypeSize)

int _imImage::plane_size

Number of bytes per plane. (line_size * height)

`int _imImage::size`

Number of bytes occupied by the image (plane_size * depth)

`int _imImage::count`

Number of pixels (width * height)

`void** _imImage::data`

Image data organized as a 2D matrix with several planes. But plane 0 is also a pointer to the full data. ($\text{data}[i] = \text{data}[0] + i * \text{plane_size}$)

`long* _imImage::palette`

Used when depth=1. Otherwise is NULL.

`int _imImage::palette_count`

The palette is always 256 colors allocated, but can have less colors used.

`void* _imImage::attrib_table`

in fact is a `imAttribTable`, but we hide this here

The documentation for this struct was generated from the following file:

- **im_image.h**

imImageFile Class Reference

[Image Storage]

Detailed Description

Usage is just like the C API. Open and New are replaced by equivalent constructors.

Close is replaced by the destructor. Error checking is done by the Error() member.

Open and New errors are checked using the Failed() member.

Public Member Functions

```
    imImageFile (const char *file_name, const char *format)
    imImageFile (const char *file_name)

    int Failed ()
    int Error ()

    void SetAttribute (const char *attrib, int data_type, int count,
                      const void *data)

    const void * GetAttribute (const char *attrib, int *data_type, int *count)

    void GetInfo (char *format, char *compression, int
                  *image_count)

    void ReadImageInfo (int index, int *width, int *height, int
                        *color_mode, int *data_type)

    void GetPalette (long *palette, int *palette_count)

    void ReadImageData (void *data, int convert2bitmap, int
                        color_mode_flags)

    void SetInfo (const char *compression)

    void SetPalette (long *palette, int palette_count)

    void WriteImageInfo (int width, int height, int color_mode, int
                        data_type)

    void WriteImageData (void *data)
```

The documentation for this class was generated from the following file:

- [im_plus.h](#)

IM File List

Here is a list of all documented files with brief descriptions:

im.h [code]	<i>Main API</i>
im_attrib.h [code]	<i>Attributes Table</i>
im_attrib_flat.h [code]	<i>Attributes Table Flat API. This will simplify the DLL export, and can be used for C applications</i>
im_binfile.h [code]	<i>Binary File Access</i>
im_capture.h [code]	<i>Video Capture</i>
im_color.h [code]	<i>Color Manipulation</i>
im_colorhsih.h [code]	<i>HSI Color Manipulation</i>
im_complex.h [code]	<i>Complex Data Type</i>
im_convert.h [code]	<i>Image Conversion</i>
im_counter.h [code]	<i>Processing Counter</i>
im_dib.h [code]	<i>Windows DIB (Device Independent Bitmap)</i>
im_file.h [code]	<i>File Access</i>
im_format.h [code]	<i>File Format Access</i>
im_format_all.h [code]	<i>All the Internal File Formats. They are all automatically registered by the library. The signatures are in C, but the functions are C++. Header for internal use only</i>
im_format_avi.h [code]	<i>Register the AVI Format</i>
im_format_jp2.h [code]	<i>Register the JP2 Format</i>

im_format_raw.h [code]	<i>Initialize the RAW Format Driver Header for internal use only</i>
im_format_wmv.h [code]	<i>Register the WMF Format</i>
im_image.h [code]	<i>Image Manipulation</i>
im_lib.h [code]	<i>Library Management and Main Documentation</i>
im_math.h [code]	<i>Math Utilities</i>
im_math_op.h [code]	<i>Math Operations</i>
im_palette.h [code]	<i>Palette Generators</i>
im_plus.h [code]	<i>C++ Wrapper for File Access</i>
im_process.h [code]	<i>Image Processing</i>
im_process_ana.h [code]	<i>Image Statistics and Analysis</i>
im_process_glo.h [code]	<i>Image Processing - Global Operations</i>
im_process_loc.h [code]	<i>Image Processing - Local Operations</i>
im_process_pon.h [code]	<i>Image Processing - Pontual Operations</i>
im_raw.h [code]	<i>RAW File Format</i>
im_util.h [code]	<i>Utilities</i>
imlua.h [code]	
old_im.h [code]	<i>Old API</i>

include

im.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Typedefs

```
typedef _imFile imFile
```

Enumerations

```
enum imDataType {
    IM_BYTE, IM USHORT, IM_INT, IM_FLOAT,
    IM_CFLOAT
}
enum imColorSpace {
    IM_RGB, IM_MAP, IM_GRAY, IM_BINARY,
    IM_CMYK, IM_YCBCR, IM_LAB, IM_LUV,
    IM_XYZ
}
enum imColorModeConfig { IM_ALPHA = 0x100, IM_PACKED =
    0x200, IM_TOPDOWN = 0x400 }
enum imErrorCodes {
    IM_ERR_NONE, IM_ERR_OPEN, IM_ERR_ACCESS,
    IM_ERR_FORMAT,
    IM_ERR_DATA, IM_ERR_COMPRESS, IM_ERR_MEM,
    IM_ERR_COUNTER
}
```

Functions

```
imFile * imFileOpen (const char *file_name, int *error)
imFile * imFileNew (const char *file_name, const char *format, int
    *error)
void imFileClose (imFile *ifile)
void * imFileHandle (imFile *ifile)
void imFileGetInfo (imFile *ifile, char *format, char
    *compression, int *image_count)
void imFileSetInfo (imFile *ifile, const char *compression)
void imFileSetAttribute (imFile *ifile, const char *attrib, int
    data_type, int count, const void *data)
const void * imFileGetAttribute (imFile *ifile, const char *attrib, int
    *data_type, int *count)
void imFileGetAttributeList (imFile *ifile, char **attrib, int
    *attrib_count)
void imFileGetPalette (imFile *ifile, long *palette, int
    *palette_count)
void imFileSetPalette (imFile *ifile, long *palette, int
    palette_count)
int imFileReadImageInfo (imFile *ifile, int index, int *width, int
    *height, int *file_color_mode, int *file_data_type)
int imFileWriteImageInfo (imFile *ifile, int width, int height, int
    user_color_mode, int user_data_type)
int imFileReadImageData (imFile *ifile, void *data, int
    convert2bitmap, int color_mode_flags)
int imFileWriteImageData (imFile *ifile, void *data)
void imFormatList (char **format_list, int *format_count)
int imFormatInfo (const char *format, char *desc, char *ext, int
    *can_sequence)
int imFormatCompressions (const char *format, char **comp,
```

```
    int *comp_count, int color_mode, int data_type)
int imFormatCanWriteImage (const char *format, const char
    *compression, int color_mode, int data_type)
```

include

im_attrib.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

class **imAttribTable**

Attributes Table. [More...](#)

class **imAttribArray**

Attributes Table. [More...](#)

include

im_attrib_flat.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Typedefs

```
typedef int(* imAttribTableCallback )(void *user_data, int index, const  
char *name, int data_type, int count, const void *data)
```

Functions

```
imAttribTablePrivate * imAttribTableCreate (int hash_size)
    void imAttribTableDestroy (imAttribTablePrivate
        *ptable)
        int imAttribTableCount (imAttribTablePrivate
            *ptable)
        void imAttribTableRemoveAll (imAttribTablePrivate
            *ptable)
    const void * imAttribTableGet (const imAttribTablePrivate
        *ptable, const char *name, int *data_type, int
        *count)
    void imAttribTableSet (imAttribTablePrivate *ptable,
        const char *name, int data_type, int count, const
        void *data)
    void imAttribTableUnSet (imAttribTablePrivate
        *ptable, const char *name)
    void imAttribTableCopyFrom (imAttribTablePrivate
        *ptable_dst, const imAttribTablePrivate
        *ptable_src)
    void imAttribTableForEach (const
        imAttribTablePrivate *ptable, void *user_data,
        imAttribTableCallback attrib_func)

imAttribTablePrivate * imAttribArrayCreate (int hash_size)
    const void * imAttribArrayGet (const imAttribTablePrivate
        *ptable, int index, char *name, int *data_type, int
        *count)
    void imAttribArraySet (imAttribTablePrivate *ptable,
        int index, const char *name, int data_type, int
        count, const void *data)
    void imAttribArrayCopyFrom (imAttribTablePrivate
        *ptable_dst, const imAttribTablePrivate
```

*ptable_src)

Typedef Documentation

```
typedef int(* imAttribTableCallback)(void *user_data, int index, cons
```

Definition of the callback used in ForEach function.

include

im_binfile.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

struct [_imBinMemoryFileName](#)
Memory File I/O Filename. [More...](#)

Typedefs

```
typedef _imBinFile imBinFile  
typedef _imBinMemoryFileName imBinMemoryFileName
```

Enumerations

```
enum imBinFileModule {  
    IM_RAWFILE, IM_STREAM, IM_MEMFILE, IM_SUBFILE,  
    IM_IOCUSTOM0  
}
```

Functions

```
imBinFile * imBinFileOpen (const char *pFileName)
imBinFile * imBinFileNew (const char *pFileName)
    void imBinFileClose (imBinFile *bfile)
        int imBinFileError (imBinFile *bfile)
unsigned long imBinFileSize (imBinFile *bfile)
    int imBinFileByteOrder (imBinFile *bfile, int pByteOrder)
unsigned long imBinFileRead (imBinFile *bfile, void *pValues, unsigned
    long pCount, int pSizeOf)
unsigned long imBinFileWrite (imBinFile *bfile, void *pValues, unsigned
    long pCount, int pSizeOf)
unsigned long imBinFilePrintf (imBinFile *bfile, char *format,...)
    void imBinFileSeekTo (imBinFile *bfile, unsigned long
        pOffset)
    void imBinFileSeekOffset (imBinFile *bfile, long pOffset)
    void imBinFileSeekFrom (imBinFile *bfile, long pOffset)
unsigned long imBinFileTell (imBinFile *bfile)
    int imBinFileEndOfFile (imBinFile *bfile)
    int imBinFileSetCurrentModule (int pModule)
```

include

im_capture.h File Reference

Detailed Description

See Copyright Notice in **im.h**

Id

Exp

[Go to the source code of this file.](#)

Typedefs

```
typedef _imVideoCapture imVideoCapture
```

Functions

```
int imVideoCaptureDeviceCount (void)
const char * imVideoCaptureDeviceDesc (int device)
    int imVideoCaptureReloadDevices (void)
imVideoCapture * imVideoCaptureCreate (void)
    void imVideoCaptureDestroy (imVideoCapture *vc)
    int imVideoCaptureConnect (imVideoCapture *vc, int
        device)
    void imVideoCaptureDisconnect (imVideoCapture *vc)
    int imVideoCaptureShowDialog (imVideoCapture *vc,
        int dialog, void *parent)
    int imVideoCaptureDialogCount (imVideoCapture *vc)
const char * imVideoCaptureDialogDesc (imVideoCapture *vc,
    int dialog)
    void imVideoCaptureGetImageSize (imVideoCapture *vc,
        int *width, int *height)
    int imVideoCaptureSetImageSize (imVideoCapture *vc,
        int width, int height)
    int imVideoCaptureFrame (imVideoCapture *vc,
        unsigned char *data, int color_mode, int timeout)
    int imVideoCaptureOneFrame (imVideoCapture *vc,
        unsigned char *data, int color_mode)
    int imVideoCaptureLive (imVideoCapture *vc, int live)
    int imVideoCaptureResetAttribute (imVideoCapture
        *vc, const char *attrib, int fauto)
    int imVideoCaptureGetAttribute (imVideoCapture *vc,
        const char *attrib, float *percent)
    int imVideoCaptureSetAttribute (imVideoCapture *vc,
        const char *attrib, float percent)
const char ** imVideoCaptureGetAttributeList (imVideoCapture
```

*vc, int *num_attrib)

include

im_color.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Defines

```
#define IM_FWLAB(_w)  
#define IM_GWLAB(_w)
```

Functions

```
float imColorZero (int data_type)
    int imColorMax (int data_type)
template<class T>
    T imColorQuantize (const float &value, const T &max)
template<class T>
float imColorReconstruct (const T &value, const T &max)
template<class T>
void imColorYCbCr2RGB (const T Y, const T Cb, const T Cr, T &R, T
    &G, T &B, const T &zero, const T &max)
template<class T>
void imColorRGB2YCbCr (const T R, const T G, const T B, T &Y, T
    &Cb, T &Cr, const T &zero)
template<class T>
void imColorCMYK2RGB (const T C, const T M, const T Y, const T K,
    T &R, T &G, T &B, const T &max)
template<class T>
void imColorXYZ2RGB (const T X, const T Y, const T Z, T &R, T &G, T
    &B, const T &max)
template<class T>
void imColorRGB2XYZ (const T R, const T G, const T B, T &X, T &Y, T
    &Z)
void imColorXYZ2Lab (const float X, const float Y, const float Z, float
    &L, float &a, float &b)
void imColorLab2XYZ (const float L, const float a, const float b, float
    &X, float &Y, float &Z)
void imColorXYZ2Luv (const float X, const float Y, const float Z, float
    &L, float &u, float &v)
void imColorLuv2XYZ (const float L, const float u, const float v, float
```

```
&X, float &Y, float &Z)
float imColorTransfer2Linear (const float &nonlinear_value)
float imColorTransfer2Nonlinear (const float &value)
void imColorRGB2RGBNonlinear (const float RL, const float GL, const
    float BL, float &R, float &G, float &B)
template<class T>
    T imColorRGB2Luma (const T R, const T G, const T B)
float imColorLuminance2Lightness (const float &Y)
float imColorLightness2Luminance (const float &L)
```

Define Documentation

```
#define IM_FWLAB( _w )
```

Value:

```
#define IM_GWLAB( _w )
```

Value:

include

im_colorhsi.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

include

im_complex.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

class **imcfloat**

Complex Float Data Type. [More...](#)

Functions

```
<=> int operator<= (const imcfloat &C1, const imcfloat &C2)
<=> int operator<= (const imcfloat &C, const float &F)
imcfloat operator+ (const imcfloat &C1, const imcfloat &C2)
imcfloat operator+= (const imcfloat &C1, const imcfloat &C2)
imcfloat operator- (const imcfloat &C1, const imcfloat &C2)
imcfloat operator * (const imcfloat &C1, const imcfloat &C2)
imcfloat operator/ (const imcfloat &C1, const imcfloat &C2)
imcfloat operator/ (const imcfloat &C, const float &R)
imcfloat operator/= (const imcfloat &C, const float &R)
imcfloat operator * (const imcfloat &C, const float &R)

    int operator== (const imcfloat &C1, const imcfloat &C2)
    float cpxreal (const imcfloat &C)
    float cpximag (const imcfloat &C)
    float cpxmag (const imcfloat &C)
    float cpxphase (const imcfloat &C)
imcfloat cpxconj (const imcfloat &C)
imcfloat log (const imcfloat &C)
imcfloat exp (const imcfloat &C)
imcfloat pow (const imcfloat &C1, const imcfloat &C2)
imcfloat sqrt (const imcfloat &C)
imcfloat cpxpolar (const float &mag, const float &phase)
```

include

im_convert.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Enumerations

```
enum imComplex2Real { IM_CPX_REAL, IM_CPX_IMAG,
    IM_CPX_MAG, IM_CPX_PHASE }

enum imGammaFactor {
    IM_GAMMA_LINEAR = 0, IM_GAMMA_LOGLITE = -10,
    IM_GAMMA_LOGHEAVY = -1000, IM_GAMMA_EXPLITE = 2,
    IM_GAMMA_EXPHEAVY = 7
}
enum imCastMode { IM_CAST_MINMAX, IM_CAST_FIXED,
    IM_CAST_DIRECT }
```

Functions

```
int imConvertDataType (const imImage *src_image, imImage  
*dst_image, int cpx2real, float gamma, int abssolute, int  
cast_mode)  
int imConvertColorSpace (const imImage *src_image, imImage  
*dst_image)  
int imConvertToBitmap (const imImage *src_image, imImage  
*dst_image, int cpx2real, float gamma, int abssolute, int  
cast_mode)  
void imConvertPacking (const void *src_data, void *dst_data, int width,  
int height, int depth, int data_type, int src_is_packed)  
void imConvertMapToRGB (unsigned char *data, int count, int depth,  
int packed, long *palette, int palette_count)  
int imConvertRGB2Map (int width, int height, unsigned char *red,  
unsigned char *green, unsigned char *blue, unsigned char *map,  
long *palette, int *palette_count)
```

include

im_counter.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Typedefs

```
typedef int(* imCounterCallback )(int counter, void *user_data, const  
char *text, int progress)
```

Functions

```
imCounterCallback imCounterSetCallback (void *user_data,
                                         imCounterCallback counter_func)
    int imCounterBegin (const char *title)
    void imCounterEnd (int counter)
    int imCounterInc (int counter)
    void imCounterTotal (int counter, int total, const char
                         *message)
```

include

im_dib.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

struct [_imDib](#)

Windows DIB Structure. [More...](#)

Typedefs

```
typedef _imDib imDib  
typedef unsigned long(*) imDibLineGetPixel )(unsigned char *line, int  
col)  
typedef void(* imDibLineSetPixel )(unsigned char *line, int  
col, unsigned long pixel)
```

Functions

```
imDib * imDibCreate (int width, int height, int bpp)
imDib * imDibCreateCopy (const imDib *dib)
imDib * imDibCreateReference (BYTE *bmi, BYTE *bits)
imDib * imDibCreateSection (HDC hDC, HBITMAP
                           *image, int width, int height, int bpp)
void imDibDestroy (imDib *dib)

imDibLineGetPixel imDibLineGetPixelFunc (int bpp)
imDibLineSetPixel imDibLineSetPixelFunc (int bpp)
    imDib * imDibFromHBitmap (const HBITMAP image,
                               const HPALETTE hPalette)
HBITMAP imDibToHBitmap (const imDib *dib)
HPALETTE imDibLogicalPalette (const imDib *dib)
    imDib * imDibCaptureScreen (int x, int y, int width, int
                                 height)
void imDibCopyClipboard (imDib *dib)
imDib * imDibPasteClipboard (void)
    int imDibIsClipboardAvailable (void)
    int imDibSaveFile (const imDib *dib, const char
                       *filename)
imDib * imDibLoadFile (const char *filename)
    void imDibDecodeToRGBA (const imDib *dib,
                            unsigned char *red, unsigned char *green,
                            unsigned char *blue, unsigned char *alpha)
    void imDibDecodeToMap (const imDib *dib, unsigned
                           char *map, long *palette)
    void imDibEncodeFromRGBA (imDib *dib, const
                               unsigned char *red, const unsigned char *green,
                               const unsigned char *blue, const unsigned char
                               *alpha)
```

```
void imDibEncodeFromMap (imDib *dib, const  
    unsigned char *map, const long *palette, int  
    palette_count)  
void imDibEncodeFromBitmap (imDib *dib, const  
    unsigned char *data)  
void imDibDecodeToBitmap (const imDib *dib,  
    unsigned char *data)
```

include

im_file.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

struct [_imFile](#)

Image File Format Base (SDK Use Only). [More...](#)

Functions

```
void imFileClear (imFile *ifile)
void imFileLineBufferInit (imFile *ifile)
int imFileCheckConversion (imFile *ifile)
int imFileLineBufferCount (imFile *ifile)
void imFileLineBufferInc (imFile *ifile, int *row, int *plane)
void imFileLineBufferRead (imFile *ifile, void *data, int line, int plane)
void imFileLineBufferWrite (imFile *ifile, const void *data, int line, int
plane)
int imFileLineSizeAligned (int width, int bpp, int align)
```

include

im_format.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

class **imFormat**

Image File Format Driver (SDK Use Only). [More...](#)

Typedefs

```
typedef imFormat *(* imFormatFunc )()
```

Functions

```
imFormat * imFormatOpen (const char *file_name, int *error)
imFormat * imFormatNew (const char *file_name, const char *format,
int *error)
void imFormatRegisterAll (void)
void imFormatRegister (imFormatFunc format_init)
```

include

im_format_all.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

```
void imFormatRegisterTIFF (void)
void imFormatRegisterJPEG (void)
void imFormatRegisterPNG (void)
void imFormatRegisterGIF (void)
void imFormatRegisterBMP (void)
void imFormatRegisterRAS (void)
void imFormatRegisterLED (void)
void imFormatRegisterSGI (void)
void imFormatRegisterPCX (void)
void imFormatRegisterTGA (void)
void imFormatRegisterPNM (void)
void imFormatRegisterICO (void)
void imFormatRegisterKRN (void)
```

include

im_format_avi.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

void **imFormatRegisterAVI** (void)

include

im_format_jp2.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

void **imFormatRegisterJP2** (void)

include

im_format_raw.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

imFormat * imFormatInitRAW (void)

include

im_format_wmv.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

void **imFormatRegisterWMV** (void)

include

im_image.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

struct [_imlImage](#)

Image Structure Definition. [More...](#)

Defines

```
#define cdPutBitmap(_image, _x, _y, _w, _h, _xmin, _xmax, _ymin,  
_ymax)
```

Typedefs

```
typedef _imImage imImage
```

Functions

```
imlImage * imlImageCreate (int width, int height, int color_space, int
                           data_type)
imlImage * imlImageInit (int width, int height, int color_space, int
                           data_type, void *data_buffer, long *palette, int
                           palette_count)
void imlImageDestroy (imlImage *image)
void imlImageReshape (imlImage *image, int width, int height)
void imlImageCopy (const imlImage *src_image, imlImage
                  *dst_image)
void imlImageCopyData (const imlImage *src_image, imlImage
                      *dst_image)
imlImage * imlImageDuplicate (const imlImage *image)
imlImage * imlImageClone (const imlImage *image)
void imlImageSetAttribute (imlImage *image, const char *attrib,
                           int data_type, int count, const void *data)
const void * imlImageGetAttribute (const imlImage *image, const char
                                   *attrib, int *data_type, int *count)
void imlImageGetAttributeList (const imlImage *image, char
                               **attrib, int *attrib_count)
void imlImageClear (imlImage *image)
int imlImageIsBitmap (const imlImage *image)
void imlImageSetPalette (imlImage *image, long *palette, int
                        palette_count)
void imlImageCopyAttributes (const imlImage *src_image,
                            imlImage *dst_image)
int imlImageMatchSize (const imlImage *image1, const
                      imlImage *image2)
int imlImageMatchColor (const imlImage *image1, const
                       imlImage *image2)
```

```
int imlImageMatchDataType (const imlImage *image1, const
                           imlImage *image2)
int imlImageMatchColorSpace (const imlImage *image1,
                            const imlImage *image2)
int imlImageMatch (const imlImage *image1, const imlImage
                  *image2)
imlImage * imFileLoadImage (imFile *ifile, int index, int *error)
imlImage * imFileLoadBitmap (imFile *ifile, int index, int *error)
           int imFileSaveImage (imFile *ifile, const imlImage *image)
imlImage * imlImageLoad (const char *file_name, int index, int *error)
imlImage * imlImageLoadBitmap (const char *file_name, int index, int
                               *error)
void imlImageSetBinary (imlImage *image)
void imlImageMakeBinary (imlImage *image)
```

include

im_lib.h File Reference

Detailed Description

See Copyright Notice in this file.

Id

Exp

[Go to the source code of this file.](#)

Defines

```
#define IM_AUTHOR "Antonio Scuri"  
#define IM_COPYRIGHT "Copyright (C) 1994-2004 Tecgraf/PUC-Rio  
and PETROBRAS S/A"  
#define IM_VERSION "3.0.3"  
#define IM_VERSION_DATE "2004/10/14"  
#define IM_VERSION_NUMBER 300003
```

Functions

```
const char * imVersion (void)
const char * imVersionDate (void)
    int imVersionNumber (void)
```

include

im_math.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Defines

```
#define C0 (-x3 + 2.0f*x2 - x)
#define C1 ( x3 - 2.0f*x2 + 1.0f)
#define C2 (-x3 + x2 + x)
#define C3 ( x3 - x2)
```

Functions

```
template<class T, class TU>
T imZeroOrderDecimation (int width, int height, T *map, float xl,
float yl, float box_width, float box_height, TU Dummy)
template<class T, class TU>
T imBilinearDecimation (int width, int height, T *map, float xl, float
yl, float box_width, float box_height, TU Dummy)
template<class T>
T imZeroOrderInterpolation (int width, int height, T *map, float xl,
float yl)
template<class T>
T imBilinearInterpolation (int width, int height, T *map, float xl, float
yl)
template<class T, class TU>
T imBicubicInterpolation (int width, int height, T *map, float xl, float
yl, TU Dummy)
template<class T>
void imMinMax (const T *map, int count, T &min, T &max)
```

include

im_math_op.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

```
template<class T>
    T crop_byte (const T &v)
template<class T1, class T2>
    T1 add_op (const T1 &v1, const T2 &v2)
template<class T1, class T2>
    T1 sub_op (const T1 &v1, const T2 &v2)
template<class T1, class T2>
    T1 mul_op (const T1 &v1, const T2 &v2)
template<class T1, class T2>
    T1 div_op (const T1 &v1, const T2 &v2)
template<class T>
    T inv_op (const T &v)
template<class T1, class T2>
    T1 diff_op (const T1 &v1, const T2 &v2)
template<class T1, class T2>
    T1 min_op (const T1 &v1, const T2 &v2)
template<class T1, class T2>
    T1 max_op (const T1 &v1, const T2 &v2)
template<class T1, class T2>
    T1 pow_op (const T1 &v1, const T2 &v2)
template<class T>
    T abs_op (const T &v)
template<class T>
    T less_op (const T &v)
template<class T>
    T sqr_op (const T &v)
    int sqrt (const int &C)
```

```
template<class T>
    T sqrt_op (const T &v)
    int exp (const int &v)
template<class T>
    T exp_op (const T &v)
    int log (const int &v)
template<class T>
    T log_op (const T &v)
imcfloat sin (const imcfloat &v)
    int sin (const int &v)
template<class T>
    T sin_op (const T &v)
    int cos (const int &v)
imcfloat cos (const imcfloat &v)
template<class T>
    T cos_op (const T &v)
    void imDataBitSet (imbyte *data, int index, int bit)
    int imDataBitGet (imbyte *data, int index)
```

Function Documentation

```
template<class T>
T crop_byte( const T & v ) [inline]
```

Crop value to Byte limit.

```
00019 {
00020     return v <= 0? 0: v <= 255? v: 255;
00021 }
```

```
template<class T1, class T2>
T1 add_op( const T1 & v1,
            const T2 & v2
        ) [inline]
```

Generic Addition with 2 template types.

```
00026 {
00027     return v2 + v1;
00028 }
```

```
template<class T1, class T2>
T1 sub_op( const T1 & v1,
            const T2 & v2
        ) [inline]
```

Generic Subtraction with 2 template types.

```
00033 {
00034     return v2 - v1;
00035 }
```

```
template<class T1, class T2>
```

```
T1 mul_op( const T1 & v1,  
           const T2 & v2  
         ) [inline]
```

Generic Multiplication with 2 template types.

```
00040 {  
00041   return v2 * v1;  
00042 }
```

```
template<class T1, class T2>
```

```
T1 div_op( const T1 & v1,  
           const T2 & v2  
         ) [inline]
```

Generic Division with 2 template types.

```
00047 {  
00048 //  if (v2 == 0) return (T1)IM_NEARINF;  
00049   return v1 / v2;  
00050 }
```

```
template<class T>
```

```
T inv_op( const T & v ) [inline]
```

Generic Invert.

```
00055 {  
00056 //  if (v == 0) return (T)IM_NEARINF;  
00057   return 1/v;  
00058 }
```

```
template<class T1, class T2>
```

```
T1 diff_op( const T1 & v1,
```

```
const T2 & v2
```

```
) [inline]
```

Generic Difference with 2 template types.

```
00063 {  
00064     if (v1 <= v2)  
00065         return v2 - v1;  
00066     return v1 - v2;  
00067 }
```

```
template<class T1, class T2>
```

```
T1 min_op ( const T1 & v1,  
            const T2 & v2  
        ) [inline]
```

Generic Minimum with 2 template types.

```
00072 {  
00073     if (v1 <= v2)  
00074         return v1;  
00075     return v2;  
00076 }
```

```
template<class T1, class T2>
```

```
T1 max_op ( const T1 & v1,  
            const T2 & v2  
        ) [inline]
```

Generic Maximum with 2 template types.

```
00081 {  
00082     if (v1 <= v2)  
00083         return v2;  
00084     return v1;  
00085 }
```

```
template<class T1, class T2>
```

```
T1 pow_op( const T1 & v1,  
           const T2 & v2  
         ) [inline]
```

Generic Power with 2 template types.

```
00090 {  
00091   return (T1)pow(v1, v2);  
00092 }
```

```
template<class T>
```

```
T abs_op( const T & v ) [inline]
```

Generic Abssolute.

```
00097 {  
00098   if (v <= 0)  
00099     return -1*v;  
00100   return v;  
00101 }
```

```
template<class T>
```

```
T less_op( const T & v ) [inline]
```

Generic Less.

```
00106 {  
00107   return -1*v;  
00108 }
```

```
template<class T>
```

```
T sqr_op( const T & v ) [inline]
```

Generic Square.

```
00113 {  
00114     return v*v;  
00115 }
```

```
template<class T>  
T sqrt_op( const T & v ) [inline]
```

Generic Square Root.

```
00125 {  
00126     return (T)sqrt(v);  
00127 }
```

```
template<class T>  
T exp_op( const T & v ) [inline]
```

Generic Exponential.

```
00137 {  
00138     return (T)exp(v);  
00139 }
```

```
template<class T>  
T log_op( const T & v ) [inline]
```

Generic Logarithm.

```
00149 {  
00150 // if (v <= 0) return (T)IM_NEARINF;  
00151     return (T)log(v);  
00152 }
```

```
template<class T>  
T sin_op( const T & v ) [inline]
```

Generic Sine.

```
00168 {  
00169     return (T)sin(v);  
00170 }
```

```
template<class T>  
T cos_op( const T & v ) [inline]
```

Generic Cosine.

```
00186 {  
00187     return (T)cos(v);  
00188 }
```

```
void imDataBitSet( imbyte * data,  
                    int      index,  
                    int      bit  
                ) [inline]
```

Sets a bit in an array.

```
00192 {  
00193     if (bit)  
00194         data[index / 8] |=  (0x01 << (7 - (index % 8)));  
00195     else  
00196         data[index / 8] &= ~(0x01 << (7 - (index % 8)));  
00197 }
```

```
int imDataBitGet( imbyte * data,  
                  int      index  
                ) [inline]
```

Gets a bit from an array.

```
00201 {
```

```
00202     return (data[index / 8] >> (7 - (index % 8))) & 0x01;  
00203 }
```

include

im_palette.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

```
int imPaletteFindNearest (const long *palette, int palette_count,  
                           long color)  
int imPaletteFindColor (const long *palette, int palette_count, long  
                        color, unsigned char tol)  
long * imPaletteGray (void)  
long * imPaletteRed (void)  
long * imPaletteGreen (void)  
long * imPaletteBlue (void)  
long * imPaletteYellow (void)  
long * imPaletteMagenta (void)  
long * imPaletteCian (void)  
long * imPaletteRainbow (void)  
long * imPaletteHues (void)  
long * imPaletteBlueIce (void)  
long * imPaletteHotIron (void)  
long * imPaletteBlackBody (void)  
long * imPaletteHighContrast (void)  
long * imPaletteUniform (void)  
    int imPaletteUniformIndex (long color)  
    int imPaletteUniformIndexHalftoned (long color, int x, int y)
```

include

im_plus.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

class [imImageFile](#)

C++ Wrapper for the Image File Structure. [More...](#)

include

im_process.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

include

im_raw.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

imFile * imFileOpenRaw (const char *file_name, int *error)

imFile * imFileNewRaw (const char *file_name, int *error)

include

im_util.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Defines

```
#define IM_MIN(_a, _b) (_a < _b? _a: _b)
#define IM_MAX(_a, _b) (_a > _b? _a: _b)
#define imColorModeSpace(_cm) (_cm & 0xFF)
#define imColorModeMatch(_cm1,
    _cm2) (imColorModeSpace(_cm1) ==
    imColorModeSpace(_cm2))
#define imColorModeHasAlpha(_cm) (_cm & IM_ALPHA)
#define imColorModelIsPacked(_cm) (_cm & IM_PACKED)
#define imColorModelIsTopDown(_cm) (_cm & IM_TOPDOWN)
#define IM_BYTECROP(_v) (_v < 0? 0: _v > 255? 255: _v)
#define IM_CROPMAX(_v, _max) (_v < 0? 0: _v > _max? _max: _v)
```

TypeDefs

```
typedef unsigned char imbyte  
typedef unsigned short imushort
```

Enumerations

```
enum imByteOrder { IM_LITTLEENDIAN, IM_BIGENDIAN }
```

Functions

```
int imStrEqual (const char *str1, const char *str2)
int imStrNLen (const char *str, int max_len)
int imStrCheck (const void *data, int count)
int imImageContentSize (int width, int height, int color_mode,
int data_type)
int imImageLineSize (int width, int color_mode, int
data_type)
int imImageLineCount (int width, int color_mode)
int imImageCheckFormat (int color_mode, int data_type)
long imColorEncode (unsigned char red, unsigned char
green, unsigned char blue)
void imColorDecode (unsigned char *red, unsigned char
*green, unsigned char *blue, long color)
const char * imColorModeSpaceName (int color_mode)
    int imColorModeDepth (int color_mode)
    int imColorModeToBitmap (int color_mode)
    int imColorModelsBitmap (int color_mode, int data_type)
    int imDataTypeSize (int data_type)
const char * imDataTypeName (int data_type)
unsigned long imDataTypeIntMax (int data_type)
    long imDataTypeIntMin (int data_type)
    int imBinCPUByteOrder (void)
void imBinSwapBytes (void *data, int count, int size)
void imBinSwapBytes2 (void *data, int count)
void imBinSwapBytes4 (void *data, int count)
void imBinSwapBytes8 (void *data, int count)
int imCompressDataZ (const void *src_data, int src_size,
void *dst_data, int dst_size, int zip_quality)
```

```
int imCompressDataUnZ (const void *src_data, int  
src_size, void *dst_data, int dst_size)
```

include

old_im.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Defines

```
#define IM_ERR_READ  IM_ERR_ACCESS
#define IM_ERR_WRITE  IM_ERR_ACCESS
#define IM_ERR_TYPE   IM_ERR_DATA
#define IM_ERR_COMP   IM_ERR_COMPRESS
#define IM_INTERRUPTED -1
#define IM_ALL -1
#define IM_COUNTER_CB 0
#define IM_RESOLUTION_CB 1
#define IM_GIF_TRANSPARENT_COLOR_CB 0
#define IM_TIF_IMAGE_DESCRIPTION_CB 0
```

Typedefs

```
typedef int(* imCallback )(char *filename)
typedef int(* imFileCounterCallback )(char *filename, int percent, int
                                         io)
typedef int(* imResolutionCallback )(char *filename, double *xres,
                                         double *yres, int *res_unit)
typedef int(* imGifTranspIndex )(char *filename, unsigned char
                                     *transp_index)
typedef int(* imTiffImageDesc )(char *filename, char *img_desc)
```

Enumerations

```
enum {  
    IM_BMP, IM_PCX, IM_GIF, IM_TIF,  
    IM_RAS, IM_SGI, IM_JPG, IM_LED,  
    IM_TGA  
}  
enum { IM_NONE = 0x0000, IM_DEFAULT = 0x0100,  
    IM_COMPRESSED = 0x0200 }  
enum { IM_RES_NONE, IM_RES_DPI, IM_RES_DPC }
```

Functions

```
long imEncodeColor (unsigned char red, unsigned char green,  
                     unsigned char blue)  
void imDecodeColor (unsigned char *red, unsigned char *green,  
                     unsigned char *blue, long palette)  
int imFileFormat (char *filename, int *format)  
int imImageInfo (char *filename, int *width, int *height, int *type, int  
                  *palette_count)  
int imLoadRGB (char *filename, unsigned char *red, unsigned char  
                  *green, unsigned char *blue)  
int imSaveRGB (int width, int height, int format, unsigned char *red,  
                  unsigned char *green, unsigned char *blue, char *filename)  
int imLoadMap (char *filename, unsigned char *map, long *palette)  
int imSaveMap (int width, int height, int format, unsigned char *map,  
                  int palette_count, long *palette, char *filename)  
void imRGB2Map (int width, int height, unsigned char *red, unsigned  
                  char *green, unsigned char *blue, unsigned char *map, int  
                  palette_count, long *palette)  
void imMap2RGB (int width, int height, unsigned char *map, int  
                  palette_count, long *colors, unsigned char *red, unsigned char  
                  *green, unsigned char *blue)  
void imRGB2Gray (int width, int height, unsigned char *red, unsigned  
                  char *green, unsigned char *blue, unsigned char *map, long  
                  *grays)  
void imMap2Gray (int width, int height, unsigned char *map, int  
                  palette_count, long *colors, unsigned char *grey_map, long *grays)  
void imResize (int src_width, int src_height, unsigned char *src_map,  
                  int dst_width, int dst_height, unsigned char *dst_map)  
void imStretch (int src_width, int src_height, unsigned char *src_map,  
                  int dst_width, int dst_height, unsigned char *dst_map)
```

```
int imRegisterCallback (imCallback cb, int cb_id, int format)
```

[All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumeration values](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

Here is a list of all documented functions, variables, defines, enums, and typedefs with links to the documentation:

- a -

- abs_op() : [im_math_op.h](#)
- add_op() : [im_math_op.h](#)

- C -

- cdPutBitmap : [im_image.h](#)
- cos_op() : [im_math_op.h](#)
- crop_byte() : [im_math_op.h](#)

- d -

- diff_op() : [im_math_op.h](#)
- div_op() : [im_math_op.h](#)

- e -

- exp_op() : [im_math_op.h](#)

- i -

- IM_ALPHA : [im.h](#)
- IM_BIGENDIAN : [im_util.h](#)
- IM_BIN_ADD : [im_process_pon.h](#)
- IM_BIN_DIFF : [im_process_pon.h](#)
- IM_BIN_DIV : [im_process_pon.h](#)
- IM_BIN_MAX : [im_process_pon.h](#)
- IM_BIN_MIN : [im_process_pon.h](#)
- IM_BIN_MUL : [im_process_pon.h](#)
- IM_BIN_POW : [im_process_pon.h](#)
- IM_BIN_SUB : [im_process_pon.h](#)
- IM_BINARY : [im.h](#)
- IM_BIT_AND : [im_process_pon.h](#)
- IM_BIT_OR : [im_process_pon.h](#)
- IM_BIT_XOR : [im_process_pon.h](#)
- IM_BYTE : [im.h](#)
- IM_CAST_DIRECT : [im_convert.h](#)
- IM_CAST_FIXED : [im_convert.h](#)
- IM_CAST_MINMAX : [im_convert.h](#)
- IM_CFLOAT : [im.h](#)
- IM_CMYK : [im.h](#)
- IM_ERR_ACCESS : [im.h](#)
- IM_ERR_COMPRESS : [im.h](#)
- IM_ERR_COUNTER : [im.h](#)
- IM_ERR_DATA : [im.h](#)
- IM_ERR_FORMAT : [im.h](#)
- IM_ERR_MEM : [im.h](#)
- IM_ERR_NONE : [im.h](#)
- IM_ERR_OPEN : [im.h](#)
- IM_FLOAT : [im.h](#)
- IM_GAMUT_BRIGHTCONT : [im_process_pon.h](#)
- IM_GAMUT_CROP : [im_process_pon.h](#)
- IM_GAMUT_EXP : [im_process_pon.h](#)
- IM_GAMUT_EXPAND : [im_process_pon.h](#)
- IM_GAMUT_INVERT : [im_process_pon.h](#)
- IM_GAMUT_LOG : [im_process_pon.h](#)
- IM_GAMUT_NORMALIZE : [im_process_pon.h](#)
- IM_GAMUT_POW : [im_process_pon.h](#)
- IM_GAMUT_SLICE : [im_process_pon.h](#)
- IM_GAMUT_SOLARIZE : [im_process_pon.h](#)

- IM_GAMUT_ZEROSTART : [im_process_pon.h](#)
- IM_GRAY : [im.h](#)
- IM_INT : [im.h](#)
- IM_IOCUSTOM0 : [im_binfile.h](#)
- IM_LAB : [im.h](#)
- IM_LITTLEENDIAN : [im_util.h](#)
- IM_LUV : [im.h](#)
- IM_MAP : [im.h](#)
- IM_MEMFILE : [im_binfile.h](#)
- IM_PACKED : [im.h](#)
- IM_RAWFILE : [im_binfile.h](#)
- IM_RGB : [im.h](#)
- IM_STREAM : [im_binfile.h](#)
- IM_SUBFILE : [im_binfile.h](#)
- IM_TOPDOWN : [im.h](#)
- IM_UN_ABS : [im_process_pon.h](#)
- IM_UN_CONJ : [im_process_pon.h](#)
- IM_UN_COS : [im_process_pon.h](#)
- IM_UN_CPXNORM : [im_process_pon.h](#)
- IM_UN_EQL : [im_process_pon.h](#)
- IM_UN_EXP : [im_process_pon.h](#)
- IM_UN_INC : [im_process_pon.h](#)
- IM_UN_INV : [im_process_pon.h](#)
- IM_UN_LESS : [im_process_pon.h](#)
- IM_UN_LOG : [im_process_pon.h](#)
- IM_UN_SIN : [im_process_pon.h](#)
- IM_UN_SQR : [im_process_pon.h](#)
- IM_UN_SQRT : [im_process_pon.h](#)
- IM USHORT : [im.h](#)
- IM_VERSION_NUMBER : [im_lib.h](#)
- IM_XYZ : [im.h](#)
- IM_YCBCR : [im.h](#)
- imAnalyzeFindRegions() : [im_process_ana.h](#)
- imAnalyzeMeasureArea() : [im_process_ana.h](#)
- imAnalyzeMeasureCentroid() : [im_process_ana.h](#)
- imAnalyzeMeasureHoles() : [im_process_ana.h](#)
- imAnalyzeMeasurePerimArea() : [im_process_ana.h](#)
- imAnalyzeMeasurePerimeter() : [im_process_ana.h](#)
- imAnalyzeMeasurePrincipalAxis() : [im_process_ana.h](#)

- imAttribTableCallback : [**im_attrib_flat.h**](#)
- imBicubicInterpolation() : [**im_math.h**](#)
- imBilinearDecimation() : [**im_math.h**](#)
- imBilinearInterpolation() : [**im_math.h**](#)
- imBinaryOp : [**im_process_pon.h**](#)
- imBinCPUByteOrder() : [**im_util.h**](#)
- imBinFileByteOrder() : [**im_binfile.h**](#)
- imBinFileClose() : [**im_binfile.h**](#)
- imBinFileEndOfFile() : [**im_binfile.h**](#)
- imBinFileError() : [**im_binfile.h**](#)
- imBinFileModule : [**im_binfile.h**](#)
- imBinFileNew() : [**im_binfile.h**](#)
- imBinFileOpen() : [**im_binfile.h**](#)
- imBinFilePrintf() : [**im_binfile.h**](#)
- imBinFileRead() : [**im_binfile.h**](#)
- imBinFileSeekFrom() : [**im_binfile.h**](#)
- imBinFileSeekOffset() : [**im_binfile.h**](#)
- imBinFileSeekTo() : [**im_binfile.h**](#)
- imBinFileSetCurrentModule() : [**im_binfile.h**](#)
- imBinFileSize() : [**im_binfile.h**](#)
- imBinFileTell() : [**im_binfile.h**](#)
- imBinFileWrite() : [**im_binfile.h**](#)
- imBinMemoryFileName : [**im_binfile.h**](#)
- imBinSwapBytes() : [**im_util.h**](#)
- imBinSwapBytes2() : [**im_util.h**](#)
- imBinSwapBytes4() : [**im_util.h**](#)
- imBinSwapBytes8() : [**im_util.h**](#)
- imByteOrder : [**im_util.h**](#)
- imCalcCountColors() : [**im_process_ana.h**](#)
- imCalcGrayHistogram() : [**im_process_ana.h**](#)
- imCalcHistogram() : [**im_process_ana.h**](#)
- imCalcHistogramStatistics() : [**im_process_ana.h**](#)
- imCalcHistoImageStatistics() : [**im_process_ana.h**](#)
- imCalcImageStatistics() : [**im_process_ana.h**](#)
- imCalcRMSError() : [**im_process_ana.h**](#)
- imCalcSNR() : [**im_process_ana.h**](#)
- imCalcUShortHistogram() : [**im_process_ana.h**](#)
- imCastMode : [**im_convert.h**](#)
- imColorCMYK2RGB() : [**im_color.h**](#)

- `imColorDecode()` : [**im_util.h**](#)
- `imColorEncode()` : [**im_util.h**](#)
- `imColorHSI2RGB()` : [**im_colorhsi.h**](#)
- `imColorHSI2RGBbyte()` : [**im_colorhsi.h**](#)
- `imColorHSI_ImaxS()` : [**im_colorhsi.h**](#)
- `imColorHSI_Smax()` : [**im_colorhsi.h**](#)
- `imColorLab2XYZ()` : [**im_color.h**](#)
- `imColorLightness2Luminance()` : [**im_color.h**](#)
- `imColorLuminance2Lightness()` : [**im_color.h**](#)
- `imColorLuv2XYZ()` : [**im_color.h**](#)
- `imColorMax()` : [**im_color.h**](#)
- `imColorModeConfig` : [**im.h**](#)
- `imColorModeDepth()` : [**im_util.h**](#)
- `imColorModeHasAlpha` : [**im_util.h**](#)
- `imColorModeIsBitmap()` : [**im_util.h**](#)
- `imColorModeIsPacked` : [**im_util.h**](#)
- `imColorModeIsTopDown` : [**im_util.h**](#)
- `imColorModeMatch` : [**im_util.h**](#)
- `imColorModeSpace` : [**im_util.h**](#)
- `imColorModeSpaceName()` : [**im_util.h**](#)
- `imColorModeToBitmap()` : [**im_util.h**](#)
- `imColorQuantize()` : [**im_color.h**](#)
- `imColorReconstruct()` : [**im_color.h**](#)
- `imColorRGB2HSI()` : [**im_colorhsi.h**](#)
- `imColorRGB2HSIbyte()` : [**im_colorhsi.h**](#)
- `imColorRGB2Luma()` : [**im_color.h**](#)
- `imColorRGB2RGBNonlinear()` : [**im_color.h**](#)
- `imColorRGB2XYZ()` : [**im_color.h**](#)
- `imColorRGB2YCbCr()` : [**im_color.h**](#)
- `imColorSpace` : [**im.h**](#)
- `imColorTransfer2Linear()` : [**im_color.h**](#)
- `imColorTransfer2Nonlinear()` : [**im_color.h**](#)
- `imColorXYZ2Lab()` : [**im_color.h**](#)
- `imColorXYZ2Luv()` : [**im_color.h**](#)
- `imColorXYZ2RGB()` : [**im_color.h**](#)
- `imColorYCbCr2RGB()` : [**im_color.h**](#)
- `imColorZero()` : [**im_color.h**](#)
- `imComplex2Real` : [**im_convert.h**](#)
- `imCompressDataUnZ()` : [**im_util.h**](#)

- imCompressDataZ() : [**im_util.h**](#)
- imConvertColorSpace() : [**im_convert.h**](#)
- imConvertDataType() : [**im_convert.h**](#)
- imConvertMapToRGB() : [**im_convert.h**](#)
- imConvertPacking() : [**im_convert.h**](#)
- imConvertToBitmap() : [**im_convert.h**](#)
- imCounterBegin() : [**im_counter.h**](#)
- imCounterCallback : [**im_counter.h**](#)
- imCounterEnd() : [**im_counter.h**](#)
- imCounterInc() : [**im_counter.h**](#)
- imCounterSetCallback() : [**im_counter.h**](#)
- imCounterTotal() : [**im_counter.h**](#)
- imDataBitGet() : [**im_math_op.h**](#)
- imDataBitSet() : [**im_math_op.h**](#)
- imDataType : [**im.h**](#)
- imDataTypeIntMax() : [**im_util.h**](#)
- imDataTypeIntMin() : [**im_util.h**](#)
- imDataTypeName() : [**im_util.h**](#)
- imDataTypeSize() : [**im_util.h**](#)
- imDib : [**im_dib.h**](#)
- imDibCaptureScreen() : [**im_dib.h**](#)
- imDibCopyClipboard() : [**im_dib.h**](#)
- imDibCreate() : [**im_dib.h**](#)
- imDibCreateCopy() : [**im_dib.h**](#)
- imDibCreateReference() : [**im_dib.h**](#)
- imDibCreateSection() : [**im_dib.h**](#)
- imDibDecodeToBitmap() : [**im_dib.h**](#)
- imDibDecodeToMap() : [**im_dib.h**](#)
- imDibDecodeToRGBA() : [**im_dib.h**](#)
- imDibDestroy() : [**im_dib.h**](#)
- imDibEncodeFromBitmap() : [**im_dib.h**](#)
- imDibEncodeFromMap() : [**im_dib.h**](#)
- imDibEncodeFromRGBA() : [**im_dib.h**](#)
- imDibFromHBitmap() : [**im_dib.h**](#)
- imDibIsClipboardAvailable() : [**im_dib.h**](#)
- imDibLineGetPixel : [**im_dib.h**](#)
- imDibLineGetPixelFunc() : [**im_dib.h**](#)
- imDibLineSetPixel : [**im_dib.h**](#)
- imDibLineSetPixelFunc() : [**im_dib.h**](#)

- imDibLoadFile() : **im_dib.h**
- imDibLogicalPalette() : **im_dib.h**
- imDibPasteClipboard() : **im_dib.h**
- imDibSaveFile() : **im_dib.h**
- imDibToHBitmap() : **im_dib.h**
- imErrorCodes : **im.h**
- imFileClose() : **im.h**
- imFileGetAttribute() : **im.h**
- imFileGetAttributeList() : **im.h**
- imFileGetInfo() : **im.h**
- imFileGetPalette() : **im.h**
- imFileHandle() : **im.h**
- imFileLineBufferCount() : **im_file.h**
- imFileLineBufferInc() : **im_file.h**
- imFileLineBufferRead() : **im_file.h**
- imFileLineBufferWrite() : **im_file.h**
- imFileLineSizeAligned() : **im_file.h**
- imFileLoadBitmap() : **im_image.h**
- imFileLoadImage() : **im_image.h**
- imFileNew() : **im.h**
- imFileNewRaw() : **im_raw.h**
- imFileOpen() : **im.h**
- imFileOpenRaw() : **im_raw.h**
- imFileReadImageData() : **im.h**
- imFileReadImageInfo() : **im.h**
- imFileSaveImage() : **im_image.h**
- imFileSetAttribute() : **im.h**
- imFileSetInfo() : **im.h**
- imFileSetPalette() : **im.h**
- imFileWriteImageData() : **im.h**
- imFileWriteImageInfo() : **im.h**
- imFormatCanWriteImage() : **im.h**
- imFormatCompressions() : **im.h**
- imFormatFunc : **im_format.h**
- imFormatInfo() : **im.h**
- imFormatList() : **im.h**
- imFormatRegister() : **im_format.h**
- imFormatRegisterAVI() : **im_format_avi.h**
- imFormatRegisterJP2() : **im_format_jp2.h**

- imFormatRegisterWMV() : [**im_format_wmv.h**](#)
- imGammaFactor : [**im_convert.h**](#)
- imGaussianStdDev2KernelSize() : [**im_process_loc.h**](#)
- imGaussianStdDev2Repetitions() : [**im_process_loc.h**](#)
- imImage : [**im_image.h**](#)
- imImageCheckFormat() : [**im_util.h**](#)
- imImageClear() : [**im_image.h**](#)
- imImageClone() : [**im_image.h**](#)
- imImageCopy() : [**im_image.h**](#)
- imImageCopyAttributes() : [**im_image.h**](#)
- imImageCopyData() : [**im_image.h**](#)
- imImageCreate() : [**im_image.h**](#)
- imImageDataSize() : [**im_util.h**](#)
- imImageDestroy() : [**im_image.h**](#)
- imImageDuplicate() : [**im_image.h**](#)
- imImageGetAttribute() : [**im_image.h**](#)
- imImageGetAttributeList() : [**im_image.h**](#)
- imImageInit() : [**im_image.h**](#)
- imImageIsBitmap() : [**im_image.h**](#)
- imImageLineCount() : [**im_util.h**](#)
- imImageLineSize() : [**im_util.h**](#)
- imImageLoad() : [**im_image.h**](#)
- imImageLoadBitmap() : [**im_image.h**](#)
- imImageMakeBinary() : [**im_image.h**](#)
- imImageMatch() : [**im_image.h**](#)
- imImageMatchColor() : [**im_image.h**](#)
- imImageMatchColorSpace() : [**im_image.h**](#)
- imImageMatchDataType() : [**im_image.h**](#)
- imImageMatchSize() : [**im_image.h**](#)
- imImageReshape() : [**im_image.h**](#)
- imImageSetAttribute() : [**im_image.h**](#)
- imImageSetBinary() : [**im_image.h**](#)
- imImageSetPalette() : [**im_image.h**](#)
- imLogicOp : [**im_process_pon.h**](#)
- imMinMax() : [**im_math.h**](#)
- imPaletteBlackBody() : [**im_palette.h**](#)
- imPaletteBlue() : [**im_palette.h**](#)
- imPaletteBlueIce() : [**im_palette.h**](#)
- imPaletteCian() : [**im_palette.h**](#)

- imPaletteFindColor() : [im_palette.h](#)
- imPaletteFindNearest() : [im_palette.h](#)
- imPaletteGray() : [im_palette.h](#)
- imPaletteGreen() : [im_palette.h](#)
- imPaletteHighContrast() : [im_palette.h](#)
- imPaletteHotIron() : [im_palette.h](#)
- imPaletteHues() : [im_palette.h](#)
- imPaletteMagenta() : [im_palette.h](#)
- imPaletteRainbow() : [im_palette.h](#)
- imPaletteRed() : [im_palette.h](#)
- imPaletteUniform() : [im_palette.h](#)
- imPaletteUniformIndex() : [im_palette.h](#)
- imPaletteUniformIndexHalftoned() : [im_palette.h](#)
- imPaletteYellow() : [im_palette.h](#)
- imProcessAddMargins() : [im_process_loc.h](#)
- imProcessArithmeticConstOp() : [im_process_pon.h](#)
- imProcessArithmeticOp() : [im_process_pon.h](#)
- imProcessAutoCorrelation() : [im_process_glo.h](#)
- imProcessAutoCovariance() : [im_process_pon.h](#)
- imProcessBinMorphClose() : [im_process_loc.h](#)
- imProcessBinMorphConvolve() : [im_process_loc.h](#)
- imProcessBinMorphDilate() : [im_process_loc.h](#)
- imProcessBinMorphErode() : [im_process_loc.h](#)
- imProcessBinMorphOpen() : [im_process_loc.h](#)
- imProcessBinMorphOutline() : [im_process_loc.h](#)
- imProcessBinMorphThin() : [im_process_loc.h](#)
- imProcessBitMask() : [im_process_pon.h](#)
- imProcessBitPlane() : [im_process_pon.h](#)
- imProcessBitwiseNot() : [im_process_pon.h](#)
- imProcessBitwiseOp() : [im_process_pon.h](#)
- imProcessBlend() : [im_process_pon.h](#)
- imProcessCalcRotateSize() : [im_process_loc.h](#)
- imProcessCanny() : [im_process_loc.h](#)
- imProcessCompassConvolve() : [im_process_loc.h](#)
- imProcessConvolve() : [im_process_loc.h](#)
- imProcessConvolveRep() : [im_process_loc.h](#)
- imProcessCrop() : [im_process_loc.h](#)
- imProcessCrossCorrelation() : [im_process_glo.h](#)
- imProcessDiffOfGaussianConvolve() : [im_process_loc.h](#)

- imProcessDiffOfGaussianConvolveRep() : [**im_process_loc.h**](#)
- imProcessDifusionErrThreshold() : [**im_process_pon.h**](#)
- imProcessDirectConv() : [**im_process_pon.h**](#)
- imProcessDistanceTransform() : [**im_process_glo.h**](#)
- imProcessEqualizeHistogram() : [**im_process_pon.h**](#)
- imProcessExpandHistogram() : [**im_process_pon.h**](#)
- imProcessFFT() : [**im_process_glo.h**](#)
- imProcessFFTraw() : [**im_process_glo.h**](#)
- imProcessFillHoles() : [**im_process_ana.h**](#)
- imProcessFlip() : [**im_process_loc.h**](#)
- imProcessGaussianConvolve() : [**im_process_loc.h**](#)
- imProcessGaussianConvolveRep() : [**im_process_loc.h**](#)
- imProcessGrayMorphClose() : [**im_process_loc.h**](#)
- imProcessGrayMorphConvolve() : [**im_process_loc.h**](#)
- imProcessGrayMorphDilate() : [**im_process_loc.h**](#)
- imProcessGrayMorphErode() : [**im_process_loc.h**](#)
- imProcessGrayMorphGradient() : [**im_process_loc.h**](#)
- imProcessGrayMorphOpen() : [**im_process_loc.h**](#)
- imProcessGrayMorphTopHat() : [**im_process_loc.h**](#)
- imProcessGrayMorphWell() : [**im_process_loc.h**](#)
- imProcessHoughLines() : [**im_process_glo.h**](#)
- imProcessHoughLinesDraw() : [**im_process_glo.h**](#)
- imProcessHysteresisThresEstimate() : [**im_process_pon.h**](#)
- imProcessHysteresisThreshold() : [**im_process_pon.h**](#)
- imProcessIFFT() : [**im_process_glo.h**](#)
- imProcessLapOfGaussianConvolve() : [**im_process_loc.h**](#)
- imProcessLocalMaxThreshold() : [**im_process_loc.h**](#)
- imProcessLocaMaxThresEstimate() : [**im_process_pon.h**](#)
- imProcessMeanConvolve() : [**im_process_loc.h**](#)
- imProcessMedianConvolve() : [**im_process_loc.h**](#)
- imProcessMergeComplex() : [**im_process_pon.h**](#)
- imProcessMergeComponents() : [**im_process_pon.h**](#)
- imProcessMergeHSI() : [**im_process_pon.h**](#)
- imProcessMinMaxThreshold() : [**im_process_pon.h**](#)
- imProcessMirror() : [**im_process_loc.h**](#)
- imProcessMultipleMean() : [**im_process_pon.h**](#)
- imProcessMultipleStdDev() : [**im_process_pon.h**](#)
- imProcessMultiplyConj() : [**im_process_pon.h**](#)
- imProcessNegative() : [**im_process_pon.h**](#)

- imProcessNormalizeComponents() : [**im_process_pon.h**](#)
- imProcessOtsuThreshold() : [**im_process_pon.h**](#)
- imProcessPercentThreshold() : [**im_process_pon.h**](#)
- imProcessPerimeterLine() : [**im_process_ana.h**](#)
- imProcessPixelate() : [**im_process_pon.h**](#)
- imProcessPosterize() : [**im_process_pon.h**](#)
- imProcessPrune() : [**im_process_ana.h**](#)
- imProcessQuantizeGrayUniform() : [**im_process_pon.h**](#)
- imProcessQuantizeRGBUniform() : [**im_process_pon.h**](#)
- imProcessRadial() : [**im_process_loc.h**](#)
- imProcessRangeContrastThreshold() : [**im_process_loc.h**](#)
- imProcessRangeConvolve() : [**im_process_loc.h**](#)
- imProcessRankClosestConvolve() : [**im_process_loc.h**](#)
- imProcessRankMaxConvolve() : [**im_process_loc.h**](#)
- imProcessRankMinConvolve() : [**im_process_loc.h**](#)
- imProcessReduce() : [**im_process_loc.h**](#)
- imProcessReduceBy4() : [**im_process_loc.h**](#)
- imProcessRegionalMaximum() : [**im_process_glo.h**](#)
- imProcessRenderAddGaussianNoise() : [**im_process_pon.h**](#)
- imProcessRenderAddSpeckleNoise() : [**im_process_pon.h**](#)
- imProcessRenderAddUniformNoise() : [**im_process_pon.h**](#)
- imProcessRenderBox() : [**im_process_pon.h**](#)
- imProcessRenderChessboard() : [**im_process_pon.h**](#)
- imProcessRenderCondOp() : [**im_process_pon.h**](#)
- imProcessRenderCone() : [**im_process_pon.h**](#)
- imProcessRenderConstant() : [**im_process_pon.h**](#)
- imProcessRenderCosine() : [**im_process_pon.h**](#)
- imProcessRenderGaussian() : [**im_process_pon.h**](#)
- imProcessRenderGrid() : [**im_process_pon.h**](#)
- imProcessRenderLapOfGaussian() : [**im_process_pon.h**](#)
- imProcessRenderOp() : [**im_process_pon.h**](#)
- imProcessRenderRamp() : [**im_process_pon.h**](#)
- imProcessRenderRandomNoise() : [**im_process_pon.h**](#)
- imProcessRenderSinc() : [**im_process_pon.h**](#)
- imProcessRenderTent() : [**im_process_pon.h**](#)
- imProcessRenderWheel() : [**im_process_pon.h**](#)
- imProcessReplaceColor() : [**im_process_pon.h**](#)
- imProcessResize() : [**im_process_loc.h**](#)
- imProcessRotate() : [**im_process_loc.h**](#)

- imProcessRotate180() : [**im_process_loc.h**](#)
- imProcessRotate90() : [**im_process_loc.h**](#)
- imProcessRotateKernel() : [**im_process_loc.h**](#)
- imProcessSliceThreshold() : [**im_process_pon.h**](#)
- imProcessSobelConvolve() : [**im_process_loc.h**](#)
- imProcessSplitComplex() : [**im_process_pon.h**](#)
- imProcessSplitComponents() : [**im_process_pon.h**](#)
- imProcessSplitHSI() : [**im_process_pon.h**](#)
- imProcessSplitYChroma() : [**im_process_pon.h**](#)
- imProcessSwapQuadrants() : [**im_process_glo.h**](#)
- imProcessThreshold() : [**im_process_pon.h**](#)
- imProcessThresholdByDiff() : [**im_process_pon.h**](#)
- imProcessToneGamut() : [**im_process_pon.h**](#)
- imProcessUnArithmeticOp() : [**im_process_pon.h**](#)
- imProcessUniformErrThreshold() : [**im_process_pon.h**](#)
- imProcessUnNormalize() : [**im_process_pon.h**](#)
- imProcessZeroCrossing() : [**im_process_loc.h**](#)
- imRenderCondFunc : [**im_process_pon.h**](#)
- imRenderFunc : [**im_process_pon.h**](#)
- imStats : [**im_process_ana.h**](#)
- imStrCheck() : [**im_util.h**](#)
- imStrEqual() : [**im_util.h**](#)
- imStrNLen() : [**im_util.h**](#)
- imToneGamut : [**im_process_pon.h**](#)
- imUnaryOp : [**im_process_pon.h**](#)
- imVersion() : [**im_lib.h**](#)
- imVersionDate() : [**im_lib.h**](#)
- imVersionNumber() : [**im_lib.h**](#)
- imVideoCaptureConnect() : [**im_capture.h**](#)
- imVideoCaptureCreate() : [**im_capture.h**](#)
- imVideoCaptureDestroy() : [**im_capture.h**](#)
- imVideoCaptureDeviceCount() : [**im_capture.h**](#)
- imVideoCaptureDeviceDesc() : [**im_capture.h**](#)
- imVideoCaptureDialogCount() : [**im_capture.h**](#)
- imVideoCaptureDialogDesc() : [**im_capture.h**](#)
- imVideoCaptureDisconnect() : [**im_capture.h**](#)
- imVideoCaptureFrame() : [**im_capture.h**](#)
- imVideoCaptureGetAttribute() : [**im_capture.h**](#)
- imVideoCaptureGetAttributeList() : [**im_capture.h**](#)

- imVideoCaptureGetImageSize() : [im_capture.h](#)
- imVideoCaptureLive() : [im_capture.h](#)
- imVideoCaptureOneFrame() : [im_capture.h](#)
- imVideoCaptureReloadDevices() : [im_capture.h](#)
- imVideoCaptureResetAttribute() : [im_capture.h](#)
- imVideoCaptureSetAttribute() : [im_capture.h](#)
- imVideoCaptureSetImageSize() : [im_capture.h](#)
- imVideoCaptureShowDialog() : [im_capture.h](#)
- imZeroOrderDecimation() : [im_math.h](#)
- imZeroOrderInterpolation() : [im_math.h](#)
- inv_op() : [im_math_op.h](#)

- | -

- less_op() : [im_math_op.h](#)
- log_op() : [im_math_op.h](#)

- m -

- max_op() : [im_math_op.h](#)
- min_op() : [im_math_op.h](#)
- mul_op() : [im_math_op.h](#)

- p -

- pow_op() : [im_math_op.h](#)

- S -

- sin_op() : [im_math_op.h](#)

- `sqr_op()` : **im_math_op.h**
- `sqrt_op()` : **im_math_op.h**
- `sub_op()` : **im_math_op.h**

include

im_process_pon.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Typedefs

```
typedef float(* imRenderFunc )(int x, int y, int d, float *param)  
typedef float(* imRenderCondFunc )(int x, int y, int d, int *cond, float  
*param)
```

Enumerations

```
enum imUnaryOp {
    IM_UN_EQL, IM_UN_ABS, IM_UN_LESS, IM_UN_INC,
    IM_UN_INV, IM_UN_SQR, IM_UN_SQRT, IM_UN_LOG,
    IM_UN_EXP, IM_UN_SIN, IM_UN_COS, IM_UN_CONJ,
    IM_UN_CPNORM
}
enum imBinaryOp {
    IM_BIN_ADD, IM_BIN_SUB, IM_BIN_MUL, IM_BIN_DIV,
    IM_BIN_DIFF, IM_BIN_POW, IM_BIN_MIN, IM_BIN_MAX
}
enum imLogicOp { IM_BIT_AND, IM_BIT_OR, IM_BIT_XOR }
enum imToneGamut {
    IM_GAMUT_NORMALIZE, IM_GAMUT_POW,
    IM_GAMUT_LOG, IM_GAMUT_EXP,
    IM_GAMUT_INVERT, IM_GAMUT_ZEROSTART,
    IM_GAMUT_SOLARIZE, IM_GAMUT_SLICE,
    IM_GAMUT_EXPAND, IM_GAMUT_CROP,
    IM_GAMUT_BRIGHTCONT
}
```

Functions

```
void imProcessUnArithmeticOp (const imlImage *src_image,
    imlImage *dst_image, int op)
void imProcessArithmeticOp (const imlImage *src_image1, const
    imlImage *src_image2, imlImage *dst_image, int op)
void imProcessArithmeticConstOp (const imlImage *src_image, float
    src_const, imlImage *dst_image, int op)
void imProcessBlend (const imlImage *src_image1, imlImage
    *src_image2, imlImage *dst_image, float alpha)
void imProcessSplitComplex (const imlImage *src_image, imlImage
    *dst_image1, imlImage *dst_image2, int polar)
void imProcessMergeComplex (const imlImage *src_image1, const
    imlImage *src_image2, imlImage *dst_image, int polar)
void imProcessMultipleMean (const imlImage **src_image_list, int
    src_image_count, imlImage *dst_image)
void imProcessMultipleStdDev (const imlImage **src_image_list, int
    src_image_count, const imlImage *mean_image, imlImage
    *dst_image)
int imProcessAutoCovariance (const imlImage *src_image, const
    imlImage *mean_image, imlImage *dst_image)
void imProcessMultiplyConj (const imlImage *src_image1, const
    imlImage *src_image2, imlImage *dst_image)
void imProcessQuantizeRGBUniform (const imlImage *src_image,
    imlImage *dst_image, int dither)
void imProcessQuantizeGrayUniform (const imlImage *src_image,
    imlImage *dst_image, int grays)
void imProcessExpandHistogram (const imlImage *src_image,
    imlImage *dst_image, float percent)
void imProcessEqualizeHistogram (const imlImage *src_image,
    imlImage *dst_image)
```

```
void imProcessSplitYChroma (const imlImage *src_image, imlImage
    *y_image, imlImage *chroma_image)
void imProcessSplitHSI (const imlImage *src_image, imlImage
    *h_image, imlImage *s_image, imlImage *i_image)
void imProcessMergeHSI (const imlImage *h_image, const imlImage
    *s_image, const imlImage *i_image3, imlImage *dst_image)
void imProcessSplitComponents (const imlImage *src_image,
    imlImage **dst_image)
void imProcessMergeComponents (const imlImage **src_image_list,
    imlImage *dst_image)
void imProcessNormalizeComponents (const imlImage *src_image,
    imlImage *dst_image)
void imProcessReplaceColor (const imlImage *src_image, imlImage
    *dst_image, float *src_color, float *dst_color)
void imProcessBitwiseOp (const imlImage *src_image1, const
    imlImage *src_image2, imlImage *dst_image, int op)
void imProcessBitwiseNot (const imlImage *src_image, imlImage
    *dst_image)
void imProcessBitMask (const imlImage *src_image, imlImage
    *dst_image, unsigned char mask, int op)
void imProcessBitPlane (const imlImage *src_image, imlImage
    *dst_image, int plane, int reset)
int imProcessRenderOp (imlImage *image, imRenderFunc
    render_func, char *render_name, float *param, int plus)
int imProcessRenderCondOp (imlImage *image,
    imRenderCondFunc render_func, char *render_name, float
    *param)
int imProcessRenderAddSpeckleNoise (const imlImage *src_image,
    imlImage *dst_image, float percent)
int imProcessRenderAddGaussianNoise (const imlImage
    *src_image, imlImage *dst_image, float mean, float stddev)
int imProcessRenderAddUniformNoise (const imlImage
    *src_image, imlImage *dst_image, float mean, float stddev)
int imProcessRenderRandomNoise (imlImage *image)
```

```
int imProcessRenderConstant (imImage *image, float *value)
int imProcessRenderWheel (imImage *image, int int_radius, int ext_radius)
int imProcessRenderCone (imImage *image, int radius)
int imProcessRenderTent (imImage *image, int width, int height)
int imProcessRenderRamp (imImage *image, int start, int end, int dir)
int imProcessRenderBox (imImage *image, int width, int height)
int imProcessRenderSinc (imImage *image, float xperiod, float yperiod)
int imProcessRenderGaussian (imImage *image, float stddev)
int imProcessRenderLapOfGaussian (imImage *image, float stddev)
int imProcessRenderCosine (imImage *image, float xperiod, float yperiod)
int imProcessRenderGrid (imImage *image, int x_space, int y_space)
int imProcessRenderChessboard (imImage *image, int x_space, int y_space)
void imProcessToneGamut (const imImage *src_image, imImage *dst_image, int op, float *param)
void imProcessUnNormalize (const imImage *src_image, imImage *dst_image)
void imProcessDirectConv (const imImage *src_image, imImage *dst_image)
void imProcessNegative (const imImage *src_image, imImage *dst_image)
void imProcessThreshold (const imImage *src_image, imImage *dst_image, int level, int value)
void imProcessThresholdByDiff (const imImage *src_image1, const imImage *src_image2, imImage *dst_image)
void imProcessHysteresisThreshold (const imImage *src_image, imImage *dst_image, int low_thres, int high_thres)
void imProcessHysteresisThresEstimate (const imImage
```

```
*src_image, int *low_thres, int *high_thres)
int imProcessUniformErrThreshold (const imlImage *src_image,
imlImage *dst_image)

void imProcessDifusionErrThreshold (const imlImage *src_image,
imlImage *dst_image, int level)
int imProcessPercentThreshold (const imlImage *src_image,
imlImage *dst_image, float percent)
int imProcessOtsuThreshold (const imlImage *src_image, imlImage
*dst_image)
int imProcessMinMaxThreshold (const imlImage *src_image,
imlImage *dst_image)
void imProcessLocaMaxThresEstimate (const imlImage *src_image,
int *thres)
void imProcessSliceThreshold (const imlImage *src_image, imlImage
*dst_image, int start_level, int end_level)
void imProcessPixelate (const imlImage *src_image, imlImage
*dst_image, int box_size)
void imProcessPosterize (const imlImage *src_image, imlImage
*dst_image, int level)
```

include

im_process_loc.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

```
int imProcessReduce (const imlImage *src_image, imlImage  
*dst_image, int order)  
int imProcessResize (const imlImage *src_image, imlImage  
*dst_image, int order)  
void imProcessReduceBy4 (const imlImage *src_image, imlImage  
*dst_image)  
void imProcessCrop (const imlImage *src_image, imlImage  
*dst_image, int xmin, int ymin)  
void imProcessAddMargins (const imlImage *src_image, imlImage  
*dst_image, int xmin, int ymin)  
void imProcessCalcRotateSize (int width, int height, int *new_width, int  
*new_height, double cos0, double sin0)  
int imProcessRotate (const imlImage *src_image, imlImage  
*dst_image, double cos0, double sin0, int order)  
void imProcessRotate90 (const imlImage *src_image, imlImage  
*dst_image, int dir)  
void imProcessRotate180 (const imlImage *src_image, imlImage  
*dst_image)  
void imProcessMirror (const imlImage *src_image, imlImage  
*dst_image)  
void imProcessFlip (const imlImage *src_image, imlImage *dst_image)  
int imProcessRadial (const imlImage *src_image, imlImage  
*dst_image, float k1, int order)  
int imProcessGrayMorphConvolve (const imlImage *src_image,  
imlImage *dst_image, const imlImage *kernel, int ismax)  
int imProcessGrayMorphErode (const imlImage *src_image,  
imlImage *dst_image, int kernel_size)  
int imProcessGrayMorphDilate (const imlImage *src_image,  
imlImage *dst_image, int kernel_size)
```



```
int imProcessRangeContrastThreshold (const imlImage  
    *src_image, imlImage *dst_image, int kernel_size, int min_range)  
int imProcessLocalMaxThreshold (const imlImage *src_image,  
    imlImage *dst_image, int kernel_size, int min_thres)  
int imProcessConvolve (const imlImage *src_image, imlImage  
    *dst_image, const imlImage *kernel)  
int imProcessConvolveRep (const imlImage *src_image, imlImage  
    *dst_image, const imlImage *kernel, int count)  
int imProcessCompassConvolve (const imlImage *src_image,  
    imlImage *dst_image, imlImage *kernel)  
void imProcessRotateKernel (imlImage *kernel)  
    int imProcessDiffOfGaussianConvolve (const imlImage *src_image,  
        imlImage *dst_image, float stddev1, float stddev2)  
    int imProcessDiffOfGaussianConvolveRep (const imlImage  
        *src_image, imlImage *dst_image, float stddev1, float stddev2)  
    int imProcessLapOfGaussianConvolve (const imlImage *src_image,  
        imlImage *dst_image, float stddev)  
    int imProcessMeanConvolve (const imlImage *src_image, imlImage  
        *dst_image, int kernel_size)  
    int imProcessGaussianConvolveRep (const imlImage *src_image,  
        imlImage *dst_image, float stddev)  
    int imProcessGaussianConvolve (const imlImage *src_image,  
        imlImage *dst_image, float stddev)  
    int imProcessSobelConvolve (const imlImage *src_image, imlImage  
        *dst_image)  
void imProcessZeroCrossing (const imlImage *src_image, imlImage  
    *dst_image)  
void imProcessCanny (const imlImage *src_image, imlImage  
    *dst_image, float stddev)  
int imGaussianStdDev2Repetitions (float stddev)  
int imGaussianStdDev2KernelSize (float stddev)
```

include

im_process_glo.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Functions

```
void imProcessFFT (const imImage *src_image, imImage *dst_image)
void imProcessIFFT (const imImage *src_image, imImage
    *dst_image)
void imProcessFFTRaw (imImage *src_image, int inverse, int center, int
    normalize)
void imProcessSwapQuadrants (imImage *src_image, int
    center2origin)
int imProcessHoughLines (const imImage *src_image, imImage
    *dst_image)
int imProcessHoughLinesDraw (const imImage *src_image, const
    imImage *hough_points, imImage *dst_image)
void imProcessCrossCorrelation (const imImage *src_image1, const
    imImage *src_image2, imImage *dst_image)
void imProcessAutoCorrelation (const imImage *src_image,
    imImage *dst_image)
void imProcessDistanceTransform (const imImage *src_image,
    imImage *dst_image)
void imProcessRegionalMaximum (const imImage *src_image,
    imImage *dst_image)
```

include

im_process_ana.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

Exp

[Go to the source code of this file.](#)

Data Structures

```
struct _imStats
```

Typedefs

```
typedef _imStats imStats
```

Functions

```
float imCalcRMSError (const imImage *image1, const
                      imImage *image2)
float imCalcSNR (const imImage *src_image, const imImage
                  *noise_image)
unsigned long imCalcCountColors (const imImage *src_image)
void imCalcHistogram (const unsigned char *data, int count,
                      unsigned long *histo, int accum)
void imCalcUShortHistogram (const unsigned short *data,
                            int count, unsigned long *histo, int accum)
void imCalcGrayHistogram (const imImage *src_image,
                          unsigned long *histo, int accum)
void imCalcImageStatistics (const imImage *src_image,
                            imStats *stats)
void imCalcHistogramStatistics (const imImage
                                 *src_image, imStats *stats)
void imCalcHistoImageStatistics (const imImage
                                  *src_image, int *median, int *mode)
int imAnalyzeFindRegions (const imImage *src_image,
                           imImage *dst_image, int connect)
void imAnalyzeMeasureArea (const imImage *image, int
                           *area)
void imAnalyzeMeasurePerimArea (const imImage *image,
                                float *perimarea)
void imAnalyzeMeasureCentroid (const imImage *image,
                               const int *area, int region_count, float *cx, float *cy)
void imAnalyzeMeasurePrincipalAxis (const imImage
                                      *image, const int *data_area, const float *cx, const float
                                      *cy, const int region_count, float *major_slope, float
                                      *major_length, float *minor_slope, float *minor_length)
```

```
void imAnalyzeMeasureHoles (const imlImage *image, int
    connect, int *holes_count, int *area, float *perim)
void imAnalyzeMeasurePerimeter (const imlImage *image,
    float *perim)
void imProcessPerimeterLine (const imlImage *src_image,
    imlImage *dst_image)
void imProcessPrune (const imlImage *src_image, imlImage
    *dst_image, int connect, int start_size, int end_size)
void imProcessFillHoles (const imlImage *src_image,
    imlImage *dst_image, int connect)
```

_imStats Struct Reference

[Image Statistics Calculations]

Detailed Description

Numerical Statistics Structure

Data Fields

```
float max
float min
unsigned long positive
unsigned long negative
unsigned long zeros
float mean
float stddev
```

Field Documentation

float _imStats::max

Maximum value

float _imStats::min

Minimum value

unsigned long _imStats::positive

Number of Positive Values

unsigned long _imStats::negative

Number of Negative Values

unsigned long _imStats::zeros

Number of Zeros

float _imStats::mean

Mean

float _imStats::stddev

Standard Deviation

The documentation for this struct was generated from the following file:

- [**im_process_ana.h**](#)

imAttribArray Class Reference

[Utilities]

Detailed Description

Same as [imAttribTable](#), but uses an array of fixed size.

Public Member Functions

```
imAttribArray (int count)
~imAttribArray ()
int Count () const
void RemoveAll ()
void CopyFrom (const imAttribArray &table)
void Set (int index, const char *name, int data_type, int count,
          const void *data)
const void * Get (int index, char *name=0, int *data_type=0, int
                  *count=0) const
void ForEach (void *user_data, imAttribTableCallback
              attrib_func) const
```

Constructor & Destructor Documentation

imAttribArray::imAttribArray(int *count*) [inline]

Creates an empty array.

```
00080      { ptable = imAttribArrayCreate(count); }
```

imAttribArray::~imAttribArray() [inline]

Destroys the array and all the attributes.

```
00084      { imAttribTableDestroy(ptable); ptable = 0; }
```

Member Function Documentation

int imAttribArray::Count() const [inline]

Returns the number of elements in the array.

```
00088 { return imAttribTableCount(ptable); }
```

void imAttribArray::RemoveAll() [inline]

Removes all the attributes in the array

```
00092 { imAttribTableRemoveAll(ptable); }
```

void imAttribArray::CopyFrom(const imAttribArray & table) [inline]

Copies the contents of the given table into this table.

```
00096 { imAttribArrayCopyFrom(ptable, table.ptable); }
```

void imAttribArray::Set(int	<i>index,</i>
	const char * <i>name,</i>
	int <i>data_type,</i>
	int <i>count,</i>
	const void * <i>data</i>
) [inline]

Inserts an attribute into the array.

Data is duplicated if not NULL, else data is initialized with zeros.

```
00101 { imAttribArraySet(ptable, index, name, data_type, count,
```

const void* imAttribArray::Get (int	<i>index,</i>
	char *	<i>name = 0,</i>
	int *	<i>data_type = 0,</i>
	int *	<i>count = 0</i>
) const [inline]		

Finds an attribute in the array. Returns the attribute if found, NULL otherwise.

```
00106      { return imAttribArrayGet(ptable, index, name, data_type,
```

void imAttribArray::ForEach (void *	<i>user_data,</i>
	imAttribTableCallback	<i>attrib_func</i>
) const [inline]		

For each attribute calls the user callback. If the callback returns 0 the function returns.

```
00110      { imAttribTableForEach(ptable, user_data, attrib_func); }
```

The documentation for this class was generated from the following file:

- [im_attrib.h](#)

include

im_attrib.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Attributes Table.
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_ATTRIB_H_
00009 #define __IM_ATTRIB_H_
00010
00011 #include "im_attrib_flat.h"
00012
00013 /** \brief Attributes Table.
00014 *
00015 * \par
00016 * All the attributes have a name, a type, a count and the dat
00017 * Names are usually strings with less than 30 chars.
00018 * \par
00019 * Attributes are stored in a hash table for fast access. \n
00020 * We use the hash function described in "The Practice of Progr
00021 * \ingroup util */
00022 class imAttribTable
00023 {
00024     imAttribTablePrivate* ptable;
00025 public:
00026
00027     /** Creates an empty table.
00028     * If size is zero the default size of 101 is used. Size mus
00029     * Other common values are 67, 599 and 1499.*/
00030     imAttribTable(int hash_size)
00031     { ptable = imAttribTableCreate(hash_size); }
00032
00033     /** Destroys the table and all the attributes. */
00034     ~imAttribTable()
00035     { imAttribTableDestroy(ptable); ptable = 0; }
00036
00037     /** Returns the number of elements in the table. */
00038     int Count() const
00039     { return imAttribTableCount(ptable); }
```

```
00040
00041     /** Removes all the attributes in the table */
00042     void RemoveAll()
00043         { imAttribTableRemoveAll(ptable); }
00044
00045     /** Copies the contents of the given table into this table.
00046     void CopyFrom(const imAttribTable& table)
00047         { imAttribTableCopyFrom(ptable, table.ptable); }
00048
00049     /** Inserts an attribute into the table. \n
00050     * Data is duplicated if not NULL, else data is initialized
00051     void Set(const char* name, int data_type, int count, const void* data)
00052         { imAttribTableSet(ptable, name, data_type, count, data); }
00053
00054     /** Removes an attribute from the table given its name. */
00055     void UnSet(const char *name)
00056         { imAttribTableUnSet(ptable, name); }
00057
00058     /** Finds an attribute in the table.
00059     * Returns the attribute if found, NULL otherwise. */
00060     const void* Get(const char *name, int *data_type = 0, int *count = 0)
00061         { return imAttribTableGet(ptable, name, data_type, count); }
00062
00063     /** For each attribute calls the user callback. If the callback
00064     void ForEach(void* user_data, imAttribTableCallback attrib_func)
00065         { imAttribTableForEach(ptable, user_data, attrib_func); }
00066     };
00067
00068     /** \brief Attributes Table.
00069     *
00070     * \par
00071     * Same as \ref imAttribTable, but uses an array of fixed size
00072     * \ingroup util */
00073     class imAttribArray
00074     {
00075         imAttribTablePrivate* ptable;
00076     public:
00077
00078         /** Creates an empty array. */
00079         imAttribArray(int count)
00080             { ptable = imAttribArrayCreate(count); }
00081
00082         /** Destroys the array and all the attributes. */
00083         ~imAttribArray()
00084             { imAttribTableDestroy(ptable); ptable = 0; }
00085
00086         /** Returns the number of elements in the array. */

```

```
00087 int Count() const
00088     { return imAttribTableCount(ptable); }
00089
00090     /** Removes all the attributes in the array */
00091 void RemoveAll()
00092     { imAttribTableRemoveAll(ptable); }
00093
00094     /** Copies the contents of the given table into this table.
00095 void CopyFrom(const imAttribArray& table)
00096     { imAttribArrayCopyFrom(ptable, table.ptable); }
00097
00098     /** Inserts an attribute into the array. \n
00099      * Data is duplicated if not NULL, else data is initialized
00100 void Set(int index, const char* name, int data_type, int cou
00101     { imAttribArraySet(ptable, index, name, data_type, count,
00102
00103     /** Finds an attribute in the array.
00104      * Returns the attribute if found, NULL otherwise. */
00105 const void* Get(int index, char *name = 0, int *data_type =
00106     { return imAttribArrayGet(ptable, index, name, data_type,
00107
00108     /** For each attribute calls the user callback. If the callb
00109 void ForEach(void* user_data, imAttribTableCallback attrib_f
00110     { imAttribTableForEach(ptable, user_data, attrib_func); }
00111 };
00112
00113 #endif
```

include

im_binfile.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Binary File Access.
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #include "im_util.h"
00009
00010 #ifndef __IM_BINFILE_H
00011 #define __IM_BINFILE_H
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018 /** \defgroup binfile Binary File Access
00019 *
00020 * \par
00021 * These functions are very usefull for reading/writing binary
00022 * that have headers or data that have to be converted dependi-
00023 * the current CPU byte order. It can invert 2, 4 or 8 bytes n-
00024 * \par
00025 * It will process the data only if the file format is differen-
00026 * \par
00027 * Can read from disk or memory. In case of a memory buffer, t-
00028 * \par
00029 * See \ref im_binfile.h
00030 * \ingroup util */
00031
00032 typedef struct _imBinFile imBinFile;
00033
00034 /** Opens an existant binary file for reading.
00035 * The default file byte order is the CPU byte order.
00036 * Returns NULL if failed.
00037 * \ingroup binfile */
00038 imBinFile* imBinFileOpen(const char* pFileName);
00039
```

```
00040 /** Creates a new binary file for writing.  
00041 * The default file byte order is the CPU byte order.  
00042 * Returns NULL if failed.  
00043 * \ingroup binfile */  
00044 imBinFile* imBinFileNew(const char* pFileName);  
00045  
00046 /** Closes the file.  
00047 * \ingroup binfile */  
00048 void imBinFileClose(imBinFile* bfile);  
00049  
00050 /** Indicates that was an error on the last operation.  
00051 * \ingroup binfile */  
00052 int imBinFileError(imBinFile* bfile);  
00053  
00054 /** Returns the file size in bytes.  
00055 * \ingroup binfile */  
00056 unsigned long imBinFileSize(imBinFile* bfile);  
00057  
00058 /** Changes the file byte order. Returns the old one.  
00059 * \ingroup binfile */  
00060 int imBinFileByteOrder(imBinFile* bfile, int pByteOrder);  
00061  
00062 /** Reads an array of count values with byte sizes: 1, 2, 4, or 8.  
00063 * \ingroup binfile */  
00064 unsigned long imBinFileRead(imBinFile* bfile, void* pValues, u  
00065  
00066 /** Writes an array of values with sizes: 1, 2, 4, or 8. And it is  
00067 * <b>ATTENTION</b>: The function will not make a temporary copy of  
00068 * So after the call the values will be invalid, if the file buffer  
00069 * \ingroup binfile */  
00070 unsigned long imBinFileWrite(imBinFile* bfile, void* pValues,  
00071  
00072 /** Writes a string without the NULL terminator. The function  
00073 * The internal buffer is fixed at 4096 bytes.  
00074 * \ingroup binfile */  
00075 unsigned long imBinFilePrintf(imBinFile* bfile, char *format,  
00076  
00077 /** Moves the file pointer from the begining of the file.\n  
00078 * When writing to a file seeking can go beyond the end of the file.  
00079 * \ingroup binfile */  
00080 void imBinFileSeekTo(imBinFile* bfile, unsigned long pOffset);  
00081  
00082 /** Moves the file pointer from current position.\n  
00083 * If the offset is a negative value the pointer moves backward.  
00084 * \ingroup binfile */  
00085 void imBinFileSeekOffset(imBinFile* bfile, long pOffset);  
00086
```

```
00087 /** Moves the file pointer from the end of the file.\n00088 * The offset is usually a negative value.\n00089 * \ingroup binfile */\n00090 void imBinFileSeekFrom(imBinFile* bfile, long poffset);\n00091\n00092 /** Returns the current offset position.\n00093 * \ingroup binfile */\n00094 unsigned long imBinFileTell(imBinFile* bfile);\n00095\n00096 /** Indicates that the file pointer is at the end of the file.\n00097 * \ingroup binfile */\n00098 int imBinFileEndOfFile(imBinFile* bfile);\n00099\n00100 /** Predefined I/O Modules.\n00101 * \ingroup binfile */\n00102 enum imBinFileModule\n00103 {\n00104     IM_RAWFILE,    /**< System dependent file I/O Rotines. */\n00105     IM_STREAM,    /**< Standard Ansi C Stream I/O Rotines. */\n00106     IM_MEMFILE,   /**< Uses a memory buffer. */\n00107     IM_SUBFILE,   /**< It is a sub file. FileName is a imBinFile\n00108     IM_IOCUSTOM0 /**< Other registered modules starts from here\n00109 };\n00110\n00111 /** Sets the current I/O module.\n00112 * \returns the previous function set, or -1 if failed.\n00113 * \ingroup binfile */\n00114 int imBinFileSetCurrentModule(int pModule);\n00115\n00116 /** \brie Memory File I/O Filename\n00117 *\n00118 * \par\n00119 * Fake file name for the memory I/O module.\n00120 * \ingroup binfile */\n00121 typedef struct _imBinMemoryFileName\n00122 {\n00123     unsigned char *buffer; /**< The memory buffer. If you are re\n00124                           *      If you are writing the buffer ca\n00125     int size;           /**< Size of the buffer. */\n00126     float reallocate;   /**< Reallocate factor for the memory\n00127 }imBinMemoryFileName;\n00128\n00129\n00130 #if defined(__cplusplus)\n00131 }\n00132 #endif\n00133
```

```
00134
00135 #if defined(__cplusplus)
00136
00137 /** Base class to help the creation of new modules.\n
00138 * It handles the read/write operations with byte order corre-\n00139 * \ingroup binfile */
00140 class imBinFileBase
00141 {
00142     friend class imBinSubFile;
00143
00144 protected:
00145     int IsNew,
00146         FileByteOrder,
00147         DoByteOrder; // to speed up byte order checking
00148
00149 // These will actually read/write the data
00150 virtual unsigned long ReadBuf(void* pValues, unsigned long p
00151 virtual unsigned long WriteBuf(void* pValues, unsigned long
00152
00153 public:
00154
00155     int InitByteOrder(int ByteOrder)
00156     {
00157         int old_byte_order = this->FileByteOrder;
00158         this->FileByteOrder = ByteOrder;
00159
00160         if (ByteOrder != imBinCPUByteOrder())
00161             this->DoByteOrder = 1;
00162         else
00163             this->DoByteOrder = 0;
00164         return old_byte_order;
00165     }
00166
00167 // These will take care of byte swap if needed.
00168
00169     unsigned long Read(void* pValues, unsigned long pCount, int
00170     {
00171         unsigned long rSize = ReadBuf(pValues, pCount * pSizeOf);
00172         if (pSizeOf != 1 && DoByteOrder) imBinSwapBytes(pValues, p
00173         return rSize/pSizeOf;
00174     }
00175
00176     unsigned long Write(void* pValues, unsigned long pCount, int
00177     {
00178         if (pSizeOf != 1 && DoByteOrder) imBinSwapBytes(pValues, p
00179         return WriteBuf(pValues, pCount * pSizeOf)/pSizeOf;
00180     }
```

```
00181
00182     virtual void Open(const char* pFileName) = 0;
00183     virtual void New(const char* pFileName) = 0;
00184     virtual void Close() = 0;
00185     virtual unsigned long FileSize() = 0;
00186     virtual int HasError() const = 0;
00187     virtual void SeekTo(unsigned long pOffset) = 0;
00188     virtual void SeekOffset(long pOffset) = 0;
00189     virtual void SeekFrom(long pOffset) = 0;
00190     virtual unsigned long Tell() const = 0;
00191     virtual int EndOfFile() const = 0;
00192 };
00193
00194 /** File I/O module creation callback.
00195  * \ingroup binfile */
00196 typedef imBinFileBase* (*imBinFileNewFunc)();
00197
00198 /** Register a user I/O module.\n
00199  * Returns the new function set id.\n
00200  * Accepts up to 10 modules.
00201  * \ingroup binfile */
00202 int imBinFileRegisterModule(imBinFileNewFunc pNewFunc);
00203
00204 #endif
00205
00206 #endif
```

include

im_dib.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Windows DIB (Device Independent Bitmap)
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_DIB_H
00009 #define __IM_DIB_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup dib Windows DIB
00017 *
00018 * \par
00019 * Windows DIBs in memory are handled just like a BMP file with
00020 * These functions will work only in Windows. They are useful
00021 * with the clipboard, with capture drivers, with the AVI and
00022 * \par
00023 * Supported DIB aspects:
00024 * \li bpp must be 1, 4, 8, 16, 24, or 32.
00025 * \li BITMAPV4HEADER or BITMAPV5HEADER are handled but ignored.
00026 * \li BITMAPCOREHEADER is not handled.
00027 * \li BI_JPEG and BI_PNG compressions are not handled.
00028 * \li biHeight can be negative, compression can be RLE only if
00029 *     from imDibCreateReference, imDibPasteClipboard, imDibLoad
00030 * \li can not encode/decode Images to/from RLE compressed Dib
00031 * \li if working with RLE Dibs bits_size is greater than use
00032 * \li the resolution of a new Dib is taken from the screen.
00033 * \li SetDIBitsToDevice(start_scan is 0, scan_lines is dib->b
00034 * \li StretchDIBits(use always DIB_RGB_COLORS).
00035 * \li CreateDIBPatternBrushPt(packed_dib is dib->dib).
00036 * \par
00037 * Must include <windows.h> before using these functions. \n
00038 * Check <wingdi.h> for structures and definitions.
00039 * \par
```

```
00040 * See \ref im_dib.h
00041 * \ingroup util */
00042
00043
00044 /** \brief Windows DIB Structure
00045 *
00046 * \par
00047 * Handles a DIB in memory. \n
00048 * The DIB is stored in only one buffer.
00049 * The secondary members are pointers to the main buffer.
00050 * \ingroup dib */
00051 typedef struct _imDib
00052 {
00053     HGLOBAL handle;           /**< The windows memory han
00054     BYTE* dib;             /**< The DIB as it is defin
00055     int size;              /**< Full size in memory */
00056
00057     BITMAPINFO* bmi;        /**< Bitmap Info = Bitmap I
00058     BITMAPINFOHEADER* bmih; /**< Bitmap Info Header */
00059     RGBQUAD* bmic;          /**< Bitmap Info Colors = P
00060     BYTE* bits;            /**< Bitmap Bits */
00061
00062     int palette_count; /**< number of colors in the pa
00063     int bits_size;          /**< size in bytes of the B
00064     int line_size;          /**< size in bytes of one line
00065     int pad_size;           /**< number of bytes remaining
00066
00067     int is_reference; /**< only a reference, do not copy
00068 } imDib;
00069
00070 /** Creates a new DIB. \n
00071 * use bpp=-16/-32 to allocate space for BITFLIEDS.
00072 * \ingroup dib */
00073 imDib* imDibCreate(int width, int height, int bpp);
00074
00075 /**Duplicates the DIB contents in a new DIB.
00076 * \ingroup dib */
00077 imDib* imDibCreateCopy(const imDib* dib);
00078
00079 /**Creates a DIB using an already allocated memory. \n
00080 * "bmi" must be a pointer to BITMAPINFOHEADER. \n
00081 * "bits" can be NULL if it is inside "bmi" after the palette.
00082 * \ingroup dib */
00083 imDib* imDibCreateReference(BYTE* bmi, BYTE* bits);
00084
00085 /**Creates a DIB section for drawing porposes. \n
00086 * Returns the image handle also created.
```

```
00087  * \ingroup dib */
00088  imDib* imDibCreateSection(HDC hDC, HBITMAP *image, int width,
00089
00090  /** Destroy the DIB
00091  * \ingroup dib */
00092  void imDibDestroy(imDib* dib);
00093
00094  /** DIB GetPixel function definition. \n
00095  * the ulong is a raw copy of the bits, use (unsigned char*)&
00096  * \ingroup dib */
00097  typedef unsigned long (*imDibLineGetPixel)(unsigned char* line
00098
00099  /** Returns a function to read pixels from a DIB line.
00100  * \ingroup dib */
00101  imDibLineGetPixel imDibLineGetPixelFunc(int bpp);
00102
00103  /** DIB SetPixel function definition
00104  * \ingroup dib */
00105  typedef void (*imDibLineSetPixel)(unsigned char* line, int col
00106
00107  /** Returns a function to write pixels into a DIB line.
00108  * \ingroup dib */
00109  imDibLineSetPixel imDibLineSetPixelFunc(int bpp);
00110
00111  /** Creates a DIB from a image handle and a palette handle.
00112  * \ingroup dib */
00113  imDib* imDibFromHBitmap(const HBITMAP image, const HPALETTE hP
00114
00115  /** Creates a image handle from a DIB.
00116  * \ingroup dib */
00117  HBITMAP imDibToHBitmap(const imDib* dib);
00118
00119  /** Returns a Logical palette from the DIB palette. \n
00120  * DIB bpp must be <=8.
00121  * \ingroup dib */
00122  HPALETTE imDibLogicalPalette(const imDib* dib);
00123
00124  /** Captures the screen into a DIB.
00125  * \ingroup dib */
00126  imDib* imDibCaptureScreen(int x, int y, int width, int height)
00127
00128  /** Transfer the DIB to the clipboard. \n
00129  * "dib" pointer can not be used after, or use imDibCopyClipbo
00130  * Warning: Clipboard functions in C++ can fail with Visual C+
00131  * \ingroup dib */
00132  void imDibCopyClipboard(imDib* dib);
00133
```

```
00134 /** Creates a reference for the DIB in the clipboard if any. R
00135  * Warning: Clipboard functions in C++ can fail with Visual C+
00136  * \ingroup dib */
00137 imDib* imDibPasteClipboard(void);
00138
00139 /** Checks if there is a dib at the clipboard.
00140  * \ingroup dib */
00141 int imDibIsClipboardAvailable(void);
00142
00143 /** Saves the DIB into a file ".bmp".
00144  * \ingroup dib */
00145 int imDibSaveFile(const imDib* dib, const char* filename);
00146
00147 /** Creates a DIB from a file ".bmp".
00148  * \ingroup dib */
00149 imDib* imDibLoadFile(const char* filename);
00150
00151 /** Converts a DIB into an RGBA image. alpha is optional. bpp
00152  * alpha is used only when bpp=32.
00153  * \ingroup dib */
00154 void imDibDecodeToRGBA(const imDib* dib, unsigned char* red, u
00155
00156 /** Converts a DIB into an indexed image. bpp must be <=8. col
00157  * colors is rgb packed (RGBRGBRGB...)
00158  * \ingroup dib */
00159 void imDibDecodeToMap(const imDib* dib, unsigned char* map, lo
00160
00161 /** Converts an RGBA image into a DIB. alpha is optional. bpp
00162  * alpha is used only when bpp=32.
00163  * \ingroup dib */
00164 void imDibEncodeFromRGBA(imDib* dib, const unsigned char* red,
00165
00166 /** Converts an indexed image into a DIB. bpp must be <=8. \n
00167  * colors is rgb packed (RGBRGBRGB...)
00168  * \ingroup dib */
00169 void imDibEncodeFromMap(imDib* dib, const unsigned char* map,
00170
00171 /** Converts a IM_RGB packed image, with or without alpha, int
00172  * \ingroup dib */
00173 void imDibEncodeFromBitmap(imDib* dib, const unsigned char* da
00174
00175 /** Converts a DIB into IM_RGB packed image, with or without a
00176  * \ingroup dib */
00177 void imDibDecodeToBitmap(const imDib* dib, unsigned char* data
00178
00179 #ifdef __IM_IMAGE_H
00180 /* You must include "im_image.h" before this header to enable
```

```
00181
00182 /** Creates a imImage from the dib data.
00183 * \ingroup dib */
00184 imImage* imDibToImage(const imDib* dib);
00185
00186 /** Creates a Dib from the image. It must be a bitmap image.
00187 * \ingroup dib */
00188 imDib* imDibFromImage(const imImage* image);
00189
00190 #endif
00191
00192 #if defined(__cplusplus)
00193 }
00194 #endif
00195
00196 #endif
```

include

im_file.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief File Access
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_FILE_H
00009 #define __IM_FILE_H
00010
00011 #include "im.h"
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018 /** \defgroup filesdk File Format SDK
00019 * \par
00020 * All the file formats are based on theses structures. Use the
00021 * The LineBuffer functions will help transfer image from form
00022 * \par
00023 * See \ref im_file.h
00024 * \ingroup file */
00025
00026
00027 /** \brief Image File Format Base (SDK Use Only)
00028 *
00029 * \par
00030 * Base container to hold format independent state variables.
00031 * \ingroup filesdk */
00032 struct _imFile
00033 {
00034     int is_new;
00035     void* attrib_table;    /**< in fact is a imAttribTable, but we
00036
00037     void* line_buffer;    /**< used for line conversion, contains
00038     int line_buffer_size;
00039     int line_buffer_extra; /*< extra bytes to be allocated */
```

```
00040     int line_buffer_alloc; /**< total allocated so far */
00041     int counter;
00042
00043     int convert_bpp;          /**< number of bpp to expand or com
00044     int switch_type;         /**< flag to switch the original data
00045
00046     long palette[256];
00047     int palette_count;
00048
00049     int user_color_mode,
00050         user_data_type,
00051         file_color_mode, /* these two must be filled by te driv
00052         file_data_type;
00053
00054     /* these must be filled by the driver when reading,
00055      and given by the user when writing. */
00056
00057     char compression[10];
00058     int image_count;
00059     int width,
00060         height;
00061 };
00062
00063
00064 /* Internal Use only */
00065
00066 /* Initializes the imFile structure.
00067  * Used by the special format RAW. */
00068 void imFileClear(imFile* ifile);
00069
00070 /* Initializes the line buffer.
00071  * Used by "im_file.cpp" only. */
00072 void imFileLineBufferInit(imFile* ifile);
00073
00074 /* Check if the conversion is valid.
00075  * Used by "im_file.cpp" only. */
00076 int imFileCheckConversion(imFile* ifile);
00077
00078
00079
00080 /* File Format SDK */
00081
00082 /** Number of lines to be accessed.
00083  * \ingroup filesdk */
00084 int imFileLineBufferCount(imFile* ifile);
00085
00086 /** Increments the row and plane counters.
```

```
00087 * \ingroup filesdk */
00088 void imFileLineBufferInc(imFile* ifile, int *row, int *plane);
00089
00090 /** Converts from FILE color mode to USER color mode.
00091 * \ingroup filesdk */
00092 void imFileLineBufferRead(imFile* ifile, void* data, int line,
00093
00094 /** Converts from USER color mode to FILE color mode.
00095 * \ingroup filesdk */
00096 void imFileLineBufferWrite(imFile* ifile, const void* data, in
00097
00098 /** Utility to calculate the line size in byte with a specificie
00099 * "align" can be 1, 2 or 4.
00100 * \ingroup filesdk */
00101 int imFileLineSizeAligned(int width, int bpp, int align);
00102
00103
00104 #if defined(__cplusplus)
00105 }
00106 #endif
00107
00108 #endif
```

include

im_format.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief File Format Access
00003 *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #include "im_file.h"
00009 #include "im_attrib.h"
00010
00011 #ifndef __IM_FORMAT_H
00012 #define __IM_FORMAT_H
00013
00014
00015 /** \brief Image File Format Driver (SDK Use Only)
00016 *
00017 * \par
00018 * Virtual Base class for file formats. All file formats inher
00019 * \ingroup filesdk */
00020 class imFormat: public _imFile
00021 {
00022 public:
00023     const char* format;
00024     const char* desc;
00025     const char* ext;
00026     const char** comp;
00027     int comp_count,
00028         can_sequence;
00029
00030     imFormat(const char* _format, const char* _desc, const char*
00031             const char** _comp, int _comp_count, int _can_seque
00032             :format(_format), desc(_desc), ext(_ext), comp(_comp),
00033             comp_count(_comp_count), can_sequence(_can_sequence)
00034     {}
00035
00036     imAttribTable* AttribTable() {return (imAttribTable*)this->a
00037
00038     /* Pure Virtual Methods. Every driver must implement all the
00039
```

```
00040     virtual int Open(const char* file_name) = 0; // Must initial
00041     virtual int New(const char* file_name) = 0;
00042     virtual void Close() = 0;
00043     virtual void* Handle() = 0;
00044     virtual int ReadImageInfo(int index) = 0;      // Should updat
00045     virtual int ReadImageData(void* data) = 0;
00046     virtual int WriteImageInfo() = 0;                // Should updat
00047     virtual int WriteImageData(void* data) = 0; // Must update
00048     virtual int CanWrite(const char* compression, int color_mode
00049 };
00050
00051 extern "C"
00052 {
00053
00054 /* Internal Use only */
00055
00056 /* Opens a file with the respective format driver
00057 * Uses the file extension to speed up the search for the form
00058 * Used by "im_file.cpp" only. */
00059 imFormat* imFormatOpen(const char* file_name, int *error);
00060
00061 /* Creates a file using the given format driver.
00062 * Used by "im_file.cpp" only. */
00063 imFormat* imFormatNew(const char* file_name, const char* forma
00064
00065 /* Registers all the internal formats.
00066 * Used by "im_format.cpp" only. */
00067 void imFormatRegisterAll(void);
00068
00069
00070 /* File Format SDK */
00071
00072 /** Format function initialization definition.
00073 * \ingroup filesdk */
00074 typedef imFormat* (*imFormatFunc)();
00075
00076 /** Register a format driver.
00077 * \ingroup filesdk */
00078 void imFormatRegister(imFormatFunc format_init);
00079
00080
00081 }
00082
00083 #endif
```

include

im_complex.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Complex Data Type.
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_COMPLEX_H
00009 #define __IM_COMPLEX_H
00010
00011 #include "im_math.h"
00012
00013 /** \defgroup cpx Complex Numbers
00014 * \par
00015 * See \ref im_complex.h
00016 * \ingroup util
00017 */
00018
00019 /** \brief Complex Float Data Type
00020 *
00021 * \par
00022 * Complex class using two floats, one for real part, one for
00023 * \par
00024 * It is not a complete complex class, we just implement const
00025 * All the other operators and functions are external to the c
00026 * \ingroup cpx */
00027 class imcfloat
00028 {
00029 public:
00030     float real; ///< Real part.
00031     float imag; ///< Imaginary part.
00032
00033     /// Default Constructor (0,0).
00034     imcfloat():real(0), imag(0) {}
00035
00036     /// Constructor from (real, imag)
00037     imcfloat(const float& r, const float& i):real(r),imag(i) {}
00038
00039     /// Constructor from (real)
```

```
00040     imcfloat(const float& r):real(r),imag(0) {}
00041 }
00042
00043 /** \addtogroup cpx
00044 * Complex numbers operators.
00045 * @{
00046 */
00047
00048 inline int operator <= (const imcfloat& C1, const imcfloat& C2)
00049 {
00050     return ((C1.real <= C2.real) && (C1.imag <= C2.imag));
00051 }
00052
00053 inline int operator <= (const imcfloat& C, const float& F)
00054 {
00055     return ((F <= C.real) && (0 <= C.imag));
00056 }
00057
00058 inline imcfloat operator + (const imcfloat& C1, const imcfloat
00059 {
00060     return imcfloat(C1.real + C2.real, C1.imag + C2.imag);
00061 }
00062
00063 inline imcfloat operator += (const imcfloat& C1, const imcfloat
00064 {
00065     return imcfloat(C1.real + C2.real, C1.imag + C2.imag);
00066 }
00067
00068 inline imcfloat operator - (const imcfloat& C1, const imcfloat
00069 {
00070     return imcfloat(C1.real - C2.real, C1.imag - C2.imag);
00071 }
00072
00073 inline imcfloat operator * (const imcfloat& C1, const imcfloat
00074 {
00075     return imcfloat(C1.real * C2.real - C1.imag * C2.imag,
00076                         C1.imag * C2.real + C1.real * C2.imag)
00077 }
00078
00079 inline imcfloat operator / (const imcfloat& C1, const imcfloat
00080 {
00081     float den = C2.real * C2.real - C2.imag * C2.imag;
00082     return imcfloat((C1.real * C2.real + C1.imag * C2.imag) / de
00083                         (C1.imag * C2.real - C1.real * C2.imag
00084 )
00085
00086 inline imcfloat operator / (const imcfloat& C, const float& R)
```

```
00087 {
00088     return imcfloat(C.real / R, C.imag / R);
00089 }
00090
00091 inline imcfloat operator /= (const imcfloat& C, const float& R
00092 {
00093     return imcfloat(C.real / R, C.imag / R);
00094 }
00095
00096 inline imcfloat operator * (const imcfloat& C, const float& R)
00097 {
00098     return imcfloat(C.real * R, C.imag * R);
00099 }
00100
00101 inline int operator == (const imcfloat& C1, const imcfloat& C2
00102 {
00103     return ((C1.real == C2.real) && (C1.imag == C2.imag));
00104 }
00105
00106 inline float cpxreal(const imcfloat& C)
00107 {
00108     return C.real;
00109 }
00110
00111 inline float cpximag(const imcfloat& C)
00112 {
00113     return C.imag;
00114 }
00115
00116 inline float cpxmag(const imcfloat& C)
00117 {
00118     return sqrtf(C.real*C.real + C.imag*C.imag);
00119 }
00120
00121 inline float cpxphase(const imcfloat& C)
00122 {
00123     return atan2f(C.real, C.imag);
00124 }
00125
00126 inline imcfloat cpxconj(const imcfloat& C)
00127 {
00128     return imcfloat(C.real, -C.imag);
00129 }
00130
00131 inline imcfloat log(const imcfloat& C)
00132 {
00133     return imcfloat(logf(cpxmag(C)), atan2f(C.real, C.imag));
```

```
00134 }
00135
00136 inline imcfloat exp(const imcfloat& C)
00137 {
00138     float mag = expf(C.real);
00139     return imcfloat(mag * cosf(C.imag), mag * sinf(C.imag));
00140 }
00141
00142 inline imcfloat pow(const imcfloat& C1, const imcfloat& C2)
00143 {
00144     return exp(C1 * log(C2));
00145 }
00146
00147 inline imcfloat sqrt(const imcfloat& C)
00148 {
00149     float mag = sqrtf(sqrtf(C.real*C.real + C.imag*C.imag));
00150     float phase = atan2f(C.real, C.imag) / 2;
00151     return imcfloat(mag * cosf(phase), mag * sinf(phase));
00152 }
00153
00154 inline imcfloat cpxpolar(const float& mag, const float& phase)
00155 {
00156     return imcfloat(mag * cosf(phase), mag * sinf(phase));
00157 }
00158
00159 /** @} */
00160
00161 #endif
```

include

im_image.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Image Manipulation
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_IMAGE_H
00009 #define __IM_IMAGE_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup imgclass Image Structure
00017 *
00018 * \par
00019 * Base definitions and functions for image representation. \
00020 * Only the image processing operations depends on these defin
00021 * Image Storage and Image Capture are completely independent.
00022 * \par
00023 * You can also initialize a structure with your own memory bu
00024 * To release the structure without releasing the buffer,
00025 * set "data[0]" to 0 before calling imImageDestroy.
00026 * \par
00027 * See \ref im_image.h
00028 * \ingroup imagerep */
00029
00030
00031 /** \brief Image Structure Definition.
00032 *
00033 * \par
00034 * An image representation than supports all the color spaces,
00035 * planes are always unpacked and the orientation is always bo
00036 * \ingroup imgclass */
00037 typedef struct _imImage
00038 {
00039     /* main parameters */
```

```

00040     int width;           /**< Number of columns */
00041     int height;          /**< Number of lines. */
00042     int color_space;      /**< Color space descriptor. */
00043     int data_type;        /**< Data type descriptor. */
00044
00045     /* secondary parameters */
00046     int depth;            /**< Number of planes
00047     int line_size;         /**< Number of bytes per line in one plane
00048     int plane_size;       /**< Number of bytes per plane.
00049     int size;              /**< Number of bytes occupied by the image
00050     int count;             /**< Number of pixels
00051
00052     /* image data */
00053     void** data;           /**< Image data organized as a 2D matrix
00054                               But plane 0 is also a pointer to the
00055
00056     /* image attributes */
00057     long *palette;          /**< Used when depth=1. Otherwise is NULL
00058     int palette_count;     /**< The palette is always 256 colors at most
00059
00060     void* attrib_table;    /**< in fact is a imAttribTable, but we
00061 } imImage;
00062
00063
00064 /** Creates a new image.
00065 * \ingroup imgclass */
00066 imImage* imImageCreate(int width, int height, int color_space,
00067
00068 /** Initializes the image structure but does not allocates memory
00069 * \ingroup imgclass */
00070 imImage* imImageInit(int width, int height, int color_space, int
00071
00072 /** Destroys the image and frees the memory used.
00073 * image data is destroyed only if it data[0] is not NULL.
00074 * \ingroup imgclass */
00075 void imImageDestroy(imImage* image);
00076
00077 /** Changes the buffer size. Reallocate internal buffers if the
00078 * \ingroup imgclass */
00079 void imImageReshape(imImage* image, int width, int height);
00080
00081 /** Copy image data and attributes from one image to another.
00082 * Images must have the same size and type.
00083 * \ingroup imgclass */
00084 void imImageCopy(const imImage* src_image, imImage* dst_image)
00085
00086 /** Copy image data only one image to another. \n

```

```
00087 * Images must have the same size and type.
00088 * \ingroup imgclass */
00089 void imImageCopyData(const imImage* src_image, imImage* dst_im
00090
00091 /** Creates a copy of the image.
00092 * \ingroup imgclass */
00093 imImage* imImageDuplicate(const imImage* image);
00094
00095 /** Creates a clone of the image. i.e. same attributes but ign
00096 * \ingroup imgclass */
00097 imImage* imImageClone(const imImage* image);
00098
00099 /** Changes an extended attribute. \n
00100 * The data will be internally duplicated. \n
00101 * If data is NULL the attribute is removed. \n
00102 * If count is -1 and data_type is IM_BYTE then data is zero t
00103 * \ingroup imgclass */
00104 void imImageSetAttribute(imImage* image, const char* attrib, i
00105
00106 /** Returns an extended attribute. \n
00107 * Returns NULL if not found.
00108 * \ingroup imgclass */
00109 const void* imImageGetAttribute(const imImage* image, const ch
00110
00111 /** Returns a list of the attribute names. \n
00112 * "attrib" must contain room enough for "attrib_count" names.
00113 * \ingroup imgclass */
00114 void imImageGetAttributeList(const imImage* image, char** attr
00115
00116 /** Sets all image data to zero.
00117 * \ingroup imgclass */
00118 void imImageClear(imImage* image);
00119
00120 /** Indicates that the image can be viewed in common graphic d
00121 * Data type must be IM_BYTE. Color mode can be IM_RGB, IM_MAP
00122 * \ingroup imgclass */
00123 int imImageIsBitmap(const imImage* image);
00124
00125 /** Changes the image palette.
00126 * This will destroy the existing palette and replace it with
00127 * \ingroup imgclass */
00128 void imImageSetPalette(imImage* image, long* palette, int pale
00129
00130 /** Copies the image attributes from src to dst.
00131 * \ingroup imgclass */
00132 void imImageCopyAttributes(const imImage* src_image, imImage*
00133
```

```
00134 /** Returns 1 if the images match width and height. Returns 0
00135 * \ingroup imgclass */
00136 int imImageMatchSize(const imImage* image1, const imImage* ima
00137
00138 /** Returns 1 if the images match color mode and data type. Re
00139 * \ingroup imgclass */
00140 int imImageMatchColor(const imImage* image1, const imImage* im
00141
00142 /** Returns 1 if the images match width, height and data type.
00143 * \ingroup imgclass */
00144 int imImageMatchDataType(const imImage* image1, const imImage*
00145
00146 /** Returns 1 if the images match width, height and color spac
00147 * \ingroup imgclass */
00148 int imImageMatchColorSpace(const imImage* image1, const imImag
00149
00150 /** Returns 1 if the images match in width, height, data type
00151 * \ingroup imgclass */
00152 int imImageMatch(const imImage* image1, const imImage* image2)
00153
00154 /** Loads an image from file. Returns NULL if failed.
00155 * This will call imFileReadImageInfo and imFileReadImageData.
00156 * \ingroup imgclass */
00157 imImage* imFileLoadImage(imFile* ifile, int index, int *error)
00158
00159 /** Loads an image from file, but forces the image to be a bit
00160 * Returns NULL if failed.
00161 * \ingroup imgclass */
00162 imImage* imFileLoadBitmap(imFile* ifile, int index, int *error)
00163
00164 /** Saves the image to file. Returns error code. \n
00165 * This will call imFileWriteImageInfo and imFileWriteImageDat
00166 * \ingroup imgclass */
00167 int imFileSaveImage(imFile* ifile, const imImage* image);
00168
00169 /** Loads an image from file. \n
00170 * Returns NULL if failed.
00171 * \ingroup imgclass */
00172 imImage* imImageLoad(const char* file_name, int index, int *er
00173
00174 /** Loads an image from file, but forces the image to be a bit
00175 * Returns NULL if failed.
00176 * \ingroup imgclass */
00177 imImage* imImageLoadBitmap(const char* file_name, int index, i
00178
00179 /** Changes the image space from gray to binary by just changi
00180 * \ingroup imgclass */
```

```
00181 void imImageSetBinary(imImage* image);
00182
00183 /** Changes a gray data into a binary data.
00184 * \ingroup imgclass */
00185 void imImageMakeBinary(imImage *image);
00186
00187 /** Utility macro to draw the image in a CD library canvas.
00188 * Works only for data_type IM_BYTE, and color spaces: IM_RGB,
00189 * \ingroup imgclass */
00190 #define cdPutBitmap(_image, _x, _y, _w, _h, _xmin, _xmax, _ymin,
00191 {
00192     if (_image->color_space == IM_RGB)
00193         cdPutImageRectRGB(_image->width, _image->height,
00194                             (unsigned char*)_image->data[0],
00195                             (unsigned char*)_image->data[1],
00196                             (unsigned char*)_image->data[2],
00197                             _x, _y, _w, _h, _xmin, _xmax, _ymin, _
00198     else
00199         cdPutImageRectMap(_image->width, _image->height,
00200                             (unsigned char*)_image->data[0], _image-
00201                             _x, _y, _w, _h, _xmin, _xmax, _ymin, _
00202 }
00203
00204
00205 #if defined(__cplusplus)
00206 }
00207 #endif
00208
00209 #endif
```

include

im_plus.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief C++ Wrapper for File Access
00003  *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_PLUS_H
00009 #define __IM_PLUS_H
00010
00011
00012 /** \brief C++ Wrapper for the Image File Structure
00013  *
00014  * \par
00015  * Usage is just like the C API. Open and New are replaced by
00016  * Close is replaced by the destructor. Error checking is done
00017  * Open and New errors are checked using the Failed() member.
00018  * \ingroup file */
00019 class imImageFile
00020 {
00021     imFile* ifile;
00022     int error;
00023
00024     imImageFile() {};
00025
00026 public:
00027
00028     imImageFile(const char* file_name, const char* format)
00029         { this->ifile = imFileNew(file_name, format, &this->error) }
00030
00031     imImageFile(const char* file_name)
00032         { this->ifile = imFileOpen(file_name, &this->error); }
00033
00034     ~imImageFile()
00035         { if (this->ifile) imFileClose(this->ifile); }
00036
00037     int Failed()
00038         { return this->ifile == 0; }
00039
```

```
00040     int Error()
00041         { return this->error; }
00042
00043     void SetAttribute(const char* attrib, int data_type, int count)
00044         { imFileSetAttribute(this->ifile, attrib, data_type, count) }
00045
00046     const void* GetAttribute(const char* attrib, int *data_type,
00047         { return imFileGetAttribute(this->ifile, attrib, data_type) }
00048
00049     void GetInfo(char* format, char* compression, int *image_count)
00050         { imFileGetInfo(this->ifile, format, compression, image_count) }
00051
00052     void ReadImageInfo(int index, int *width, int *height, int *color_mode)
00053         { this->error = imFileReadImageInfo(this->ifile, index, width, height, color_mode) }
00054
00055     void GetPalette(long* palette, int *palette_count)
00056         { imFileGetPalette(this->ifile, palette, palette_count); }
00057
00058     void ReadImageData(void* data, int convert2bitmap, int color_mode)
00059         { this->error = imFileReadImageData(this->ifile, data, convert2bitmap, color_mode) }
00060
00061     void SetInfo(const char* compression)
00062         { imFileSetInfo(this->ifile, compression); }
00063
00064     void SetPalette(long* palette, int palette_count)
00065         { imFileSetPalette(this->ifile, palette, palette_count); }
00066
00067     void WriteImageInfo(int width, int height, int color_mode, int alpha)
00068         { this->error = imFileWriteImageInfo(this->ifile, width, height, color_mode, alpha) }
00069
00070     void WriteImageData(void* data)
00071         { this->error = imFileWriteImageData(this->ifile, data); }
00072     };
00073
00074 #endif
```

include

im.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Main API
00003 *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_H
00009 #define __IM_H
00010
00011 #include "im_lib.h"
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018 /** Image data type descriptors. \n
00019  * See also \ref datatypeutl.
00020  * \ingroup imagerep */
00021 enum imDataType
00022 {
00023     IM_BYTE,    /**< "unsigned char". 1 byte from 0 to 255.
00024     IM_USHORT,  /**< "unsigned short". 2 bytes from 0 to 65,535.
00025     IM_INT,     /**< "int". 4 bytes from -2,147,483,648 to 2,147,
00026     IM_FLOAT,   /**< "float". 4 bytes single precision IEEE float
00027     IM_CFLOAT   /**< complex "float". 2 float values in sequence,
00028 };
00029
00030 /** Image color mode color space descriptors (first byte). \n
00031  * See also \ref colormodeutl.
00032  * \ingroup imagerep */
00033 enum imColorSpace
00034 {
00035     IM_RGB,     /**< Red, Green and Blue (nonlinear).
00036     IM_MAP,     /**< Indexed by RGB color map (data_type=IM_BYTE)
00037     IM_GRAY,    /**< Shades of gray, luma (nonlinear Luminance),
00038     IM_BINARY,  /**< Indexed by 2 colors: black (0) and white (1)
00039     IM_CMYK,    /**< Cian, Magenta, Yellow and Black (nonlinear).
```

```
00040     IM_YCBCR,    /**< ITU-R 601 Y'CbCr. Y' is luma (nonlinear Lumi
00041     IM_LAB,      /**< CIE L*a*b*. L* is Lightness (nonlinear Lumin
00042     IM_LUV,      /**< CIE L*u*v*. L* is Lightness (nonlinear Lumin
00043     IM_XYZ       /**< CIE XYZ. Linear Light Tristimulus, Y is line
00044 };
00045
00046 /** Image color mode configuration/extr descriptors (1 bit ea
00047 * See also \ref colormodeutl.
00048 * \ingroup imagerep */
00049 enum imColorModeConfig
00050 {
00051     IM_ALPHA      = 0x100,   /**< adds an Alpha channel */
00052     IM_PACKED     = 0x200,   /**< packed components (rgbrgbgrgb...)
00053     IM_TOPDOWN    = 0x400   /**< orientation from top down to bott
00054 };
00055
00056
00057
00058 /** File Access Error Codes
00059 * \ingroup file */
00060 enum imErrorCodes
00061 {
00062     IM_ERR_NONE,    /**< No error. */
00063     IM_ERR_OPEN,    /**< Error while opening the file (read or
00064     IM_ERR_ACCESS, /**< Error while accessing the file (read o
00065     IM_ERR_FORMAT, /**< Invalid or unrecognized file format. *
00066     IM_ERR_DATA,   /**< Invalid or unsupported data. */
00067     IM_ERR_COMPRESS, /**< Invalid or unsupported compression. */
00068     IM_ERR_MEM,    /**< Insufficient memory */
00069     IM_ERR_COUNTER /**< Interrupted by the counter */
00070 };
00071
00072 /* Internal Image File Structure. */
00073 typedef struct _imFile imFile;
00074
00075 /** Opens the file for reading. It must exists. Also reads fil
00076 * \ingroup fileread */
00077 imFile* imFileOpen(const char* file_name, int *error);
00078
00079 /** Creates a new file for writing. If the file exists will be
00080 * It will only initialize the format driver and create the fi
00081 * \ingroup filewrite */
00082 imFile* imFileNew(const char* file_name, const char* format, i
00083
00084 /** Closes the file
00085 * \ingroup file */
00086 void imFileClose(imFile* ifile);
```

```
00087
00088 /** Returns the internal handle. It is file format dependent.
00089 * \ingroup file */
00090 void* imFileHandle(imFile* ifile);
00091
00092 /** Returns file information.
00093 * image_count is the number of images in a stack or
00094 * the number of frames in a video/animation or the depth of
00095 * compression and image_count can be NULL.
00096 * \ingroup fileread */
00097 void imFileGetInfo(imFile* ifile, char* format, char* compress
00098
00099 /** Changes the write compression method. \n
00100 * If the compression is not supported will return an error code.
00101 * Use NULL to set the default compression. You can use the im
00102 * but only after imFileWriteImageInfo. Only a few formats allow
00103 * \ingroup filewrite */
00104 void imFileSetInfo(imFile* ifile, const char* compression);
00105
00106 /** Changes an extended attribute. \n
00107 * The data will be internally duplicated. \n
00108 * If data is NULL the attribute is removed.
00109 * \ingroup file */
00110 void imFileSetAttribute(imFile* ifile, const char* attrib, int
00111
00112 /** Returns an extended attribute. \n
00113 * Returns NULL if not found. data_type and count can be NULL.
00114 * \ingroup file */
00115 const void* imFileGetAttribute(imFile* ifile, const char* attrib,
00116
00117 /** Returns a list of the attribute names. \n
00118 * "attrib" must contain room enough for "attrib_count" names.
00119 * \ingroup file */
00120 void imFileGetAttributeList(imFile* ifile, char** attrib, int
00121
00122 /** Returns the palette if any. \n
00123 * "palette" must be a 256 colors allocated array. \n
00124 * Returns zero in "palette_count" if there is no palette. "pa
00125 * \ingroup fileread */
00126 void imFileGetPalette(imFile* ifile, long* palette, int *palet
00127
00128 /** Changes the palette. \n
00129 * "palette_count" is >0 and <=256.
00130 * \ingroup filewrite */
00131 void imFileSetPalette(imFile* ifile, long* palette, int palett
00132
00133 /** Reads the image header if any and returns image information
```

```
00134 * Reads also the extended image attributes, so other image at
00135 * Returns an error code.
00136 * index specifies the image number between 0 and image_count-
00137 * Some drivers reads only in sequence, so "index" can be igno
00138 * Any parameters can be NULL. This function must be called at
00139 * \ingroup fileread */
00140 int imFileReadImageInfo(imFile* ifile, int index, int *width,
00141
00142 /** Writes the image header. Writes the file header at the fir
00143 * Writes also the extended image attributes. \n
00144 * Must call imFileSetPalette and set other attributes before
00145 * In some formats the color space will be converted to match
00146 * Returns an error code. This function must be called at leas
00147 * \ingroup filewrite */
00148 int imFileWriteImageInfo(imFile* ifile, int width, int height,
00149
00150 /** Reads the image data with or without conversion. \n
00151 * The data can be converted to bitmap when reading.
00152 * Data type conversion to byte will always scan for min-max t
00153 * except integer values that min-max are already between 0-25
00154 * Color mode flags contains packed, alpha and top-bottom inf
00155 * If flag is 0 means unpacked, no alpha and bottom up. If fla
00156 * Returns an error code.
00157 * \ingroup fileread */
00158 int imFileReadImageData(imFile* ifile, void* data, int convert
00159
00160 /** Writes the image data. \n
00161 * Returns an error code.
00162 * \ingroup filewrite */
00163 int imFileWriteImageData(imFile* ifile, void* data);
00164
00165
00166
00167 /** Returns a list of the registered formats. \n
00168 * format_list is an array of format identifiers.
00169 * Each format identifier is 10 chars max, maximum of 50 forma
00170 * You can use "char* format_list[50]".
00171 * \ingroup format */
00172 void imFormatList(char** format_list, int *format_count);
00173
00174 /** Returns the format description. \n
00175 * Format description is 50 chars max. \n
00176 * Extensions are separated like ".tif;*.tiff;", 50 chars max
00177 * Returns an error code. The parameters can be NULL, except f
00178 * \ingroup format */
00179 int imFormatInfo(const char* format, char* desc, char* ext, in
00180
```

```
00181 /** Returns the format compressions. \n
00182  * Compressions are 20 chars max each, maximum of 50 compressi
00183  * color_mode and data_type are optional, use -1 to ignore the
00184  * If you use them they will select only the allowed compressi
00185  * Returns an error code.
00186  * \ingroup format */
00187 int imFormatCompressions(const char* format, char** comp, int
00188
00189 /** Checks if the format suport the given image class at the g
00190  * Returns an error code.
00191  * \ingroup format */
00192 int imFormatCanWriteImage(const char* format, const char* comp
00193
00194
00195 #if defined(__cplusplus)
00196 }
00197 #endif
00198
00199 #include "old_im.h"
00200
00201 #endif
```

include

im_attrib_flat.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Attributes Table Flat API.
00003  * This will simplify the DLL export, and can be used for C ap-
00004  *
00005  * See Copyright Notice in im_lib.h
00006  * $Id: Exp $
00007 */
00008
00009 #ifndef __IM_ATTRIB_FLAT_H_
00010 #define __IM_ATTRIB_FLAT_H_
00011
00012 #if defined(__cplusplus)
00013 extern "C" {
00014 #endif
00015
00016 struct imAttribTablePrivate;
00017
00018 /** Definition of the callback used in ForEach function. */
00019 typedef int (*imAttribTableCallback)(void* user_data, int inde-
00020
00021 imAttribTablePrivate* imAttribTableCreate(int hash_size);
00022 void imAttribTableDestroy(imAttribTablePrivate* ptable);
00023 int imAttribTableCount(imAttribTablePrivate* ptable);
00024 void imAttribTableRemoveAll(imAttribTablePrivate* ptable);
00025 const void* imAttribTableGet(const imAttribTablePrivate* ptabl-
00026 void imAttribTableSet(imAttribTablePrivate* ptable, const char*
00027 void imAttribTableUnSet(imAttribTablePrivate* ptable, const char*
00028 void imAttribTableCopyFrom(imAttribTablePrivate* ptable_dst, c-
00029 void imAttribTableForEach(const imAttribTablePrivate* ptable,
00030
00031 imAttribTablePrivate* imAttribArrayCreate(int hash_size);
00032 const void* imAttribArrayGet(const imAttribTablePrivate* ptabl-
00033 void imAttribArraySet(imAttribTablePrivate* ptable, int index,
00034 void imAttribArrayCopyFrom(imAttribTablePrivate* ptable_dst, c-
00035
00036 #if defined(__cplusplus)
00037 }
00038 #endif
00039
```

00040 #endif

include

im_capture.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Video Capture
00003  *
00004  * See Copyright Notice in im.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_CAPTURE_H
00009 #define __IM_CAPTURE_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015 /** \defgroup capture Image Capture
00016  * \par
00017  * Captures images from video devices.
00018  * \par
00019  * You must link the application with "im_capture.lib/.a/.so".
00020  * Depends also on the Direct Show 9 SDK, you must link with "
00021  * When using the "im_capture.dll" this extra library is not n
00022  * Since DX uses COM, CoInitialize(NULL) is called when the de
00023  * \par
00024  * For more information: \n
00025  * http://msdn.microsoft.com/library/en-us/directx9_c/directx/
00026  * \par
00027  * See \ref im_capture.h
00028 */
00029
00030 typedef struct _imVideoCapture imVideoCapture;
00031
00032 /** Returns the number of available devices.
00033  * \ingroup capture */
00034 int imVideoCaptureDeviceCount(void);
00035
00036 /** Returns the device description. Returns NULL in the last d
00037  * \ingroup capture */
00038 const char* imVideoCaptureDeviceDesc(int device);
00039
```

```
00040 /** Reload the device list. The devices can be dynamically rem
00041 * \ingroup capture */
00042 int imVideoCaptureReloadDevices(void);
00043
00044 /** Creates a new imVideoCapture object. \n
00045 * Returns NULL if there is no capture device available. \n
00046 * In Windows returns NULL if DirectX version is older than 8.
00047 * \ingroup capture */
00048 imVideoCapture* imVideoCaptureCreate(void);
00049
00050 /** Destroys a imVideoCapture object.
00051 * \ingroup capture */
00052 void imVideoCaptureDestroy(imVideoCapture* vc);
00053
00054 /** Connects to a capture device.
00055 * More than one imVideoCapture object can be created
00056 * but they must be connected to different devices. \n
00057 * If the object is conected it will disconnect first. \n
00058 * Use -1 to return the current connected device,
00059 * in this case returns -1 if not connected. \n
00060 * Returns zero if failed.
00061 * \ingroup capture */
00062 int imVideoCaptureConnect(imVideoCapture* vc, int device);
00063
00064 /** Disconnect from a capture device.
00065 * \ingroup capture */
00066 void imVideoCaptureDisconnect(imVideoCapture* vc);
00067
00068 /** Displays a configuration modal dialog of the connected dev
00069 * In Windows, the capturing will be stopped in some cases. \n
00070 * In Windows parent is a HWND of a parent window, it can be N
00071 * Returns zero if failed.
00072 * \ingroup capture */
00073 int imVideoCaptureShowDialog(imVideoCapture* vc, int dialog, v
00074
00075 /** Returns the number of available configuration dialogs.
00076 * \ingroup capture */
00077 int imVideoCaptureDialogCount(imVideoCapture* vc);
00078
00079 /** Returns the description of a configuration dialog.
00080 * \ingroup capture */
00081 const char* imVideoCaptureDialogDesc(imVideoCapture* vc, int d
00082
00083 /** Returns the current image size of the connected device. \n
00084 * width and height returns 0 if not connected.
00085 * \ingroup capture */
00086 void imVideoCaptureGetImageSize(imVideoCapture* vc, int *width
```

```
00087
00088 /** Changes the image size of the connected device. \n
00089 * Returns zero if failed.
00090 * \ingroup capture */
00091 int imVideoCaptureSetImageSize(imVideoCapture* vc, int width,
00092
00093 /** Returns a new captured frame. Use -1 for infinite timeout.
00094 * Color space can be IM_RGB or IM_GRAY, and mode can be packed.
00095 * It can not have an alpha channel and orientation is always
00096 * Returns zero if failed or timeout expired, the buffer is not
00097 * \ingroup capture */
00098 int imVideoCaptureFrame(imVideoCapture* vc, unsigned char* dat
00099
00100 /** Start capturing, returns the new captured frame and stop c
00101 * This is more useful if you are switching between devices.
00102 * Data format is the same as imVideoCaptureFrame. \n
00103 * Returns zero if failed.
00104 * \ingroup capture */
00105 int imVideoCaptureOneFrame(imVideoCapture* vc, unsigned char*
00106
00107 /** Start capturing. \n
00108 * Use -1 to return the current state. \n
00109 * Returns zero if failed.
00110 * \ingroup capture */
00111 int imVideoCaptureLive(imVideoCapture* vc, int live);
00112
00113 /** Resets a camera or video attribute to the default value or
00114 * to the automatic setting. \n
00115 * Not all attributes support automatic modes. \n
00116 * Returns zero if failed.
00117 * \ingroup capture */
00118 int imVideoCaptureResetAttribute(imVideoCapture* vc, const char*
00119
00120 /** Returns a camera or video attribute in percentage of the v
00121 * Returns zero if failed.
00122 * \ingroup capture */
00123 int imVideoCaptureGetAttribute(imVideoCapture* vc, const char*
00124
00125 /** Changes a camera or video attribute in percentage of the v
00126 * Returns zero if failed.
00127 * \ingroup capture */
00128 int imVideoCaptureSetAttribute(imVideoCapture* vc, const char*
00129
00130 /** Returns a list of the description of the valid attributes.
00131 * \ingroup capture */
00132 const char** imVideoCaptureGetAttributeList(imVideoCapture* vc
00133
```

```
00134
00135 /** \defgroup winattrib Windows Attributes
00136 \verbatim
00137     VideoBrightness - Specifies the brightness, also called the
00138     VideoContrast - Specifies the contrast, expressed as gain fa
00139     VideoHue - Specifies the hue angle.
00140     VideoSaturation - Specifies the saturation.
00141     VideoSharpness - Specifies the sharpness.
00142     VideoGamma - Specifies the gamma.
00143     VideoColorEnable - Specifies the color enable setting. (0/10
00144     VideoWhiteBalance - Specifies the white balance, as a color
00145     VideoBacklightCompensation - Specifies the backlight compens
00146     VideoGain - Specifies the gain adjustment.
00147     CameraPanAngle - Specifies the camera's pan angle. To 100 ro
00148     CameraTiltAngle - Specifies the camera's tilt angle. To 100
00149     CameraRollAngle - Specifies the camera's roll angle. To 100
00150     CameraLensZoom - Specifies the camera's zoom setting.
00151     CameraExposure - Specifies the exposure setting.
00152     CameraIris - Specifies the camera's iris setting.
00153     CameraFocus - Specifies the camera's focus setting, as the d
00154     FlipHorizontal - Specifies the video will be flipped in the
00155     FlipVertical - Specifies the video will be flipped in the ve
00156     AnalogFormat - Specifies the video format standard NTSC, PAL
00157         NTSC_M      = 0
00158         NTSC_M_J    = 1
00159         NTSC_433    = 2
00160         PAL_B       = 3
00161         PAL_D       = 4
00162         PAL_H       = 5
00163         PAL_I       = 6
00164         PAL_M       = 7
00165         PAL_N       = 8
00166         PAL_60      = 9
00167         SECAM_B     = 10
00168         SECAM_D     = 11
00169         SECAM_G     = 12
00170         SECAM_H     = 13
00171         SECAM_K     = 14
00172         SECAM_K1    = 15
00173         SECAM_L     = 16
00174         SECAM_L1    = 17
00175         PAL_N_COMBO = 18
00176 \endverbatim
00177 * \ingroup capture */
00178
00179
00180 #if defined(__cplusplus)
```

```
00181 }
00182
00183 /** A C++ Wrapper for the imVideoCapture structure functions.
00184 * \ingroup capture */
00185 class imCapture
00186 {
00187 public:
00188     imCapture()
00189     { vc = imVideoCaptureCreate(); }
00190
00191     ~imCapture()
00192     { if (vc) imVideoCaptureDestroy(vc); }
00193
00194     int Failed()
00195     { if (!vc) return 0; else return 1; }
00196
00197     int Connect(int device)
00198     { return imVideoCaptureConnect(vc, device); }
00199
00200     void Disconnect()
00201     { imVideoCaptureDisconnect(vc); }
00202
00203     int ShowDialog(int dialog, void* parent)
00204     { return imVideoCaptureShowDialog(vc, dialog, parent); }
00205
00206     int DialogCount()
00207     { return imVideoCaptureDialogCount(vc); }
00208
00209     const char* DialogDescription(int dialog)
00210     { return imVideoCaptureDialogDesc(vc, dialog); }
00211
00212     void GetImageSize(int *width, int *height)
00213     { imVideoCaptureGetImageSize(vc, width, height); }
00214
00215     int SetImageSize(int width, int height)
00216     { return imVideoCaptureSetImageSize(vc, width, height); }
00217
00218     int GetFrame(unsigned char* data, int color_mode, int timeout)
00219     { return imVideoCaptureFrame(vc, data, color_mode, timeout); }
00220
00221     int GetOneFrame(unsigned char* data, int color_mode)
00222     { return imVideoCaptureOneFrame(vc, data, color_mode); }
00223
00224     int Live(int live)
00225     { return imVideoCaptureLive(vc, live); }
00226
00227     int ResetAttribute(const char* attrib, int fauto)
```

```
00228     { return imVideoCaptureResetAttribute(vc, attrib, fauto);  
00229  
00230     int GetAttribute(const char* attrib, float *percent)  
00231         { return imVideoCaptureGetAttribute(vc, attrib, percent);  
00232  
00233     int SetAttribute(const char* attrib, float percent)  
00234         { return imVideoCaptureSetAttribute(vc, attrib, percent);  
00235  
00236     const char** GetAttributeList(int *num_attrib)  
00237         { return imVideoCaptureGetAttributeList(vc, num_attrib); }  
00238  
00239 protected:  
00240     imVideoCapture* vc;  
00241 };  
00242  
00243 #endif  
00244  
00245 #endif
```

include

im_color.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Color Manipulation
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_COLOR_H
00009 #define __IM_COLOR_H
00010
00011 #include "im_math.h"
00012
00013 /** \defgroup color Color Manipulation
00014 *
00015 * \par
00016 * Functions to convert from one color space to another,
00017 * and color gamut utilities.
00018 * \par
00019 * See \ref im_color.h
00020 *
00021 * \section s1 Some Color Science
00022 * \par
00023 * Y is luminance, a linear-light quantity.
00024 * It is directly proportional to physical intensity
00025 * weighted by the spectral sensitivity of human vision.
00026 * \par
00027 * L* is lightness, a nonlinear luminance
00028 * that approximates the perception of brightness.
00029 * It is nearly perceptual uniform.
00030 * It has a range of 0 to 100.
00031 * \par
00032 * Y' is luma, a nonlinear luminance that approximates lightness
00033 * \par
00034 * Brightness is a visual sensation according to which an area
00035 * appears to exhibit more or less light.
00036 * It is a subjective quantity and can not be measured.
00037 * \par
00038 * One unit of euclidian distance in CIE L*u*v* or CIE L*a*b*
00039 * roughly to a just-noticeable difference (JND) of color.
```

```

00040  * \par
00041 \verbatim
00042 ChromaUV = sqrt(u*u + v*v)
00043 HueUV = atan2(v, u)
00044 SaturationUV = ChromaUV / L      (called psychometric saturation)
00045 (the same can be calculated for Lab)
00046 \endverbatim
00047  * \par
00048  * IEC 61966-2.1 Default RGB colour space - SRGB
00049  * \li ITU-R Recommendation BT.709 (D65 white point).
00050  * \li D65 White Point (X,Y,Z) = (0.9505 1.0000 1.0890)
00051  * \par
00052  * Documentation extracted from Charles Poynton - Digital Video
00053  *
00054  * \section Links
00055  * \li www.color.org - ICC
00056  * \li www.srgb.com - SRGB
00057  * \li www.poynton.com - Charles Poynton
00058  * \li www.littlecms.com - A free Color Management System (useful)
00059  *
00060  * \section cci Color Component Intervals
00061  * \par
00062  * All the color components are stored in the 0-max interval,
00063  * Here are the pre-defined intervals for each data type. These
00064  * You should normalize data before converting between color spaces.
00065  * \par
00066 \verbatim
00067 byte [0,255] or [-128,+127] (1 byte)
00068 ushort [0,65535] or [-32768,+32767] (2 bytes)
00069 int [0,16777215] or [-8388608,+8388607] (3 bytes)
00070 float [0,1] or [-0.5,+0.5] (4 bytes)
00071 \endverbatim
00072  * \ingroup util */
00073
00074 /** Returns the zero value for color conversion purposes. \n
00075  * This is a value to be compensated when the data_type is unsigned.
00076  * \ingroup color */
00077 inline float imColorZero(int data_type)
00078 {
00079     float zero[] = {128.0f, 32768.0f, 8388608.0f, 0.5f};
00080     return zero[data_type];
00081 }
00082
00083 /** Returns the maximum value for color conversion purposes.
00084  * \ingroup color */
00085 inline int imColorMax(int data_type)
00086 {

```



```

00134 {
00135     float r = float(Y + 1.402f * (Cr - ze
00136     float g = float(Y - 0.344f * (Cb - zero) - 0.714f * (Cr - ze
00137     float b = float(Y + 1.772f * (Cb - zero));
00138
00139     // now we should enforce 0<= rgb <= max
00140
00141     R = (T)IM_CROPMAX(r, max);
00142     G = (T)IM_CROPMAX(g, max);
00143     B = (T)IM_CROPMAX(b, max);
00144 }
00145
00146 /** Converts R'G'B' to Y'CbCr (all nonlinear). \n
00147 * ITU-R Recommendation 601-1 with no headroom/footroom.
00148 \verbatim
00149 0 <= Y <= 1 ; -0.5 <= CbCr <= 0.5 ; 0 <= RGB <= 1
00150
00151 Y' = 0.299 *R' + 0.587 *G' + 0.114 *B'
00152 Cb = -0.169 *R' - 0.331 *G' + 0.500 *B'
00153 Cr = 0.500 *R' - 0.419 *G' - 0.081 *B'
00154 \endverbatim
00155 * \ingroup color */
00156 template <class T>
00157 inline void imColorRGB2YCbCr(const T R, const T G, const T B,
00158                                     T& Y, T& Cb, T& Cr,
00159                                     const T& zero)
00160 {
00161     Y = (T)( 0.299f *R + 0.587f *G + 0.114f *B);
00162     Cb = (T)(-0.169f *R - 0.331f *G + 0.500f *B + (float)zero);
00163     Cr = (T)( 0.500f *R - 0.419f *G - 0.081f *B + (float)zero);
00164
00165     // there is no need for cropping here, YCrCr is already at t
00166 }
00167
00168 /** Converts C'M'Y'K' to R'G'B' (all nonlinear). \n
00169 * This is a poor conversion that works for a simple visualiza
00170 \verbatim
00171 0 <= CMYK <= 1 ; 0 <= RGB <= 1
00172
00173     R = (1 - K) * (1 - C)
00174     G = (1 - K) * (1 - M)
00175     B = (1 - K) * (1 - Y)
00176 \endverbatim
00177 * \ingroup color */
00178 template <class T>
00179 inline void imColorCMYK2RGB(const T C, const T M, const T Y, c
00180                                     T& R, T& G, T& B, const T& max)

```



```

00228 {
00229     X = (T)(0.4124f *R + 0.3576f *G + 0.1805f *B);
00230     Y = (T)(0.2126f *R + 0.7152f *G + 0.0722f *B);
00231     Z = (T)(0.0193f *R + 0.1192f *G + 0.9505f *B);
00232
00233     // there is no need for cropping here, XYZ is already at the
00234 }
00235
00236 #define IM_FWLAD(_w) (_w > 0.008856f? \
00237                         powf(_w, 1.0f/3.0f): \
00238                         7.787f * _w + 0.16f/1.16f)
00239
00240 /** Converts CIE XYZ (linear) to CIE L*a*b* (nonlinear). \
00241 * The white point is D65. \
00242 \verbatim
00243     0 <= L <= 1 ; -0.5 <= ab <= +0.5 ; 0 <= XYZ <= 1
00244
00245     if (t > 0.008856)
00246         f(t) = pow(t, 1/3)
00247     else
00248         f(t) = 7.787*t + 16/116
00249
00250     fX = f(X / Xn)      fY = f(Y / Yn)      fZ = f(Z / Zn)
00251
00252     L = 1.16 * fY - 0.16
00253     a = 2.5 * (fX - fY)
00254     b = (fY - fZ)
00255
00256 \endverbatim
00257 * \ingroup color */
00258 inline void imColorXYZ2Lab(const float X, const float Y, const
00259                               float& L, float& a, float& b)
00260 {
00261     float fX = X / 0.9505f; // white point D65
00262     float fY = Y / 1.0f;
00263     float fZ = Z / 1.0890f;
00264
00265     fX = IM_FWLAD(fX);
00266     fY = IM_FWLAD(fY);
00267     fZ = IM_FWLAD(fZ);
00268
00269     L = 1.16f * fY - 0.16f;
00270     a = 2.5f * (fX - fY);
00271     b = (fY - fZ);
00272 }
00273
00274 #define IM_GWLAD(_w)  (_w > 0.20689f? \

```

```

00275          powf(_w, 3.0f):                                \
00276          0.1284f * (_w - 0.16f/1.16f))
00277
00278 /** Converts CIE L*a*b* (nonlinear) to CIE XYZ (linear). \n
00279 * The white point is D65. \n
00280 * 0 <= L <= 1 ; -0.5 <= ab <= +0.5 ; 0 <= XYZ <= 1
00281 * \ingroup color */
00282 inline void imColorLab2XYZ(const float L, const float a, const
00283                               float& X, float& Y, float& Z)
00284
00285 {
00286     float fY = (L + 0.16f) / 1.16f;
00287     float gY = IM_GWLAB(fY);
00288
00289     float fgY = IM_FWLAB(gY);
00290     float gX = fgY + a / 2.5f;
00291     float gZ = fgY - b;
00292     gX = IM_GWLAB(gX);
00293     gZ = IM_GWLAB(gZ);
00294
00295     X = gX * 0.9505f;      // white point D65
00296     Y = gY * 1.0f;
00297     Z = gZ * 1.0890f;
00298 }
00299
00300 /** Converts CIE XYZ (linear) to CIE L*u*v* (nonlinear). \n
00301 * The white point is D65. \n
00302 \verbatim
00303     0 <= L <= 1 ; -1 <= uv <= +1 ; 0 <= XYZ <= 1
00304
00305     Y = Y / 1.0      (for D65)
00306     if (Y > 0.008856)
00307         fY = pow(Y, 1/3)
00308     else
00309         fY = 7.787 * Y + 0.16/1.16
00310     L = 1.16 * fY - 0.16
00311
00312     U(x, y, z) = (4 * x)/(x + 15 * y + 3 * z)
00313     V(x, y, z) = (9 * x)/(x + 15 * y + 3 * z)
00314     un = U(Xn, Yn, Zn) = 0.1978      (for D65)
00315     vn = V(Xn, Yn, Zn) = 0.4683      (for D65)
00316     fu = U(X, Y, Z)
00317     fv = V(X, Y, Z)
00318
00319     u = 13 * L * (fu - un)
00320     v = 13 * L * (fv - vn)
00321 \endverbatim

```

```

00322  * \ingroup color */
00323  inline void imColorXYZ2Luv(const float X, const float Y, const
00324                                float& L, float& u, float& v)
00325  {
00326      float XYZ = (float)(X + 15 * Y + 3 * Z);
00327      float fY = Y / 1.0f;
00328
00329      if (XYZ != 0)
00330      {
00331          L = 1.16f * IM_FWLAB(fY) - 0.16f;
00332          u = 6.5f * L * ((4 * X)/XYZ - 0.1978f);
00333          v = 6.5f * L * ((9 * Y)/XYZ - 0.4683f);
00334      }
00335      else
00336      {
00337          L = u = v = 0;
00338      }
00339  }
00340
00341 /** Converts CIE L*u*v* (nonlinear) to CIE XYZ (linear). \n
00342 * The white point is D65.
00343 * 0 <= L <= 1 ; -0.5 <= uv <= +0.5 ; 0 <= XYZ <= 1 \n
00344 * \ingroup color */
00345  inline void imColorLuv2XYZ(const float L, const float u, const
00346                                float& X, float& Y, float& Z)
00347
00348  {
00349      float fY = (L + 0.16f) / 1.16f;
00350      Y = IM_GWLAB(fY) * 1.0f;
00351
00352      float ul = 0.1978f, vl = 0.4683f;
00353      if (L != 0)
00354      {
00355          ul = u / (6.5f * L) + 0.1978f;
00356          vl = v / (6.5f * L) + 0.4683f;
00357      }
00358
00359      X = ((9 * ul) / (4 * vl)) * Y;
00360      Z = ((12 - 3 * ul - 20 * vl) / (4 * vl)) * Y;
00361  }
00362
00363 /** Converts nonlinear values to linear values. \n
00364 * We use the sRGB transfer function. sRGB uses ITU-R 709 prim
00365 \verbatim
00366 0 <= l <= 1 ; 0 <= v <= 1
00367
00368  if (v < 0.03928)

```

```

00369     l = v / 12.92
00370   else
00371     l = pow((v + 0.055) / 1.055, 2.4)
00372 \endverbatim
00373 * \ingroup color */
00374 inline float imColorTransfer2Linear(const float& nonlinear_val
00375 {
00376   if (nonlinear_value < 0.03928f)
00377     return nonlinear_value / 12.92f;
00378   else
00379     return powf((nonlinear_value + 0.055f) / 1.055f, 2.4f);
00380 }
00381
00382 /** Converts linear values to nonlinear values. \n
00383 * We use the sRGB transfer function. sRGB uses ITU-R 709 prim
00384 \verbatim
00385 0 <= l <= 1 ; 0 <= v <= 1
00386
00387   if (l < 0.0031308)
00388     v = 12.92 * l
00389   else
00390     v = 1.055 * pow(l, 1/2.4) - 0.055
00391 \endverbatim
00392 * \ingroup color */
00393 inline float imColorTransfer2Nonlinear(const float& value)
00394 {
00395   if (value < 0.0031308f)
00396     return 12.92f * value;
00397   else
00398     return 1.055f * powf(value, 1.0f/2.4f) - 0.055f;
00399 }
00400
00401 /** Converts RGB (linear) to R'G'B' (nonlinear).
00402 * \ingroup color */
00403 inline void imColorRGB2RGBNonlinear(const float RL, const float GL,
00404                                         float& R, float& G, float&
00405 {
00406   R = imColorTransfer2Nonlinear(RL);
00407   G = imColorTransfer2Nonlinear(GL);
00408   B = imColorTransfer2Nonlinear(BL);
00409 }
00410
00411 /** Converts R'G'B' to Y' (all nonlinear). \n
00412 \verbatim
00413 Y' = 0.299 *R' + 0.587 *G' + 0.114 *B'
00414 \endverbatim
00415 * \ingroup color */

```

```

00416 template <class T>
00417 inline T imColorRGB2Luma(const T R, const T G, const T B)
00418 {
00419     return (T)((299 * R + 587 * G + 114 * B) / 1000);
00420 }
00421
00422 /** Converts Luminance (CIE Y) to Lightness (CIE L*) (all line
00423 * The white point is D65.
00424 \verbatim
00425     0 <= Y <= 1 ; 0 <= L* <= 1
00426
00427     Y = Y / 1.0      (for D65)
00428     if (Y > 0.008856)
00429         fY = pow(Y, 1/3)
00430     else
00431         fY = 7.787 * Y + 0.16/1.16
00432     L = 1.16 * fY - 0.16
00433 \endverbatim
00434 * \ingroup color */
00435 inline float imColorLuminance2Lightness(const float& Y)
00436 {
00437     return 1.16f * IM_FWLAB(Y) - 0.16f;
00438 }
00439
00440 /** Converts Lightness (CIE L*) to Luminance (CIE Y) (all line
00441 * The white point is D65.
00442 \verbatim
00443     0 <= Y <= 1 ; 0 <= L* <= 1
00444
00445     fY = (L + 0.16)/1.16
00446     if (fY > 0.20689)
00447         Y = pow(fY, 3)
00448     else
00449         Y = 0.1284 * (fY - 0.16/1.16)
00450     Y = Y * 1.0      (for D65)
00451 \endverbatim
00452 * \ingroup color */
00453 inline float imColorLightness2Luminance(const float& L)
00454 {
00455     float fY = (L + 0.16f) / 1.16f;
00456     return IM_GWLAB(fY);
00457 }
00458
00459 #undef IM_FWLAB
00460 #undef IM_GWLAB
00461 #undef IM_CROPL
00462 #undef IM_CROPC

```

00463

00464 #endif

include

im_colorhsih.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief HSI Color Manipulation
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_COLORHSI_H
00009 #define __IM_COLORHSI_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup hsi HSI Color Coordinate System Conversions
00017 *
00018 * \par
00019 * HSI is just the RGB color space written in a different coor
00020 * \par
00021 * "I" is the cube diagonal. HS is a polar coordinates of a pl
00022 * "S" is the distance from the diagonal. "H" is the angle fro
00023 * \par
00024 * This is not a new color space, this is exactly the same gam
00025 * Since it is still a cube, Smax depends on H.
00026 * \par
00027 * See \ref im_colorhsih.h
00028 * \ingroup color */
00029
00030
00031 /** Returns the maximum S for H (in radians) and I.
00032 * \ingroup hsi */
00033 float imColorHSI_Smax(float h, double cosh, double sinh, float
00034
00035 /** Returns I where S is maximum given H (in radians).
00036 * \ingroup hsi */
00037 float imColorHSI_ImaxS(float h, double cosh, double sinh);
00038
00039 /** Converts from RGB to HSI.
```

```
00040  * \ingroup hsi */
00041 void imColorRGB2HSI(float r, float g, float b, float *h, float
00042
00043 /** Converts from RGB (byte) to HSI.
00044  * \ingroup hsi */
00045 void imColorRGB2HSIbyte(unsigned char r, unsigned char g, unsi
00046
00047 /** Converts from HSI to RGB.
00048  * \ingroup hsi */
00049 void imColorHSI2RGB(float h, float s, float i, float *r, float
00050
00051 /** Converts from HSI to RGB (byte).
00052  * \ingroup hsi */
00053 void imColorHSI2RGBbyte(float h, float s, float i, unsigned ch
00054
00055
00056 #if defined(__cplusplus)
00057 }
00058 #endif
00059
00060 #endif
```

include

im_convert.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Image Conversion
00003  *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_CONVERT_H
00009 #define __IM_CONVERT_H
00010
00011 #include "im_image.h"
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018 /** \defgroup convert Image Conversion
00019  * \par
00020  * Converts one type of image into another. Can convert between
00021  * and between data types.
00022  * \par
00023  * See \ref im_convert.h
00024  * \ingroup imagerep */
00025
00026
00027 /** Complex to real conversions
00028  * \ingroup convert */
00029 enum imComplex2Real
00030 {
00031     IM_CPX_REAL,
00032     IM_CPX_IMAG,
00033     IM_CPX_MAG,
00034     IM_CPX_PHASE
00035 };
00036
00037 /** Predefined Gamma factors
00038  * \ingroup convert */
00039 enum imGammaFactor
```

```

00040 {
00041     IM_GAMMA_LINEAR    = 0,
00042     IM_GAMMA_LOGLITE   = -10,
00043     IM_GAMMA_LOGHEAVY = -1000,
00044     IM_GAMMA_EXPLITE  = 2,
00045     IM_GAMMA_EXPHEAVY = 7
00046 };
00047
00048 /** Predefined Cast Modes
00049 * \ingroup convert */
00050 enum imCastMode
00051 {
00052     IM_CAST_MINMAX, /*< scan for min and max values */
00053     IM_CAST_FIXED,  /*< use predefined 0-max values, see \ref co
00054     IM_CAST_DIRECT  /*< direct type cast the value. Only byte a
00055 };
00056
00057 /** Changes the image data type, using a complex2real conversi
00058 * a gamma factor, and an absolute mode (modulus). \n
00059 * When demoting the data type the function will scan for min/
00060 * to scale the result according to the destiny range. \n
00061 * Except complex to real that will use only the complex2real
00062 * Images must be of the same size and color mode. \n
00063 * Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.
00064 * \ingroup convert */
00065 int imConvertDataType(const imImage* src_image, imImage* dst_i
00066
00067 /** Converts one color space to another. Images must be of the
00068 * CMYK can be converted to RGB only, and it is a very simple
00069 * All colors can be converted to Binary, the non zero gray va
00070 * RGB to Map uses the median cut implementation from the free
00071 * All other color space conversions assume sRGB and CIE defin
00072 * Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.
00073 * \ingroup convert */
00074 int imConvertColorSpace(const imImage* src_image, imImage* dst
00075
00076 /** Converts the image to its bitmap equivalent,
00077 * uses \ref imConvertColorSpace and \ref imConvertDataType. \
00078 * Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.
00079 * \ingroup convert */
00080 int imConvertToBitmap(const imImage* src_image, imImage* dst_i
00081
00082 /** Changes the packing of the data buffer.
00083 * \ingroup convert */
00084 void imConvertPacking(const void* src_data, void* dst_data, in
00085
00086 /** Changes in-place a MAP data into a RGB data. The data must

```

```
00087 * depth can be 3 or 4. count=width*height. \n
00088 * Very usefull for OpenGL applications.
00089 * \ingroup convert */
00090 void imConvertMapToRGB(unsigned char* data, int count, int dep
00091
00092 /* Converts a RGB bitmap into a map bitmap using the median cu
00093 * Used only "im_convertcolor.cpp" implemented in "im_rgb2map.
00094 * Internal function kept here because of the compatibility mo
00095 int imConvertRGB2Map(int width, int height,
00096                     unsigned char *red, unsigned char *green, unsign
00097                     unsigned char *map, long *palette, int *palette_
00098
00099
00100 #if defined(__cplusplus)
00101 }
00102 #endif
00103
00104 #endif
```

include

im_counter.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Processing Counter
00003  *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_COUNTER_H
00009 #define __IM_COUNTER_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup counter Counter
00017  * \par
00018  * Used to notify the application that a step in the loading,
00019  * \par
00020  * See \ref im_counter.h
00021  * \ingroup util */
00022
00023 /** Counter callback, informs the progress of the operation to
00024  * Text contains a constant string that is NULL during normal
00025  * and a message in the begining of a count.
00026  * Counter id identifies different counters. \n
00027  * Progress in a count reports a value from 0 to 1000. If -1 is
00028  * If returns 0 the client should abort the operation. \n
00029  * If the counter is aborted, the callback will be called one
00030  * \ingroup counter */
00031 typedef int (*imCounterCallback)(int counter, void* user_data,
00032
00033 /** Changes the counter callback. Returns old callback. \n
00034  * User data is changed only if not NULL.
00035  * \ingroup counter */
00036 imCounterCallback imCounterSetCallback(void* user_data, imCount
00037
00038 /** Begins a new count, or a partial-count in a sequence. \n
00039  * Calls the callback with "-1" and text=title, if it is at the
```

```
00040 * This is to be used by the operations. Returns a counter Id.
00041 * \ingroup counter */
00042 int imCounterBegin(const char* title);
00043
00044 /** Ends a count, or a partial-count in a sequence. \n
00045 * Calls the callback with "1001", text=null, and releases the
00046 * \ingroup counter */
00047 void imCounterEnd(int counter);
00048
00049 /** Increments a count. Must set the total first. \n
00050 * Calls the callback, text=message if it is the first increme
00051 * Returns 0 if the callback aborted, 1 if returns normally.
00052 * \ingroup counter */
00053 int imCounterInc(int counter);
00054
00055 /** Sets the total increments of a count.
00056 * \ingroup counter */
00057 void imCounterTotal(int counter, int total, const char* messag
00058
00059
00060 #if defined(__cplusplus)
00061 }
00062 #endif
00063
00064 #endif
```

include

im_format_all.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief All the Internal File Formats.
00003 * They are all automatically registered by the library.
00004 * The signatures are in C, but the functions are C++.
00005 * Header for internal use only.
00006 *
00007 * See Copyright Notice in im_lib.h
00008 * $Id: Exp $
00009 */
00010
00011 #ifndef __IM_FORMAT_ALL_H
00012 #define __IM_FORMAT_ALL_H
00013
00014 #if defined(__cplusplus)
00015 extern "C" {
00016 #endif
00017
00018 /** \defgroup tiff TIFF - Tagged Image File Format
00019 * \section Description
00020 *
00021 * \par
00022 * Copyright (c) 1986-1988, 1992 by Adobe Systems Incorporated
00023 * Originally created by a group of companies,
00024 * the Aldus Corporation kepted the copyright until Aldus was
00025 * TIFF Revision 6.0 Final June 3, 1992 \n
00026 * http://www.adobe.com/Support/TechNotes.html
00027 * \par
00028 * Access to the TIFF file format uses libTIFF version 3.6.1 \
00029 * http://www.libtiff.org \n
00030 * Copyright (c) 1988-1997 Sam Leffler \n
00031 * Copyright (c) 1991-1997 Silicon Graphics, Inc. \n
00032 *
00033 * \section Features
00034 *
00035 \verbatim
00036     Data Types: <all>
00037     Color Spaces: Gray, RGB, CMYK, YCbCr, Lab, XYZ, Map and Bi
00038     Compressions:
00039         NONE - no compression [default for IEEE Floating Point
```

00040 CCITTRLE - CCITT modified Huffman RLE (binary only) [def
00041 CCITTFAX3 - CCITT Group 3 fax (binary only)
00042 CCITTFAX4 - CCITT Group 4 fax (binary only)
00043 LZW - Lempel-Ziv & Welch [default]
00044 JPEG - ISO JPEG [default for YCBCR]
00045 NEXT - NeXT 2-bit RLE (2 bpp only)
00046 CCITTRLEW - CCITT modified Huffman RLE with word alignme
00047 RLE - Packbits (Macintosh RLE) [default for MAP]
00048 THUNDERSCAN - ThunderScan 4-bit RLE (only for 2 or 4 bpp)
00049 PIXARLOG - Pixar companded 11-bit ZIP (only byte, ushort)
00050 DEFLATE - LZ77 variation (ZIP)
00051 ADOBE_DEFLATE - Adobe LZ77 variation
00052 SGILOG - SGI Log Luminance RLE for L and Luv (only byte,
00053 SGILOG24 - SGI Log 24-bit packed for Luv (only byte, ushort)
00054 Can have more than one image.
00055 Can have an alpha channel.
00056 Components can be packed or not.
00057 Lines arranged from top down to bottom or bottom up to top
00058 Handle() returns a TIFF* of libTIFF.
00059
00060 Attributes:
00061 Photometric IM_USHORT (1) (when writing this will comple
00062 ExtraSampleInfo IM_USHORT (1) (description of alpha chan
00063 JPEGQuality IM_INT (1) [0-100, default 75] (write only)
00064 ZIPQuality IM_INT (1) [1-9, default 6] (write only)
00065 ResolutionUnit (string) ["DPC", "DPI"]
00066 XResolution, YResolution IM_FLOAT (1)
00067 Description, Author, Copyright, DateTime, DocumentName,
00068 PageName, TargetPrinter, Make, Model, Software, HostComp
00069 InkNames (strings separated by '\0's)
00070 InkSet IM_USHORT (1)
00071 NumberOfInks IM_USHORT (1)
00072 DotRange IM_USHORT (2)
00073 TransferFunction0, TransferFunction1, TransferFunction3
00074 ReferenceBlackWhite IMFLOAT (6)
00075 WhitePoint IMFLOAT (2)
00076 PrimaryChromaticities IMFLOAT (6)
00077 YCbCrCoefficients IM_FLOAT (3)
00078 YCbCrSubSampling IM_USHORT (2)
00079 YCbCrPositioning IM_USHORT (1)
00080 PageNumber IM_USHORT (2)
00081 StoNits IM_FLOAT (1)
00082 XPosition, YPosition IM_FLOAT (1)
00083 SMinSampleValue, SMaxSampleValue IM_FLOAT (1)
00084 HalftoneHints IM_USHORT (2)
00085 SubfileType IM_INT (1)
00086 ICCProfile IM_BYTE (N)

```
00087     GeoTiePoints, GeoTransMatrix, IntergraphMatrix, GeoPixel
00088     GeoASCIIParams (string)
00089     (other attributes can be obtained by using libTIFF direc
00090
00091 Comments:
00092     LogLuv is in fact Y'+CIE(u,v), so we choose to convert t
00093     SubIFD is not handled.
00094     Since LZW patent expired, we use the libtiff-lzw-compres
00095     LZW Copyright Unisys.
00096     libGeoTIFF can be used without XTIFF initialization. Use
00097     Must define in the makefile: JPEG_SUPPORT, ZIP_SUPPORT,
00098     If your system does not have the definitions u_char, u_s
00099     you must define BSDTYPES in the makefile when compilin
00100     Our include file "port.h" simply includes "tiffcomp.h".
00101     Changed "tiff_jpeg.c" - commented "downsampled_output =
00102     New file tiff_binfile.c
00103 \endverbatim
00104 * \ingroup format */
00105 void imFormatRegisterTIFF(void);
00106
00107 /** \defgroup jpeg JPEG - JPEG File Interchange Format
00108 * \section Description
00109 *
00110 * \par
00111 * ISO/IEC 10918 (1994, 1995, 1997, 1999)\n
00112 * http://www.jpeg.org/
00113 * \par
00114 * Access to the JPEG file format uses libJPEG version 6b. \n
00115 * http://www.ijg.org\n
00116 * Copyright (C) 1991-1998, Thomas G. Lane\n
00117 * from the Independent JPEG Group.\n
00118 * \par
00119 * Access to the EXIF attributes uses libEXIF version 0.5.12.
00120 * http://sourceforge.net/projects/libexif
00121 * Copyright (C) 2001-2003, Lutz Müller
00122 *
00123 * \section Features
00124 *
00125 \verbatim
00126     Data Types: Byte
00127     Color Spaces: Gray, RGB, CMYK and YCbCr (Binary Saved as G
00128     Compressions:
00129         JPEG - ISO JPEG [default]
00130     Only one image.
00131     No alpha channel.
00132     Internally the components are always packed.
00133     Internally the lines are arranged from top down to bottom.
```

```
00134     Handle() returns jpeg_decompress_struct* when reading, and
00135         jpeg_compress_struct* when writing.
00136
00137     Attributes:
00138         JPEGQuality IM_INT (1) [0-100, default 75] (write only)
00139         ResolutionUnit (string) ["DPC", "DPI"]
00140         XResolution, YResolution IM_FLOAT (1)
00141         Interlaced (same as Progressive) IM_INT (1 | 0) default
00142         Description (string)
00143         (lots of Exif tags)
00144
00145     Changes to libJPEG:
00146         jdatadst.c - fflush and ferror replaced by macros JFFLUS
00147         jinclude.h - standard JFFLUSH and JFERROR definitions, a
00148         jmorecfg.h - changed definition of INT32 to JINT32 for b
00149         new file created: jconfig.h
00150
00151     Changes to libEXIF:
00152         new file config.h
00153         changed "exif-tag.c" to add new function
00154         changed "exif-entry.c" to improve exif_entry_initialize
00155
00156     Comments:
00157         Other APPx markers are ignored.
00158         No thumbnail support.
00159 \endverbatim
00160 * \ingroup format */
00161 void imFormatRegisterJPEG(void);
00162
00163 /** \defgroup png PNG - Portable Network Graphic Format
00164 * \section Description
00165 *
00166 * \par
00167 * Access to the PNG file format uses libPNG version 1.2.5. \n
00168 * http://www.libpng.org \n
00169 * Copyright (C) 2000-2002 Glenn Randers-Pehrson
00170 * \par
00171 * Deflate compression support uses zlib version 1.2.1. \n
00172 * http://www.zlib.org \n
00173 * Copyright (C) 1995-2003 Jean-loup Gailly and Mark Adler
00174 *
00175 * \section Features
00176 *
00177 \verbatim
00178     Data Types: Byte and UShort
00179     Color Spaces: Gray, RGB, MAP and Binary
00180     Compressions:
```

```
00181      DEFLATE - LZ77 variation (ZIP) [default]
00182      Only one image.
00183      Can have an alpha channel.
00184      Internally the components are always packed.
00185      Internally the lines are arranged from top down to bottom.
00186      Handle() returns png_structp
00187
00188      Attributes:
00189          ZIPQuality IM_INT (1) [1-9, default 6] (write only)
00190          ResolutionUnit (string) ["DPC", "DPI"]
00191          XResolution, YResolution IM_FLOAT (1)
00192          Interlaced (same as Progressive) IM_INT (1 | 0) default
00193          Gamma IM_FLOAT (1)
00194          WhitePoint IMFLOAT (2)
00195          PrimaryChromaticities IMFLOAT (6)
00196          XPosition, YPosition IM_FLOAT (1)
00197          sRGBIntent IM_INT (1) [0: Perceptual, 1: Relative colorimetric]
00198          TransparencyIndex IM_BYTE (1 or N)
00199          TransparentColor IM_BYTE (3)
00200          CalibrationName, CalibrationUnits (string)
00201          CalibrationLimits IM_INT (2)
00202          CalibrationEquation IM_BYTE (1) [0-Linear,1-Exponential,
00203          CalibrationParam (string) [params separated by '\\n']
00204          Title, Author, Description, Copyright, DateTime (string)
00205          Software, Disclaimer, Warning, Source, Comment, ...
00206          DateTimeModified (string) [when writing uses the current
00207          ICCProfile IM_BYTE (N)
00208          ScaleUnit (string) ["meters", "radians"]
00209          XScale, YScale IM_FLOAT (1)
00210
00211      Comments:
00212          Attributes after the image are ignored.
00213          Define PNG_NO_CONSOLE_IO to avoid printf's.
00214          We define PNG_TIME_RFC1123_SUPPORTED.
00215          Add the following files to the makefile to optimize the
00216          pngvcrd.c - PNG_USE_PNGVCRD
00217                  For Intel x86 CPU and Microsoft Visual C++
00218          pnggccrd.c - PNG_USE_PNGGCCRD
00219                  For Intel x86 CPU (Pentium-MMX or later) and
00220 \endverbatim
00221 * \ingroup format */
00222 void imFormatRegisterPNG(void);
00223
00224 /** \defgroup gif GIF - Graphics Interchange Format
00225 * \section Description
00226 *
00227 * \par
```

```
00228 * Copyright (c) 1987,1988,1989,1990 CompuServe Incorporated.  
00229 * GIF is a Service Mark property of CompuServe Incorporated.  
00230 * Graphics Interchange Format Programming Reference, 1990. \n  
00231 * LZW Copyright Unisys.  
00232 * \par  
00233 * Partial Internal Implementation. \n  
00234 * Decoding and encoding code were extracted from GIFLib 1.0.  
00235 * Copyright (c) 1989 Gershon Elber.  
00236 *  
00237 * \section Features  
00238 *  
00239 \verbatim  
00240     Data Types: Byte  
00241     Color Spaces: MAP only, (Gray and Binary saved as MAP)  
00242     Compressions:  
00243         LZW - Lempel-Ziv & Welch      [default]  
00244     Can have more than one image.  
00245     No alpha channel.  
00246     Internally the lines are arranged from top down to bottom.  
00247     Handle() returns a imBinFile* pointer.  
00248  
00249     Attributes:  
00250         ScreenHeight, ScreenWidth IM_USHORT (1) screen size [def  
00251         Interlaced IM_INT (1 | 0) default 0  
00252         Description (string)  
00253         TransparencyIndex IM_BYTE (1)  
00254         XScreen, YScreen IM_USHORT (1) screen position  
00255         UserInput IM_BYTE (1) [1, 0]  
00256         Disposal (string) [UNDEF, LEAVE, RBACK, RPREV]  
00257         Delay IM_USHORT (1)  
00258         Iterations IM_USHORT (1) (NETSCAPE2.0 Application Extens  
00259  
00260     Comments:  
00261         Attributes after the last image are ignored.  
00262         Reads GIF87 and GIF89, but writes GIF89 always.  
00263         Ignored attributes: Background Color Index, Pixel Aspect  
00264                         Plain Text Extensions, Application E  
00265 \endverbatim  
00266     * \ingroup format */  
00267 void imFormatRegisterGIF(void);  
00268  
00269 /** \defgroup bmp BMP - Windows Device Independent Bitmap  
00270     * \section Description  
00271     *  
00272     * \par  
00273     * Windows Copyright Microsoft Corporation.  
00274     * \par
```

```
00275 * Internal Implementation.  
00276 *  
00277 * \section Features  
00278 *  
00279 \verbatim  
00280     Data Types: Byte  
00281     Color Spaces: RGB, MAP and Binary (Gray saved as MAP)  
00282     Compressions:  
00283         NONE - no compression [default]  
00284         RLE - Run Length Encoding (only for MAP and Gray)  
00285     Only one image.  
00286     Can have an alpha channel (only for RGB)  
00287     Internally the components are always packed.  
00288     Lines arranged from top down to bottom or bottom up to top  
00289     Handle() returns imBinFile* pointer.  
00290  
00291     Attributes:  
00292         ResolutionUnit (string) ["DPC", "DPI"]  
00293         XResolution, YResolution IM_FLOAT (1)  
00294  
00295     Comments:  
00296         Reads OS2 1.x and Windows 3, but writes Windows 3 always  
00297         Version 4 and 5 BMPs are not supported.  
00298 \endverbatim  
00299 * \ingroup format */  
00300 void imFormatRegisterBMP(void);  
00301  
00302 /** \defgroup ras RAS - Sun Raster File  
00303     * \section Description  
00304     *  
00305     * \par  
00306     * Copyright Sun Corporation.  
00307     * \par  
00308     * Internal Implementation.  
00309     *  
00310     * \section Features  
00311     *  
00312 \verbatim  
00313     Data Types: Byte  
00314     Color Spaces: Gray, RGB, MAP and Binary  
00315     Compressions:  
00316         NONE - no compression [default]  
00317         RLE - Run Length Encoding  
00318     Only one image.  
00319     Can have an alpha channel (only for IM_RGB)  
00320     Internally the components are always packed.  
00321     Internally the lines are arranged from top down to bottom.
```

```
00322     Handle() returns imBinFile* pointer.
00323
00324     Attributes:
00325         none
00326 \endverbatim
00327 * \ingroup format */
00328 void imFormatRegisterRAS(void);
00329
00330 /** \defgroup led LED - IUP image in LED
00331 * \section Description
00332 *
00333 * \par
00334 * Copyright Tecgraf/PUC-Rio and PETROBRAS/CENPES.
00335 * \par
00336 * Internal Implementation.
00337 *
00338 * \section Features
00339 *
00340 \verbatim
00341     Data Types: Byte
00342     Color Spaces: MAP only (Gray and Binary saved as MAP)
00343     Compressions:
00344         NONE - no compression [default]
00345     Only one image.
00346     No alpha channel.
00347     Internally the lines are arranged from top down to bottom.
00348     Handle() returns imBinFile* pointer.
00349
00350     Attributes:
00351         none
00352
00353     Comments:
00354         LED file must start with "LEDImage = IMAGE[".
00355 \endverbatim
00356 * \ingroup format */
00357 void imFormatRegisterLED(void);
00358
00359 /** \defgroup sgi SGI - Silicon Graphics Image File Format
00360 * \section Description
00361 *
00362 * \par
00363 * SGI is a trademark of Silicon Graphics, Inc.
00364 * \par
00365 * Internal Implementation.
00366 *
00367 * \section Features
00368 *
```

```
00369 \verbatim
00370     Data Types: Byte and USHORT
00371     Color Spaces: Gray and RGB (Binary saved as Gray, MAP with
00372     Compressions:
00373         NONE - no compression [default]
00374         RLE - Run Length Encoding
00375     Only one image.
00376     Can have an alpha channel (only for IM_RGB)
00377     Internally the components are always packed.
00378     Internally the lines are arranged from bottom up to top.
00379     Handle() returns imBinFile* pointer.
00380
00381     Attributes:
00382         Description (string)
00383 \endverbatim
00384     * \ingroup format */
00385 void imFormatRegisterSGI(void);
00386
00387 /** \defgroup pcx PCX - ZSoft Picture
00388     * \section Description
00389     *
00390     * \par
00391     * Copyright ZSoft Corporation. \n
00392     * ZSoft (1988) PCX Technical Reference Manual.
00393     * \par
00394     * Internal Implementation.
00395     *
00396     * \section Features
00397     *
00398 \verbatim
00399     Data Types: Byte
00400     Color Spaces: RGB, MAP and Binary (Gray saved as MAP)
00401     Compressions:
00402         NONE - no compression
00403         RLE - Run Length Encoding [default - since uncompressed]
00404     Only one image.
00405     No alpha channel.
00406     Internally the components are always packed.
00407     Internally the lines are arranged from top down to bottom.
00408     Handle() returns imBinFile* pointer.
00409
00410     Attributes:
00411         ResolutionUnit (string) ["DPC", "DPI"]
00412         XResolution, YResolution IM_FLOAT (1)
00413         XScreen, YScreen IM_USHORT (1) screen position
00414
00415     Comments:
```

```
00416      Reads Versions 0-5, but writes Version 5 always.
00417 \endverbatim
00418 * \ingroup format */
00419 void imFormatRegisterPCX(void);
00420
00421 /** \defgroup tga TGA - Truevision Graphics Adapter File
00422 * \section Description
00423 *
00424 * \par
00425 * Truevision TGA File Format Specification Version 2.0 \n
00426 * Technical Manual Version 2.2 January, 1991 \n
00427 * Copyright 1989, 1990, 1991 Truevision, Inc.
00428 * \par
00429 * Internal Implementation.
00430 *
00431 * \section Features
00432 *
00433 \verbatim
00434     Supports 8 bits per component only. Data type is always By
00435     Color Spaces: Gray, RGB and MAP (Binary saved as Gray)
00436     Compressions:
00437         NONE - no compression [default]
00438         RLE - Run Length Encoding
00439     Only one image.
00440     No alpha channel.
00441     Internally the components are always packed.
00442     Internally the lines are arranged from bottom up to top or
00443     Handle() returns imBinFile* pointer.
00444
00445     Attributes:
00446         XScreen, YScreen IM_USHORT (1) screen position
00447         Title, Author, Description, JobName, Software (string)
00448         SoftwareVersion (read only) (string)
00449         DateTimeModified (string) [when writing uses the current
00450         Gamma IM_FLOAT (1)
00451 \endverbatim
00452 * \ingroup format */
00453 void imFormatRegisterTGA(void);
00454
00455 /** \defgroup pnm PNM - Netpbm Portable Image Map
00456 * \section Description
00457 *
00458 * \par
00459 * PNM formats Copyright Jef Poskanzer
00460 * \par
00461 * Internal Implementation.
00462 *
```

```
00463 * \section Features
00464 *
00465 \verbatim
00466     Data Types: Byte and UShort
00467     Color Spaces: Gray, RGB and Binary
00468     Compressions:
00469         NONE - no compression [default]
00470         ASCII (textual data)
00471     Can have more than one image, but sequential access only.
00472     No alpha channel.
00473     Internally the components are always packed.
00474     Internally the lines are arranged from top down to bottom.
00475     Handle() returns imBinFile* pointer.
00476
00477     Attributes:
00478         Description (string)
00479
00480     Comments:
00481         In fact ASCII is an expansion...
00482 \endverbatim
00483 * \ingroup format */
00484 void imFormatRegisterPNM(void);
00485
00486 /** \defgroup ico ICO - Windows Icon
00487 * \section Description
00488 *
00489 * \par
00490 * Windows Copyright Microsoft Corporation.
00491 * \par
00492 * Internal Implementation.
00493 *
00494 * \section Features
00495 *
00496 \verbatim
00497     Data Types: Byte
00498     Color Spaces: RGB, MAP and Binary (Gray saved as MAP)
00499     Compressions:
00500         NONE - no compression [default]
00501     Can have more than one image. But writing is limited to 5
00502         and all images must have different sizes and bpp.
00503     No alpha channel.
00504     Internally the components are always packed.
00505     Internally the lines are arranged from bottom up to top.
00506     Handle() returns imBinFile* pointer.
00507
00508     Attributes:
00509         none
```

```
00510
00511     Comments:
00512         If the user specifies an alpha channel, the AND mask is
00513             but the file color mode never contains the IM_ALPHA fl
00514             Although any size and bpp can be used is recomended to u
00515                 16x16, 32x32, 48x48, 64x64 or 96x96
00516                 2 colors, 16 colors or 256 colors
00517 \endverbatim
00518 * \ingroup format */
00519 void imFormatRegisterICO(void);
00520
00521 /** \defgroup krn KRN - IM Kernel File Format
00522 * \section Description
00523 *
00524 * \par
00525 * Textual format to provied a simple way to create kernel con
00526 * \par
00527 * Internal Implementation.
00528 *
00529 * \section Features
00530 *
00531 \verbatim
00532     Data Types: Byte, Int
00533     Color Spaces: Gray
00534     Compressions:
00535         NONE - no compression [default]
00536     Only one image.
00537     No alpha channel.
00538     Internally the lines are arranged from top down to bottom.
00539     Handle() returns imBinFile* pointer.
00540
00541     Attributes:
00542         Description (string)
00543
00544     Comments:
00545         The format is very simple, inspired by PNM.
00546         It was developed because PNM does not have support for I
00547         Remeber that usually convolution operations use kernel s
00548
00549     Format Model:
00550         IMKERNEL
00551         Description up to 512 characters
00552         width height
00553         type (0 - IM_INT, 1 - IM_FLOAT)
00554         data...
00555
00556     Example:
```

```
00557     IMKERNEL
00558     Gradian
00559     3 3
00560     0
00561     0 -1 0
00562     0 1 0
00563     0 0 0
00564 \endverbatim
00565 * \ingroup format */
00566 void imFormatRegisterKRN(void);
00567
00568
00569 #if defined(__cplusplus)
00570 }
00571 #endif
00572
00573 #endif
```

include

im_format_avi.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Register the AVI Format
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_FORMAT_AVI_H
00009 #define __IM_FORMAT_AVI_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015 /** \defgroup avi AVI - Windows Audio-Video Interleaved RIFF
00016 * \section Description
00017 *
00018 * \par
00019 * Windows Copyright Microsoft Corporation.
00020 * \par
00021 * Internal Implementation, Windows Only. \n
00022 * You must link the application with "im_avi.lib"
00023 * and you must call the function \ref imFormatRegisterAVI once
00024 * to register the format into the IM core library. \n
00025 * Depends also on the VFW library (vfw32.lib).
00026 * When using the "im_avi.dll" this extra library is not necessary.
00027 * If using Cygwin or MingW must link with "vfw_ms32.a" and "vfw32.lib".
00028 * \par
00029 * See \ref im_format_avi.h
00030 *
00031 * \section Features
00032 *
00033 \verbatim
00034     Data Types: Byte
00035     Color Spaces: RGB, MAP and Binary (Gray saved as MAP)
00036     Compressions (installed in Windows XP by default):
00037         NONE      - no compression [default]
00038         RLE       - Microsoft RLE (8bpp only)
00039         CINEPACK - Cinepak Codec by Radius
```

```

00040      MSVC      - Microsoft Video 1 (old)
00041      M261      - Microsoft H.261 Video Codec
00042      M263      - Microsoft H.263 Video Codec
00043      I420      - Intel 4:2:0 Video Codec (same as M263)
00044      IV32      - Intel Indeo Video Codec 3.2 (old)
00045      IV41      - Intel Indeo Video Codec 4.5 (old)
00046      IV50      - Intel Indeo Video 5.1
00047      IYUV      - Intel IYUV Codec
00048      MPG4      - Microsoft MPEG-4 Video Codec V1 (not MPEG-4 c
00049      MP42      - Microsoft MPEG-4 Video Codec V2 (not MPEG-4 c
00050      CUSTOM    - (show compression dialog)
00051      DIVX      - DivX 5.0.4 Codec (DivX must be installed)
00052          (others, must be the 4 charaters of the fourfcc code)
00053      Can have more than one image.
00054      Can have an alpha channel (only for RGB)
00055      Internally the components are always packed.
00056      Lines arranged from top down to bottom or bottom up to top
00057      Handle() returns PAVIFILE.

00058
00059      Attributes:
00060          FPS IM_FLOAT (1) (should set when writing, default 15)
00061          AVIQuality IM_INT (1) [1-10000, default -1] (write only)
00062          KeyFrameRate IM_INT (1) (write only) [key frame frequenc
00063          DataRate IM_INT (1) (write only) [kilobits/second, defau
00064
00065      Comments:
00066          Reads only the first video stream. Other streams are ign
00067          All the images have the same size, you must call imFileR
00068          at least once.
00069          For codecs comparsion and download go to:
00070              http://graphics.lcs.mit.edu/~tbuehler/video/codecs/
00071              http://www.fourcc.org
00072 \endverbatim
00073 * \ingroup format */
00074
00075 /** Register the AVI Format
00076 * \ingroup avi */
00077 void imFormatRegisterAVI(void);
00078
00079 #if defined(__cplusplus)
00080 }
00081 #endif
00082
00083 #endif

```

include

im_format_jp2.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Register the JP2 Format
00003  *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_FORMAT_JP2_H
00009 #define __IM_FORMAT_JP2_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup jp2 JP2 - JPEG-2000 JP2 File Format
00017  * \section Description
00018  *
00019  * \par
00020  * ISO/IEC 15444 (2000, 2003)\n
00021  * http://www.jpeg.org/
00022  * \par
00023  * You must link the application with "im_jp2.lib"
00024  * and you must call the function \ref imFormatRegisterJP2 once
00025  * to register the format into the IM core library. \n
00026  * \par
00027  * Access to the JPEG2000 file format uses libJasper version 1
00028  * http://www.ece.uvic.ca/~mdadams/jasper
00029  * Copyright (c) 2001-2003 Michael David Adams.
00030  * \par
00031  * See \ref im_format_jp2.h
00032  *
00033  * \section Features
00034  *
00035 \verbatim
00036     Data Types: Byte and UShort
00037     Color Spaces: Binary, Gray, RGB, YCbCr, Lab and XYZ
00038     Compressions:
00039         JPEG-2000 - ISO JPEG 2000 [default]
```

```
00040 Only one image.
00041 Can have an alpha channel.
00042 Internally the components are always unpacked.
00043 Internally the lines are arranged from top down to bottom.
00044 Handle() returns jas_image_t*
00045
00046 Attributes:
00047     CompressionRatio IM_FLOAT (1) [write only, example: Rati
00048
00049 Comments:
00050     We read code stream syntax and JP2, but write as JP2 alw
00051     Used definitions EXCLUDE_JPG_SUPPORT,EXCLUDE_MIF_SUPPORT
00052                 EXCLUDE_PNM_SUPPORT,EXCLUDE_RAS_SUPPORT
00053                 EXCLUDE_BMP_SUPPORT,EXCLUDE_PGX_SUPPORT
00054     Changed jas_config.h to match our needs.
00055     New file jas_binfile.c
00056     Changed jas_stream.c to export 2 functions.
00057     Changed jp2_dec.c and jpc_cs.c to remove unit and ulong
00058 \endverbatim
00059 * \ingroup format */
00060
00061 /** Register the JP2 Format
00062 * \ingroup jp2 */
00063 void imFormatRegisterJP2(void);
00064
00065
00066 #if defined(__cplusplus)
00067 }
00068 #endif
00069
00070 #endif
```

include

im_format_raw.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Initialize the RAW Format Driver
00003  * Header for internal use only.
00004  *
00005  * See Copyright Notice in im_lib.h
00006  * $Id: Exp $
00007 */
00008
00009 #ifndef __IM_FORMAT_RAW_H
00010 #define __IM_FORMAT_RAW_H
00011
00012 #if defined(__cplusplus)
00013 extern "C" {
00014 #endif
00015
00016 /** \defgroup raw RAW - RAW File
00017  *
00018  * \par
00019  * The file must be open/created with the functions \ref imFil
00020  *
00021  * \section Description
00022  *
00023  * \par
00024  * Internal Implementation.
00025  * \par
00026  * Supports RAW binary images. You must know image parameters
00027  * You must set the IM_INT attributes "Width", "Height", "Colo
00028  * \par
00029  * The data must be in binary form, but can start in an arbitr
00030  * The default is at 0 offset.
00031  * \par
00032  * Integer sign and double precision can be converted using at
00033  * The conversions will be BYTE<->CHAR, USHORT<->SHORT, INT<->
00034  * \par
00035  * Byte Order can be Little Endian (Intel=1) or Big Endian (Mo
00036  * \par
00037  * The lines can be aligned to a BYTE (1), WORD (2) or DWORD (
00038  * \par
00039  * See \ref im_raw.h
```

```
00040  *
00041  * \section Features
00042  *
00043 \verbatim
00044     Data Types: <all>
00045     Color Spaces: all, except MAP.
00046     Compressions:
00047         NONE - no compression
00048     Can have more than one image, depends on "StartOffset" att
00049     Can have an alpha channel.
00050     Components can be packed or not.
00051     Lines arranged from top down to bottom or bottom up to top
00052     Handle() returns a imBinFile* pointer.
00053
00054     Attributes:
00055         Width, Height, ColorMode, DataType IM_INT (1)
00056         StartOffset, SwitchType, ByteOrder, Padding IM_INT (1)
00057 \endverbatim
00058 * \ingroup format */
00059 imFormat* imFormatInitRAW(void);
00060
00061
00062 #if defined(__cplusplus)
00063 }
00064 #endif
00065
00066 #endif
```

include

im_format_wmv.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Register the WMF Format
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_FORMAT_WMV_H
00009 #define __IM_FORMAT_WMV_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015 /** \defgroup wmv WMV - Windows Media Video Format
00016 * \section Description
00017 *
00018 * \par
00019 * Advanced Systems Format (ASF) \n
00020 * Windows Copyright Microsoft Corporation.
00021 * \par
00022 * Internal Implementation, Windows Only. \n
00023 * You must link the application with "im_wmv.lib"
00024 * and you must call the function \ref imFormatRegisterWMV once
00025 * to register the format into the IM core library. \n
00026 * Depends also on the WMF SDK (wmvcore.lib).
00027 * When using the "im_wmv.dll" this extra library is not necessary.
00028 * \par
00029 * The application users should have the WMV codec 9 installed
00030 * http://www.microsoft.com/windows/windowsmedia/format/codecs/
00031 * \par
00032 * You must agree with the WMF SDK EULA to use the SDK. \n
00033 * http://wmlicense.smdisp.net/v9sdk/
00034 * \par
00035 * For more information: \n
00036 * http://www.microsoft.com/windows/windowsmedia/9series/sdk.aspx
00037 * http://msdn.microsoft.com/library/en-us/wmform/htm/introduction.asp
00038 * \par
00039 * See \ref im_format_wmv.h
```

```
00040  *
00041  * \section Features
00042  *
00043 \verbatim
00044     Data Types: Byte
00045     Color Spaces: RGB and MAP (Gray and Binary saved as MAP)
00046     Compressions (installed in Windows XP by default):
00047         NONE      - no compression
00048         MPEG-4v3   - Windows Media MPEG-4 Video V3
00049         MPEG-4v1   - ISO MPEG-4 Video V1
00050         WMV7       - Windows Media Video V7
00051         WMV7Screen - Windows Media Screen V7
00052         WMV8       - Windows Media Video V8
00053         WMV9Screen - Windows Media Video 9 Screen
00054         WMV9       - Windows Media Video 9 [default]
00055         Unknown    - Others
00056     Can have more than one image.
00057     Can have an alpha channel (only for RGB) ?
00058     Internally the components are always packed.
00059     Lines arranged from top down to bottom or bottom up to top
00060     Handle() returns IWMSyncReader* when reading, IWMWriter* w
00061
00062     Attributes:
00063         FPS IM_FLOAT (1) (should set when writing, default 15)
00064         WMFQuality IM_INT (1) [0-100, default 50] (write only)
00065         MaxKeyFrameTime IM_INT (1) (write only) [maximum key fra
00066         DataRate IM_INT (1) (write only) [kilobits/second, defau
00067         VBR IM_INT (1) [0, 1] (write only) [0 - Constant Bit Rat
00068         (and several others from the file-level attributes) For
00069             Title, Author, Copyright, Description (string)
00070             Duration IM_INT [100-nanosecond units]
00071             Seekable, HasAudio, HasVideo, Is_Protected, Is_Trusted
00072             NumberOfFrames IM_INT (1)
00073
00074     Comments:
00075         IMPORTANT - The "image_count" and the "FPS" attribute ma
00076             we try to estimate from the duration and from the aver
00077             We do not handle DRM protected files (Digital Rights Man
00078             Reads only the first video stream. Other streams are ign
00079             All the images have the same size, you must call imFileR
00080                 at least once.
00081             For optimal random reading, the file should be indexed p
00082                 If not indexed by frame, random positioning may not be p
00083                 Sequencial reading will always be precise.
00084             When writing we use a custom profile and time indexing o
00085             We do not support multipass encoding.
00086             Since the driver uses COM, CoInitialize(NULL) and CoUnin
```

```
00087 \endverbatim
00088 * \ingroup format */
00089
00090 /** Register the WMF Format
00091 * \ingroup wmv */
00092 void imFormatRegisterWMV(void);
00093
00094
00095 #if defined(__cplusplus)
00096 }
00097 #endif
00098
00099 #endif
```

include

im_lib.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Library Management and Main Documentation
00003  *
00004  * See Copyright Notice in this file.
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_LIB_H
00009 #define __IM_LIB_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup lib Library Management
00017  * \par
00018  * Usefull definitions for about dialogs and
00019  * for checking the compiled version with the linked version f
00020  * \par
00021  * See \ref im_lib.h
00022  * @{
00023 */
00024 #define IM_AUTHOR "Antonio Scuri"
00025 #define IM_COPYRIGHT "Copyright (C) 1994-2004 Tecgraf/PUC-Rio
00026 #define IM_VERSION "3.0.3"
00027 #define IM_VERSION_DATE "2004/10/14"
00028 /** @} */
00029
00030 /** Library release number used in the compilation time. \n
00031  * You can compare this with the value returned by \ref imVers
00032  * \ingroup lib */
00033 #define IM_VERSION_NUMBER 300003
00034
00035 /** Returns the library current version.
00036  * \ingroup lib */
00037 const char* imVersion(void);
00038
00039 /** Returns the library current version release date.
```

```
00040 * \ingroup lib */
00041 const char* imVersionDate(void);
00042
00043 /** Returns the library current version number.
00044 * \ingroup lib */
00045 int imVersionNumber(void);
00046
00047
00048 #if defined(__cplusplus)
00049 }
00050 #endif
00051
00052
00053 /*! \mainpage IM
00054 * <CENTER>
00055 * <H3> Image Representation, Storage, Capture and Processing
00056 * Tecgraf: Computer Graphics Technology Group, PUC-Rio, Brazil
00057 * http://www.tecgraf.puc-rio.br/im \n
00058 * mailto:im@tecgraf.puc-rio.br
00059 * </CENTER>
00060 *
00061 * \section over Overview
00062 * \par
00063 * IM is a toolkit for Digital Imaging.
00064 * \par
00065 * It provides support for image capture, several image file f
00066 * \par
00067 * Image representation includes scientific data types (like I
00068 * and attributes (or metadata like GeoTIFF and Exif tags).
00069 * Animation, video and volumes are supported as image sequenc
00070 * but there is no digital audio support.
00071 * \par
00072 * The main goal of the library is to provide a simple API and
00073 * of images for scientific applications.
00074 * \par
00075 * The toolkit API is written in C.
00076 * The core library source code is implemented in C++ and it i
00077 * it can be compiled in Windows and UNIX with no modification
00078 * New image processing operations can be implemented in C or
00079 * \par
00080 * IM is free software, can be used for public and commercial
00081 * \par
00082 * This work was developed at Tecgraf/PUC-Rio
00083 * by means of the partnership with PETROBRAS/CENPES.
00084 *
00085 * \section author Author
00086 * \par
```

```
00087 * Basic Software Group @ Tecgraf/PUC-Rio
00088 * - Antonio Scuri scuri@tecgraf.puc-rio.br
00089 *
00090 * \section copyright Copyright Notice
00091 \verbatim
00092 ****
00093 ****
00094 * Copyright (C) 1994-2004 Tecgraf/PUC-Rio and PETROBRAS S/A.
00095 * All rights reserved.
00096 *
00097 * Permission is hereby granted, free of charge, to any person
00098 * a copy of this software and associated documentation files (
00099 * "Software"), to deal in the Software without restriction, in
00100 * without limitation the rights to use, copy, modify, merge, p
00101 * distribute, sublicense, and/or sell copies of the Software,
00102 * permit persons to whom the Software is furnished to do so, s
00103 * the following conditions:
00104 *
00105 * The above copyright notice and this permission notice shall
00106 * included in all copies or substantial portions of the Softwa
00107 *
00108 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KI
00109 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANT
00110 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINF
00111 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
00112 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF C
00113 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WIT
00114 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00115 ****
00116 \endverbatim
00117 */
00118
00119
00120 /** \defgroup imagerep Image Representation
00121 * \par
00122 * See \ref im.h
00123 */
00124
00125
00126 /** \defgroup file Image Storage
00127 * \par
00128 * See \ref im.h
00129 */
00130
00131
00132 /** \defgroup fileread Read Access
00133 * \par
```

```
00134 * See \ref im.h
00135 * \ingroup file */
00136
00137
00138 /** \defgroup filewrite Write Access
00139 * \par
00140 * See \ref im.h
00141 * \ingroup file */
00142
00143
00144 /** \defgroup format File Formats
00145 * \par
00146 * See \ref im.h
00147 *
00148 * Internal Predefined File Formats:
00149 * \li "BMP" - Windows Device Independent Bitmap
00150 * \li "PCX" - ZSoft Picture
00151 * \li "GIF" - Graphics Interchange Format
00152 * \li "TIFF" - Tagged Image File Format
00153 * \li "RAS" - Sun Raster File
00154 * \li "SGI" - Silicon Graphics Image File Format
00155 * \li "JPEG" - JPEG File Interchange Format
00156 * \li "LED" - IUP image in LED
00157 * \li "TGA" - Truevision Targa
00158 * \li "RAW" - RAW File
00159 * \li "PNM" - Netpbm Portable Image Map
00160 * \li "ICO" - Windows Icon
00161 * \li "PNG" - Portable Network Graphic Format
00162 *
00163 * Other Supported File Formats:
00164 * \li "JP2" - JPEG-2000 JP2 File Format
00165 * \li "AVI" - Windows Audio-Video Interleaved RIFF
00166 * \li "WMV" - Windows Media Video Format
00167 *
00168 * Some Known Compressions:
00169 * \li "NONE" - No Compression.
00170 * \li "RLE" - Run Length Encoding.
00171 * \li "LZW" - Lempel, Ziff and Welsh.
00172 * \li "JPEG" - Joint Photographic Experts Group.
00173 * \li "DEFLATE" - LZ77 variation (ZIP)
00174 *
00175 * \ingroup file */
00176
00177
00178 /* Library Names Convention
00179 *
00180 * Global Functions and Types - "im[Object][Action]" using
```

```
00181 * Local Functions and Types - "i[Object][Action]" using
00182 * Local Static Variables - same as local functions and type
00183 * Local Static Tables - same as local functions and types w
00184 * Variables and Members - no prefix, all lower case (width)
00185 * Defines and Enumerations - all capitals (IM_ERR_NONE)
00186 *
00187 */
00188
00189
00190 #endif
```

include

im_math.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Math Utilities
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_MATH_H
00009 #define __IM_MATH_H
00010
00011 #include <math.h>
00012 #include "im_util.h"
00013
00014 #ifdef IM_DEFMATHFLOAT
00015 inline float acosf(float _X) {return ((float)acos((double)_X))}
00016 inline float asinf(float _X) {return ((float)asin((double)_X))}
00017 inline float atanf(float _X) {return ((float)atan((double)_X))}
00018 inline float atan2f(float _X, float _Y) {return ((float)atan2(
00019 inline float ceilf(float _X) {return ((float)ceil((double)_X))
00020 inline float cosf(float _X) {return ((float)cos((double)_X));
00021 inline float coshf(float _X) {return ((float)cosh((double)_X))
00022 inline float expf(float _X) {return ((float)exp((double)_X));
00023 inline float fabsf(float _X) {return ((float)fabs((double)_X))
00024 inline float floorf(float _X) {return ((float)floor((double)_X
00025 inline float fmodf(float _X, float _Y) {return ((float)fmod((d
00026 inline float logf(float _X) {return ((float)log((double)_X));
00027 inline float log10f(float _X) {return ((float)log10((double)_X
00028 inline float powf(float _X, float _Y) {return ((float)pow((dou
00029 inline float sinf(float _X) {return ((float)sin((double)_X));
00030 inline float sinhf(float _X) {return ((float)sinh((double)_X))
00031 inline float sqrtf(float _X) {return ((float)sqrt((double)_X))
00032 inline float tanf(float _X) {return ((float)tan((double)_X));
00033 inline float tanhf(float _X) {return ((float)tanh((double)_X))
00034 #endif
00035
00036 /** \defgroup math Math Utilities
00037 * \par
00038 * See \ref im_color.h
00039 * \ingroup util */
```

```

00040
00041
00042 /** Does Zero Order Decimation (Mean).
00043   * \ingroup math */
00044 template <class T, class TU>
00045 inline T imZeroOrderDecimation(int width, int height, T *map,
00046 {
00047     int x0,x1,y0,y1;
00048     (void)Dummy;
00049
00050     x0 = (int)floor(xl - box_width/2.0 - 0.5) + 1;
00051     y0 = (int)floor(yl - box_height/2.0 - 0.5) + 1;
00052     x1 = (int)floor(xl + box_width/2.0 - 0.5);
00053     y1 = (int)floor(yl + box_height/2.0 - 0.5);
00054
00055     if (x0 == x1) x1++;
00056     if (y0 == y1) y1++;
00057
00058     x0 = x0<0? 0: x0>width-1? width-1: x0;
00059     y0 = y0<0? 0: y0>height-1? height-1: y0;
00060     x1 = x1<0? 0: x1>width-1? width-1: x1;
00061     y1 = y1<0? 0: y1>height-1? height-1: y1;
00062
00063     TU Value;
00064     int Count = 0;
00065
00066     Value = 0;
00067
00068     for (int y = y0; y <= y1; y++)
00069     {
00070         for (int x = x0; x <= x1; x++)
00071         {
00072             Value += map[y*width+x];
00073             Count++;
00074         }
00075     }
00076
00077     if (Count == 0)
00078     {
00079         Value = 0;
00080         return (T)Value;
00081     }
00082
00083     return (T)(Value/(float)Count);
00084 }
00085
00086 /** Does Bilinear Decimation.

```

```
00087 * \ingroup math */
00088 template <class T, class TU>
00089 inline T imBilinearDecimation(int width, int height, T *map, f
00090 {
00091     int x0,x1,y0,y1;
00092     (void)Dummy;
00093
00094     x0 = (int)floor(xl - box_width/2.0 - 0.5) + 1;
00095     y0 = (int)floor(yl - box_height/2.0 - 0.5) + 1;
00096     x1 = (int)floor(xl + box_width/2.0 - 0.5);
00097     y1 = (int)floor(yl + box_height/2.0 - 0.5);
00098
00099     if (x0 == x1) x1++;
00100     if (y0 == y1) y1++;
00101
00102     x0 = x0<0? 0: x0>width-1? width-1: x0;
00103     y0 = y0<0? 0: y0>height-1? height-1: y0;
00104     x1 = x1<0? 0: x1>width-1? width-1: x1;
00105     y1 = y1<0? 0: y1>height-1? height-1: y1;
00106
00107     TU Value, LineValue;
00108     float LineNorm, Norm, dxr, dyr;
00109
00110     Value = 0;
00111     Norm = 0;
00112
00113     for (int y = y0; y <= y1; y++)
00114     {
00115         dyr = yl - (y+0.5f);
00116         if (dyr < 0) dyr *= -1;
00117
00118         LineValue = 0;
00119         LineNorm = 0;
00120
00121         for (int x = x0; x <= x1; x++)
00122         {
00123             dxr = xl - (x+0.5f);
00124             if (dxr < 0) dxr *= -1;
00125
00126             LineValue += map[y*width+x] * dxr;
00127             LineNorm += dxr;
00128         }
00129
00130         Value += LineValue * dyr;
00131         Norm += dyr * LineNorm;
00132     }
00133 }
```

```

00134     if (Norm == 0)
00135     {
00136         Value = 0;
00137         return (T)Value;
00138     }
00139
00140     return (T)(Value/Norm);
00141 }
00142
00143 /** Does Zero Order Interpolation (Nearest Neighborhood).
00144 * \ingroup math */
00145 template <class T>
00146 inline T imZeroOrderInterpolation(int width, int height, T *map
00147 {
00148     int x0 = (int)(xl-0.5f);
00149     int y0 = (int)(yl-0.5f);
00150     x0 = x0<0? 0: x0>width-1? width-1: x0;
00151     y0 = y0<0? 0: y0>height-1? height-1: y0;
00152     return map[y0*width + x0];
00153 }
00154
00155 /** Does Bilinear Interpolation.
00156 * \ingroup math */
00157 template <class T>
00158 inline T imBilinearInterpolation(int width, int height, T *map
00159 {
00160     int x0 = (int)(xl-0.5f);
00161     int y0 = (int)(yl-0.5f);
00162     int x1 = x0+1;
00163     int y1 = y0+1;
00164
00165     float t = xl - (x0+0.5f);
00166     float u = yl - (y0+0.5f);
00167
00168     x0 = x0<0? 0: x0>width-1? width-1: x0;
00169     y0 = y0<0? 0: y0>height-1? height-1: y0;
00170     x1 = x1<0? 0: x1>width-1? width-1: x1;
00171     y1 = y1<0? 0: y1>height-1? height-1: y1;
00172
00173     T fll = map[y0*width + x0];
00174     T fhl = map[y0*width + x1];
00175     T flh = map[y1*width + x0];
00176     T fhh = map[y1*width + x1];
00177
00178     return (T)((fhh - flh - fhl + fll) * u * t +
00179                 (fhl - fll) * t +
00180                 (flh - fll) * u +

```

```

00181                               f11);
00182 }
00183
00184 /** Does Bicubic Interpolation.
00185  * \ingroup math */
00186 template <class T, class TU>
00187 inline T imBicubicInterpolation(int width, int height, T *map,
00188 {
00189     (void)Dummy;
00190
00191     int x0 = (int)(xl-0.5f);
00192     int y0 = (int)(yl-0.5f);
00193     int x1 = x0-1;
00194     int x2 = x0+2;
00195     int y1 = y0-1;
00196     int y2 = y0+2;
00197
00198     float t = xl - (x0+0.5f);
00199     float u = yl - (y0+0.5f);
00200
00201     x1 = x1<0? 0: x1>width-1? width-1: x1;
00202     y1 = y1<0? 0: y1>height-1? height-1: y1;
00203     x2 = x2<0? 0: x2>width-1? width-1: x2;
00204     y2 = y2<0? 0: y2>height-1? height-1: y2;
00205
00206     float CX[4], CY[4];
00207
00208     // Optimize calculations
00209     {
00210         float x, x2, x3;
00211
00212 #define C0 (-x3 + 2.0f*x2 - x)
00213 #define C1 ( x3 - 2.0f*x2 + 1.0f)
00214 #define C2 (-x3 + x2 + x)
00215 #define C3 ( x3 - x2)
00216
00217         x = t;
00218         x2 = x*x; x3 = x2*x;
00219         CX[0] = C0; CX[1] = C1; CX[2] = C2; CX[3] = C3;
00220
00221         x = u;
00222         x2 = x*x; x3 = x2*x;
00223         CY[0] = C0; CY[1] = C1; CY[2] = C2; CY[3] = C3;
00224     }
00225
00226 #undef C0
00227 #undef C1

```

```

00228 #undef C2
00229 #undef C3
00230
00231     TU LineValue, Value;
00232     float LineNorm, Norm;
00233
00234     Value = 0;
00235     Norm = 0;
00236
00237     for (int y = y1; y <= y2; y++)
00238     {
00239         LineValue = 0;
00240         LineNorm = 0;
00241
00242         for (int x = x1; x <= x2; x++)
00243         {
00244             LineValue += map[y*width+x] * CX[x-x1];
00245             LineNorm += CX[x-x1];
00246         }
00247
00248         Value += LineValue * CY[y-y1];
00249         Norm += CY[y-y1] * LineNorm;
00250     }
00251
00252     if (Norm == 0)
00253     {
00254         Value = 0;
00255         return (T)Value;
00256     }
00257
00258     Value = (Value/Norm);
00259
00260     int size = sizeof(T);
00261     if (size == 1)
00262         return (T)(Value<=0? 0: Value<=255? Value: 255);
00263     else
00264         return (T)(Value);
00265 }
00266
00267 /** Calculates minimum and maximum values.
00268  * \ingroup math */
00269 template <class T>
00270 inline void imMinMax(const T *map, int count, T& min, T& max)
00271 {
00272     min = *map++;
00273     max = min;
00274     for (int i = 1; i < count; i++)

```

```
00275  {
00276      T value = *map++;
00277
00278      if (value > max)
00279          max = value;
00280      else if (value < min)
00281          min = value;
00282  }
00283 }
00284
00285 #endif
```

include

im_math_op.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Math Operations
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_MATH_OP_H
00009 #define __IM_MATH_OP_H
00010
00011 #include "im_complex.h"
00012
00013 // #define IM_NEARZERO 0.0000001f
00014 // #define IM_NEARINF 10000000
00015
00016 /// Crop value to Byte limit
00017 template <class T>
00018 inline T crop_byte(const T& v)
00019 {
00020     return v <= 0? 0: v <= 255? v: 255;
00021 }
00022
00023 /// Generic Addition with 2 template types
00024 template <class T1, class T2>
00025 inline T1 add_op(const T1& v1, const T2& v2)
00026 {
00027     return v2 + v1;
00028 }
00029
00030 /// Generic Subtraction with 2 template types
00031 template <class T1, class T2>
00032 inline T1 sub_op(const T1& v1, const T2& v2)
00033 {
00034     return v2 - v1;
00035 }
00036
00037 /// Generic Multiplication with 2 template types
00038 template <class T1, class T2>
00039 inline T1 mul_op(const T1& v1, const T2& v2)
```

```
00040 {
00041     return v2 * v1;
00042 }
00043
00044 // Generic Division with 2 template types
00045 template <class T1, class T2>
00046 inline T1 div_op(const T1& v1, const T2& v2)
00047 {
00048 //  if (v2 == 0) return (T1)IM_NEARINF;
00049     return v1 / v2;
00050 }
00051
00052 // Generic Invert
00053 template <class T>
00054 inline T inv_op(const T& v)
00055 {
00056 //  if (v == 0) return (T)IM_NEARINF;
00057     return 1/v;
00058 }
00059
00060 // Generic Difference with 2 template types
00061 template <class T1, class T2>
00062 inline T1 diff_op(const T1& v1, const T2& v2)
00063 {
00064     if (v1 <= v2)
00065         return v2 - v1;
00066     return v1 - v2;
00067 }
00068
00069 // Generic Minimum with 2 template types
00070 template <class T1, class T2>
00071 inline T1 min_op(const T1& v1, const T2& v2)
00072 {
00073     if (v1 <= v2)
00074         return v1;
00075     return v2;
00076 }
00077
00078 // Generic Maximum with 2 template types
00079 template <class T1, class T2>
00080 inline T1 max_op(const T1& v1, const T2& v2)
00081 {
00082     if (v1 <= v2)
00083         return v2;
00084     return v1;
00085 }
00086
```

```
00087 // Generic Power with 2 template types
00088 template <class T1, class T2>
00089 inline T1 pow_op(const T1& v1, const T2& v2)
00090 {
00091     return (T1)pow(v1, v2);
00092 }
00093
00094 // Generic Absbsolute
00095 template <class T>
00096 inline T abs_op(const T& v)
00097 {
00098     if (v <= 0)
00099         return -1*v;
00100     return v;
00101 }
00102
00103 // Generic Less
00104 template <class T>
00105 inline T less_op(const T& v)
00106 {
00107     return -1*v;
00108 }
00109
00110 // Generic Square
00111 template <class T>
00112 inline T sqr_op(const T& v)
00113 {
00114     return v*v;
00115 }
00116
00117 inline int sqrt(const int& C)
00118 {
00119     return (int)sqrt(float(C));
00120 }
00121
00122 // Generic Square Root
00123 template <class T>
00124 inline T sqrt_op(const T& v)
00125 {
00126     return (T)sqrt(v);
00127 }
00128
00129 inline int exp(const int& v)
00130 {
00131     return (int)exp((float)v);
00132 }
00133
```

```
00134 /// Generic Exponential
00135 template <class T>
00136 inline T exp_op(const T& v)
00137 {
00138     return (T)exp(v);
00139 }
00140
00141 inline int log(const int& v)
00142 {
00143     return (int)log((float)v);
00144 }
00145
00146 /// Generic Logarithm
00147 template <class T>
00148 inline T log_op(const T& v)
00149 {
00150 //  if (v <= 0) return (T)IM_NEARINF;
00151     return (T)log(v);
00152 }
00153
00154 // Dummy sin
00155 inline imcfloat sin(const imcfloat& v)
00156 {
00157     return (v);
00158 }
00159
00160 inline int sin(const int& v)
00161 {
00162     return (int)sin((float)v);
00163 }
00164
00165 /// Generic Sine
00166 template <class T>
00167 inline T sin_op(const T& v)
00168 {
00169     return (T)sin(v);
00170 }
00171
00172 inline int cos(const int& v)
00173 {
00174     return (int)cos((float)v);
00175 }
00176
00177 // Dummy cos
00178 inline imcfloat cos(const imcfloat& v)
00179 {
00180     return (v);
```

```
00181 }
00182
00183 // Generic Cosine
00184 template <class T>
00185 inline T cos_op(const T& v)
00186 {
00187     return (T)cos(v);
00188 }
00189
00190 // Sets a bit in an array
00191 inline void imDataBitSet(imbyte* data, int index, int bit)
00192 {
00193     if (bit)
00194         data[index / 8] |= (0x01 << (7 - (index % 8)));
00195     else
00196         data[index / 8] &= ~(0x01 << (7 - (index % 8)));
00197 }
00198
00199 // Gets a bit from an array
00200 inline int imDataBitGet(imbyte* data, int index)
00201 {
00202     return (data[index / 8] >> (7 - (index % 8))) & 0x01;
00203 }
00204
00205 #endif
```

include

im_palette.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Palette Generators
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_PALETTE_H
00009 #define __IM_PALETTE_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup palette Palette Generators
00017 * \par
00018 * Creates several standard palettes
00019 * \par
00020 * See \ref im_palette.h
00021 * \ingroup util */
00022
00023
00024 /** Searches for the nearest color on the table and returns the
00025 * It looks in all palette entries and finds the minimum euclidean
00026 * If the color matches the given color it returns immediately
00027 * \ingroup palette */
00028 int imPaletteFindNearest(const long *palette, int palette_count,
00029
00030 /** Searches for the color on the table and returns the color
00031 * If the tolerance is 0 search for the exact match in the pal
00032 * first color that fits in the tolerance range.
00033 * \ingroup palette */
00034 int imPaletteFindColor(const long *palette, int palette_count,
00035
00036 /** Creates a palette of gray scale values.
00037 * The colors are arranged from black to white.
00038 * \ingroup palette */
00039 long* imPaletteGray(void);
```

```
00040
00041 /** Creates a palette of a gradient of red colors.
00042 * The colors are arranged from black to pure red.
00043 * \ingroup palette */
00044 long* imPaletteRed(void);
00045
00046 /** Creates a palette of a gradient of green colors.
00047 * The colors are arranged from black to pure green.
00048 * \ingroup palette */
00049 long* imPaletteGreen(void);
00050
00051 /** Creates a palette of a gradient of blue colors.
00052 * The colors are arranged from black to pure blue.
00053 * \ingroup palette */
00054 long* imPaletteBlue(void);
00055
00056 /** Creates a palette of a gradient of yellow colors.
00057 * The colors are arranged from black to pure yellow.
00058 * \ingroup palette */
00059 long* imPaletteYellow(void);
00060
00061 /** Creates a palette of a gradient of magenta colors.
00062 * The colors are arranged from black to pure magenta.
00063 * \ingroup palette */
00064 long* imPaletteMagenta(void);
00065
00066 /** Creates a palette of a gradient of cian colors.
00067 * The colors are arranged from black to pure cian.
00068 * \ingroup palette */
00069 long* imPaletteCian(void);
00070
00071 /** Creates a palette of rainbow colors.
00072 * The colors are arranged in the light wave length spectrum o
00073 * \ingroup palette */
00074 long* imPaletteRainbow(void);
00075
00076 /** Creates a palette of hues with maximum saturation.
00077 * \ingroup palette */
00078 long* imPaletteHues(void);
00079
00080 /** Creates a palette of a gradient of blue colors.
00081 * The colors are arranged from pure blue to white.
00082 * \ingroup palette */
00083 long* imPaletteBlueIce(void);
00084
00085 /** Creates a palette of a gradient from black to white passin
00086 * \ingroup palette */
```

```
00087 long* imPaletteHotIron(void);
00088
00089 /** Creates a palette of a gradient from black to white passing
00090 * \ingroup palette */
00091 long* imPaletteBlackBody(void);
00092
00093 /** Creates a palette with high contrast colors.
00094 * \ingroup palette */
00095 long* imPaletteHighContrast(void);
00096
00097 /** Creates a palette of an uniform range of colors from black
00098 * This is a 2^(2.6) bits per pixel palette.
00099 * \ingroup palette */
00100 long* imPaletteUniform(void);
00101
00102 /** Returns the index of the correspondent RGB color of an uni
00103 * \ingroup palette */
00104 int imPaletteUniformIndex(long color);
00105
00106 /** Returns the index of the correspondent RGB color of an uni
00107 * Uses an 8x8 ordered dither to lookup the index in a halftone
00108 * The spatial position used by the halftone method.
00109 * \ingroup palette */
00110 int imPaletteUniformIndexHalftoned(long color, int x, int y);
00111
00112
00113 #if defined(__cplusplus)
00114 }
00115 #endif
00116
00117 #endif
00118
```

include

im_process.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief Image Processing
00003  *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_PROCESS_H
00009 #define __IM_PROCESS_H
00010
00011 #include "im_process_pon.h"
00012 #include "im_process_loc.h"
00013 #include "im_process_glo.h"
00014 #include "im_process_ana.h"
00015
00016 #if defined(__cplusplus)
00017 extern "C" {
00018 #endif
00019
00020
00021 /** \defgroup process Image Processing
00022  * \par
00023  * Several image processing functions based on the \ref imImage
00024  * \par
00025  * You must link the application with "im_process.lib/.a/.so".
00026  * Some complex operations use the \ref counter.\n
00027  * There is no check on the input/output image properties,
00028  * check each function documentation before using it.
00029  * \par
00030  * See \ref im_process.h
00031 */
00032
00033
00034 #if defined(__cplusplus)
00035 }
00036 #endif
00037
00038 #endif
```

include

im_process_ana.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Image Statistics and Analysis
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_PROC_ANA_H
00009 #define __IM_PROC_ANA_H
00010
00011 #include "im_image.h"
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018
00019 /** \defgroup stats Image Statistics Calculations
00020 * \par
00021 * Operations to calculate some statistics over images.
00022 * \par
00023 * See \ref im_process_ana.h
00024 * \ingroup process */
00025
00026 /** Calculates the RMS error between 2 images (Root Mean Square
00027 * \ingroup stats */
00028 float imCalcRMSError(const imImage* image1, const imImage* image2);
00029
00030 /** Calculates the SNR of an image and its noise (Signal Noise
00031 * \ingroup stats */
00032 float imCalcSNR(const imImage* src_image, const imImage* noise);
00033
00034 /** Count the number of different colors in an image. \n
00035 * Image must be IM_BYTE, but all color spaces except IM_CMYK.
00036 * \ingroup stats */
00037 unsigned long imCalcCountColors(const imImage* src_image);
00038
00039 /** Calculates the histogram of a IM_BYTE data. \n
```

```
00040 * Histogram is always 256 positions long. \n
00041 * When accum is different from zero it calculates the accumul
00042 * \ingroup stats */
00043 void imCalcHistogram(const unsigned char* data, int count, uns
00044
00045 /** Calculates the histogram of a IM_USHORT data. \n
00046 * Histogram is always 65535 positions long. \n
00047 * When accum is different from zero it calculates the accumul
00048 * \ingroup stats */
00049 void imCalcUShortHistogram(const unsigned short* data, int cou
00050
00051 /** Calculates the gray histogram of an image. \n
00052 * Image must be IM_BYTE/(IM_RGB, IM_GRAY, IM_BINARY or IM_MAP
00053 * If the image is IM_RGB then the histogram of the luma compo
00054 * Histogram is always 256 positions long. \n
00055 * When accum is different from zero it calculates the accumul
00056 * \ingroup stats */
00057 void imCalcGrayHistogram(const imImage* src_image, unsigned lo
00058
00059 /** Numerical Statistics Structure
00060 * \ingroup stats */
00061 typedef struct _imStats
00062 {
00063     float max;           /**< Maximum value */
00064     float min;           /**< Minimum value */
00065     unsigned long positive; /**< Number of Positive Values */
00066     unsigned long negative; /**< Number of Negative Values */
00067     unsigned long zeros;    /**< Number of Zeros */
00068     float mean;          /**< Mean */
00069     float stddev;         /**< Standard Deviation */
00070 } imStats;
00071
00072 /** Calculates the statistics about the image data. \n
00073 * There is one stats for each depth plane. For ex: stats[0]=r
00074 * Supports all data types except IM_COMPLEX. \n
00075 * \ingroup stats */
00076 void imCalcImageStatistics(const imImage* src_image, imStats*
00077
00078 /** Calculates the statistics about the image histogram data.\n
00079 * There is one stats for each depth plane. For ex: stats[0]=r
00080 * Only IM_BYTE images are supported.
00081 * \ingroup stats */
00082 void imCalcHistogramStatistics(const imImage* src_image, imSta
00083
00084 /** Calculates some extra statistics about the image histogram
00085 * There is one stats for each depth plane. \n
00086 * Only IM_BYTE images are supported. \n
```

```
00087 * mode will be -1 if more than one max is found.
00088 * \ingroup stats */
00089 void imCalcHistoImageStatistics(const imImage* src_image, int*
00090
00091
00092
00093 /** \defgroup analyze Image Analysis
00094 * \par
00095 * See \ref im_process_ana.h
00096 * \ingroup process */
00097
00098 /** Find white regions in binary image. \n
00099 * Result is IM_USHORT type. Regions can be 4 connected or 8 c
00100 * Returns the number of regions found. Background is marked a
00101 * \ingroup analyze */
00102 int imAnalyzeFindRegions(const imImage* src_image, imImage* ds
00103
00104 /** Measure the actual area of all regions. Holes are not incl
00105 * This is the number of pixels of each region. \n
00106 * Source image is IM_USHORT type (the result of \ref imAnalyz
00107 * data has size the number of regions.
00108 * \ingroup analyze */
00109 void imAnalyzeMeasureArea(const imImage* image, int* area);
00110
00111 /** Measure the polygonal area limited by the perimeter line o
00112 * Notice that some regions may have polygonal area zero. \n
00113 * Source image is IM_USHORT type (the result of \ref imAnalyz
00114 * data has size the number of regions.
00115 * \ingroup analyze */
00116 void imAnalyzeMeasurePerimArea(const imImage* image, float* pe
00117
00118 /** Calculate the centroid position of all regions. Holes are
00119 * Source image is IM_USHORT type (the result of \ref imAnalyz
00120 * data has size the number of regions. If area is NULL will b
00121 * \ingroup analyze */
00122 void imAnalyzeMeasureCentroid(const imImage* image, const int*
00123
00124 /** Calculate the principal major axis slope of all regions. \
00125 * Source image is IM_USHORT type (the result of \ref imAnalyz
00126 * data has size the number of regions. If area or centroid ar
00127 * Principal (major and minor) axes are defined to be those ax
00128 * centroid, about which the moment of inertia of the region i
00129 * \ingroup analyze */
00130 void imAnalyzeMeasurePrincipalAxis(const imImage* image, const
00131                                     const int region_count, flo
00132                                     flo
00133
```

```
00134 /** Measure the number and area of holes of all regions. \n
00135  * Source image is IM_USHORT type (the result of \ref imAnalyz
00136  * data has size the number of regions. If some data is NULL i
00137  * \ingroup analyze */
00138 void imAnalyzeMeasureHoles(const imImage* image, int connect,
00139
00140 /** Measure the total perimeter of all regions (external and i
00141  * Source image is IM_USHORT type (the result of imAnalyzeFind
00142  * It uses a half-pixel inter distance for 8 neigboors in a p
00143  * This function can also be used to measure line lenght. \n
00144  * data has size the number of regions.
00145  * \ingroup analyze */
00146 void imAnalyzeMeasurePerimeter(const imImage* image, float* pe
00147
00148 /** Isolates the perimeter line of gray integer images. Backgr
00149  * It just checks if at least one of the 4 connected neighbor
00150  * \ingroup analyze */
00151 void imProcessPerimeterLine(const imImage* src_image, imImage*
00152
00153 /** Eliminates regions that have size outside the given interv
00154  * Source and destiny are a binary images. Regions can be 4 co
00155  * Regions touching the image border will also be eliminated.
00156  * Can be done in-place. end_size can be zero to ignore big ob
00157  * \ingroup analyze */
00158 void imProcessPrune(const imImage* src_image, imImage* dst_im
00159
00160 /** Fill holes inside white regions. \n
00161  * Source and destiny are a binary images. Regions can be 4 co
00162  * Can be done in-place.
00163  * \ingroup analyze */
00164 void imProcessFillHoles(const imImage* src_image, imImage* dst
00165
00166
00167 #if defined(__cplusplus)
00168 }
00169 #endif
00170
00171 #endif
```

include

im_process_glo.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Image Processing - Global Operations
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_PROCESS_GLO_H
00009 #define __IM_PROCESS_GLO_H
00010
00011 #include "im_image.h"
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018
00019 /** \defgroup transform Domain Transform Operations
00020 * \par
00021 * FFT, Wavelets, Hough, Distance.
00022 * \par
00023 * FFTW Copyright Matteo Frigo, Steven G. Johnson and the MIT.
00024 * http://www.fftw.org
00025 * See fftw.h or fftw3.h
00026 * \par
00027 * Must link with "im_fftw.lib" for FFTW version 2.1.5, and "i
00028 * Both libraries are available because version 3 was not that
00029 * and its file size is 3x bigger than version 2. But version
00030 * optimizations.
00031 * \par
00032 * The FFTW lib has a GPL license. The license of the "im_fftw
00033 * So you cannot use it for commercial applications without co
00034 * \par
00035 * See \ref im_process_glo.h
00036 * \ingroup process */
00037
00038 /** Forward FFT. \n
00039 * The result has its lowest frequency at the center of the im
```

```

00040 * This is an unnormalized fft. \n
00041 * Images must be of the same size. Destiny image must be of t
00042 * \ingroup transform */
00043 void imProcessFFT(const imImage* src_image, imImage* dst_image
00044
00045 /** Inverse FFT. \n
00046 * The image has its lowest frequency restored to the origin b
00047 * The result is normalized by (width*height). \n
00048 * Images must be of the same size and both must be of type co
00049 * \ingroup transform */
00050 void imProcessIFFT(const imImage* src_image, imImage* dst_imag
00051
00052 /** Raw in-place FFT (forward or inverse). \n
00053 * The lowest frequency can be centered after forward, or
00054 * can be restored to the origin before inverse. \n
00055 * The result can be normalized after the transform by sqrt(w*
00056 * or left unnormalized [0]). \n
00057 * Images must be of the same size and both must be of type co
00058 * \ingroup transform */
00059 void imProcessFFTRaw(imImage* src_image, int inverse, int cent
00060
00061 /** Auxiliary function for the raw FFT. \n
00062 * This is the function used internally to change the lowest f
00063 * If the image size has even dimensions the flag "center2orig
00064 * you must specify if its from center to origin (usually used
00065 * from origin to center (usually used after forward). \n
00066 * Notice that this function is used for images in the the fre
00067 * Image type must be complex.
00068 * \ingroup transform */
00069 void imProcessSwapQuadrants(imImage* src_image, int center2ori
00070
00071 /** Hough Lines Transform. \n
00072 * It will detect white lines in a black background. So the so
00073 * with the white lines of interest enhanced. The better the t
00074 * the line detection. \n
00075 * The destiny image must have IM_GRAY, IM_INT, width=180, hei
00076 * The houfh transform defines "cos(theta) * X + sin(theta)
00077 * theta = "0 .. 179", rho = "-height/2 .. height/2" .\n
00078 * Returns zero if the counter aborted. \n
00079 * Inspired from ideas in XITE, Copyright 1991, Blab, UiO \n
00080 * http://www.ifi.uio.no/~blab/Software/Xite/
00081 * \ingroup transform */
00082 int imProcessHoughLines(const imImage* src_image, imImage* dst
00083
00084 /** Draw detected hough lines. \n
00085 * The source image must be IM_GRAY and IM_BYTE. The destiny i
00086 * it can be the source image for in place processing. \n

```

```
00087 * The hough points image is a hough transform image that was
00088 * usually using a Local Max threshold operation. Again the be
00089 * The destiny image will be set to IM_MAP, and the detected l
00090 * Returns the number of detected lines.
00091 * \ingroup transform */
00092 int imProcessHoughLinesDraw(const imImage* src_image, const im
00093
00094 /** Calculates the Cross Correlation in the frequency domain.
00095 * CrossCorr(a,b) = IFFT(Conj(FFT(a))*FFT(b)) \n
00096 * Images must be of the same size and only destiny image must
00097 * \ingroup transform */
00098 void imProcessCrossCorrelation(const imImage* src_image1, cons
00099
00100 /** Calculates the Auto Correlation in the frequency domain. \
00101 * Uses the cross correlation.
00102 * Images must be of the same size and only destiny image must
00103 * \ingroup transform */
00104 void imProcessAutoCorrelation(const imImage* src_image, imImag
00105 /** Calculates the Distance Transform of a binary image
00106 * using an approximation of the euclidian distance.\n
00107 * Each white pixel in the binary image is
00108 * assigned a value equal to its distance from the nearest
00109 * black pixel. \n
00110 * Uses a two-pass algorithm incrementally calculating the dis
00111 * Source image must be IM_BINARY, destiny must be IM_FLOAT.
00112 * \ingroup transform */
00113 void imProcessDistanceTransform(const imImage* src_image, imIm
00114
00115 /** Marks all the regional maximum of the distance transform.
00116 * source is IMGRAY/IM_FLOAT destiny in IM_BINARY. \n
00117 * We consider maximum all connected pixel values that have sm
00118 * \ingroup transform */
00119 void imProcessRegionalMaximum(const imImage* src_image, imImag
00120
00121
00122 #if defined(__cplusplus)
00123 }
00124 #endif
00125
00126 #endif
```

include

im_process_loc.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Image Processing - Local Operations
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_PROCESS_LOC_H
00009 #define __IM_PROCESS_LOC_H
00010
00011 #include "im_image.h"
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018
00019 /** \defgroup resize Image Resize
00020 * \par
00021 * Operations to change the image size.
00022 * \par
00023 * See \ref im_process_loc.h
00024 * \ingroup process */
00025
00026 /** Only reduze the image size using the given decimation order.
00027 * Supported interpolation orders:
00028 * \li 0 - zero order (mean)
00029 * \li 1 - first order (bilinear decimation)
00030 * Images must be of the same type. \n
00031 * Returns zero if the counter aborted.
00032 * \ingroup resize */
00033 int imProcessReduce(const imImage* src_image, imImage* dst_im
00034
00035 /** Change the image size using the given interpolation order.
00036 * Supported interpolation orders:
00037 * \li 0 - zero order (near neighborhood)
00038 * \li 1 - first order (bilinear interpolation)
00039 * \li 3 - third order (bicubic interpolation)
```

```
00040 * Images must be of the same type. \n
00041 * Returns zero if the counter aborted.
00042 * \ingroup resize */
00043 int imProcessResize(const imImage* src_image, imImage* dst_image)
00044
00045 /** Reduce the image area by 4 (w/2,h/2). \n
00046 * Images must be of the same type. Destiny image size must be
00047 * \ingroup resize */
00048 void imProcessReduceBy4(const imImage* src_image, imImage* dst_image)
00049
00050 /** Reduce the image size by removing pixels. \n
00051 * Images must be of the same type. Destiny image size must be
00052 * \ingroup resize */
00053 void imProcessCrop(const imImage* src_image, imImage* dst_image)
00054
00055 /** Increase the image size by adding pixels with zero value.
00056 * Images must be of the same type. Destiny image size must be
00057 * \ingroup resize */
00058 void imProcessAddMargins(const imImage* src_image, imImage* dst_image)
00059
00060
00061
00062 /** \defgroup geom Geometric Operations
00063 * \par
00064 * Operations to change the shape of the image.
00065 * \par
00066 * See \ref im_process_loc.h
00067 * \ingroup process */
00068
00069 /** Calculates the size of the new image after rotation.
00070 * \ingroup geom */
00071 void imProcessCalcRotateSize(int width, int height, int *new_width,
00072 *new_height)
00073 /** Rotates the image using the given interpolation order (see
00074 * Images must be of the same type. The destiny size can be calculated
00075 * Returns zero if the counter aborted.
00076 * \ingroup geom */
00077 int imProcessRotate(const imImage* src_image, imImage* dst_image)
00078
00079 /** Rotate the image in 90 degrees counterclockwise or clockwise.
00080 * Images must be of the same type. Destiny width and height must be
00081 * \ingroup geom */
00082 void imProcessRotate90(const imImage* src_image, imImage* dst_image)
00083
00084 /** Rotate the image in 180 degrees. Swap columns and swap lines.
00085 * Images must be of the same type and size.
00086 * \ingroup geom */
```

```
00087 void imProcessRotate180(const imImage* src_image, imImage* dst
00088
00089 /** Mirrors the image in a horizontal flip. Swap columns. \n
00090 * Images must be of the same type and size.
00091 * \ingroup geom */
00092 void imProcessMirror(const imImage* src_image, imImage* dst_im
00093
00094 /** Apply a vertical flip. Swap lines. \n
00095 * Images must be of the same type and size.
00096 * \ingroup geom */
00097 void imProcessFlip(const imImage* src_image, imImage* dst_imag
00098
00099 /** Apply a radial distortion using the given interpolation or
00100 * Images must be of the same type and size. Returns zero if t
00101 * \ingroup geom */
00102 int imProcessRadial(const imImage* src_image, imImage* dst_im
00103
00104
00105
00106 /** \defgroup morphgray Morphology Operations for Gray Images
00107 * \par
00108 * See \ref im_process_loc.h
00109 * \ingroup process */
00110
00111 /** Base gray morphology convolution. \n
00112 * Supports all data types except IM_COMPLEX. Can be applied o
00113 * Kernel is always IM_INT. Use kernel size odd for better res
00114 * You can use the maximum value or else the minimum value. \n
00115 * No border extensions are used.
00116 * All the gray morphology operations use this function. \n
00117 * If the kernel image attribute "Description" exists it is us
00118 * \ingroup morphgray */
00119 int imProcessGrayMorphConvolve(const imImage* src_image, imIma
00120
00121 /** Gray morphology convolution with a kernel full of "0"s and
00122 * \ingroup morphgray */
00123 int imProcessGrayMorphErode(const imImage* src_image, imImage*
00124
00125 /** Gray morphology convolution with a kernel full of "0"s and
00126 * \ingroup morphgray */
00127 int imProcessGrayMorphDilate(const imImage* src_image, imImage
00128
00129 /** Erode+Dilate.
00130 * \ingroup morphgray */
00131 int imProcessGrayMorphOpen(const imImage* src_image, imImage*
00132
00133 /** Dilate+Erode.
```

```

00134  * \ingroup morphgray */
00135 int imProcessGrayMorphClose(const imImage* src_image, imImage*
00136
00137 /** Open+Difference.
00138  * \ingroup morphgray */
00139 int imProcessGrayMorphTopHat(const imImage* src_image, imImage*
00140
00141 /** Close+Difference.
00142  * \ingroup morphgray */
00143 int imProcessGrayMorphWell(const imImage* src_image, imImage*
00144
00145 /** Difference(Erode, Dilate).
00146  * \ingroup morphgray */
00147 int imProcessGrayMorphGradient(const imImage* src_image, imIma
00148
00149
00150
00151 /** \defgroup morphbin Morphology Operations for Binary Images
00152  * \par
00153  * See \ref im_process_loc.h
00154  * \ingroup process */
00155
00156 /** Base binary morphology convolution. \n
00157  * Images are all IM_BINARY. Kernel is IM_INT. Use kernel size
00158  * Hit white means hit=1 and miss=0, or else hit=0 and miss=1.
00159  * Use -1 for don't care positions in kernel. \n
00160  * The operation can be repeated by a number of iterations.
00161  * The border is zero extended. \n
00162  * Almost all the binary morphology operations use this functi
00163  * If the kernel image attribute "Description" exists it is us
00164  * \ingroup morphbin */
00165 int imProcessBinMorphConvolve(const imImage* src_image, imImag
00166
00167 /** Binary morphology convolution with a kernel full of "1"s a
00168  * \ingroup morphbin */
00169 int imProcessBinMorphErode(const imImage* src_image, imImage*
00170
00171 /** Binary morphology convolution with a kernel full of "0"s a
00172  * \ingroup morphbin */
00173 int imProcessBinMorphDilate(const imImage* src_image, imImage*
00174
00175 /** Erode+Dilate.
00176  * When iteration is more than one it means Erode+Erode+Erode+
00177  * \ingroup morphbin */
00178 int imProcessBinMorphOpen(const imImage* src_image, imImage* d
00179
00180 /** Dilate+Erode.

```

```
00181 * \ingroup morphbin */
00182 int imProcessBinMorphClose(const imImage* src_image, imImage*
00183
00184 /** Erode+Difference. \n
00185 * The difference from the source image is applied only once.
00186 * \ingroup morphbin */
00187 int imProcessBinMorphOutline(const imImage* src_image, imImage
00188
00189 /** Thins the supplied binary image using Rosenfeld's parallel
00190 * Reference: \n
00191 * "Efficient Binary Image Thinning using Neighborhood Maps" \
00192 * by Joseph M. Cychosz, 3ksnn64@ecn.purdue.edu \
00193 * in "Graphics Gems IV", Academic Press, 1994
00194 * \ingroup morphbin */
00195 void imProcessBinMorphThin(const imImage* src_image, imImage*
00196
00197
00198
00199 /** \defgroup rank Rank Convolution Operations
00200 * \par
00201 * All the rank convolution use the same base function. Near t
00202 * includes only the real image pixels in the rank. No border
00203 * \par
00204 * See \ref im_process_loc.h
00205 * \ingroup process */
00206
00207 /** Rank convolution using the median value. \n
00208 * Returns zero if the counter aborted. \n
00209 * Supports all data types except IM_COMPLEX. Can be applied o
00210 * \ingroup rank */
00211 int imProcessMedianConvolve(const imImage* src_image, imImage*
00212
00213 /** Rank convolution using (maximum-minimum) value. \n
00214 * Returns zero if the counter aborted. \n
00215 * Supports all data types except IM_COMPLEX. Can be applied o
00216 * \ingroup rank */
00217 int imProcessRangeConvolve(const imImage* src_image, imImage*
00218
00219 /** Rank convolution using the closest maximum or minimum value
00220 * Returns zero if the counter aborted. \n
00221 * Supports all data types except IM_COMPLEX. Can be applied o
00222 * \ingroup rank */
00223 int imProcessRankClosestConvolve(const imImage* src_image, imI
00224
00225 /** Rank convolution using the maximum value. \n
00226 * Returns zero if the counter aborted. \n
00227 * Supports all data types except IM_COMPLEX. Can be applied o
```

```
00228 * \ingroup rank */
00229 int imProcessRankMaxConvolve(const imImage* src_image, imImage
00230
00231 /** Rank convolution using the minimum value. \n
00232 * Returns zero if the counter aborted. \n
00233 * Supports all data types except IM_COMPLEX. Can be applied o
00234 * \ingroup rank */
00235 int imProcessRankMinConvolve(const imImage* src_image, imImage
00236
00237 /** Threshold using a rank convolution with a range contrast f
00238 * Supports all integer IM_GRAY images as source, and IM_BINAR
00239 * Local variable threshold by the method of Bernsen. \n
00240 * Extracted from XITE, Copyright 1991, Blab, UiO \n
00241 * http://www.ifi.uio.no/~blab/Software/Xite/
00242 \verbatim
00243 Reference:
00244     Bernsen, J: "Dynamic thresholding of grey-level images"
00245     Proc. of the 8th ICPR, Paris, Oct 1986, 1251-1255.
00246 Author: Oivind Due Trier
00247 \endverbatim
00248 * Returns zero if the counter aborted.
00249 * \ingroup threshold */
00250 int imProcessRangeContrastThreshold(const imImage* src_image,
00251
00252 /** Threshold using a rank convolution with a local max functi
00253 * Returns zero if the counter aborted. \n
00254 * Supports all integer IM_GRAY images as source, and IM_BINAR
00255 * \ingroup threshold */
00256 int imProcessLocalMaxThreshold(const imImage* src_image, imImage
00257
00258
00259
00260 /** \defgroup convolve Convolution Operations
00261 * \par
00262 * See \ref im_process_loc.h
00263 * \ingroup process */
00264
00265 /** Base Convolution with a kernel. \n
00266 * Kernel can be IM_INT or IM_FLOAT, but always IM_GRAY. Use k
00267 * Supports all data types. The border is mirrored. \n
00268 * Returns zero if the counter aborted. Most of the convolution
00269 * If the kernel image attribute "Description" exists it is us
00270 * \ingroup convolve */
00271 int imProcessConvolve(const imImage* src_image, imImage* dst_i
00272
00273 /** Repeats the convolution a number of times. \n
00274 * Returns zero if the counter aborted.\n
```

```
00275 * If the kernel image attribute "Description" exists it is us
00276 * \ingroup convolve */
00277 int imProcessConvolveRep(const imImage* src_image, imImage* ds
00278
00279 /** Convolve with a kernel rotating it 8 times and getting the
00280 * Kernel must be square. \n
00281 * The rotation is implemented only for kernel sizes 3x3, 5x5
00282 * Supports all data types except IM_COMPLEX.
00283 * Returns zero if the counter aborted.\n
00284 * If the kernel image attribute "Description" exists it is us
00285 * \ingroup convolve */
00286 int imProcessCompassConvolve(const imImage* src_image, imImage
00287
00288 /** Utility function to rotate a kernel one time.
00289 * \ingroup convolve */
00290 void imProcessRotateKernel(imImage* kernel);
00291
00292 /** Difference(Gaussian1, Gaussian2). \n
00293 * Supports all data types,
00294 * but if source is IM_BYTE or IM USHORT destiny image must be
00295 * \ingroup convolve */
00296 int imProcessDiffOfGaussianConvolve(const imImage* src_image,
00297
00298 /** Difference(Gaussian1, Gaussian2) using gaussian repetition
00299 * Supports all data types,
00300 * but if source is IM_BYTE or IM USHORT destiny image must be
00301 * \ingroup convolve */
00302 int imProcessDiffOfGaussianConvolveRep(const imImage* src_imag
00303
00304 /** Convolution with a laplacian of a gaussian kernel. \n
00305 * Supports all data types,
00306 * but if source is IM_BYTE or IM USHORT destiny image must be
00307 * \ingroup convolve */
00308 int imProcessLapOfGaussianConvolve(const imImage* src_image, i
00309
00310 /** Convolution with a kernel full of "1"s inside a circle. \n
00311 * Supports all data types.
00312 * \ingroup convolve */
00313 int imProcessMeanConvolve(const imImage* src_image, imImage* d
00314
00315 /** Convolution with a gaussian kernel. The gaussian is obtain
00316 * Supports all data types.
00317 * \ingroup convolve */
00318 int imProcessGaussianConvolveRep(const imImage* src_image, imI
00319
00320 /** Convolution with a float gaussian kernel. \n
00321 * Supports all data types.
```

```
00322 * \ingroup convolve */
00323 int imProcessGaussianConvolve(const imImage* src_image, imImage*
00324
00325 /** Magnitude of the sobel convolution. \n
00326 * Supports all data types except IM_COMPLEX.
00327 * \ingroup convolve */
00328 int imProcessSobelConvolve(const imImage* src_image, imImage*
00329
00330 /** Finds the zero crossings of IM_INT and IM_FLOAT images. Cr
00331 * indicating the intensity of the edge. It is usually used af
00332 * Extracted from XITE, Copyright 1991, Blab, UiO \n
00333 * http://www.ifi.uio.no/~blab/Software/Xite/
00334 * \ingroup convolve */
00335 void imProcessZeroCrossing(const imImage* src_image, imImage*
00336
00337 /** First part of the Canny edge detector. Includes the gaussi
00338 * After using this you could apply a Hysteresis Threshold, se
00339 * Image must be IM_BYTE/IM_GRAY. \n
00340 * Implementation from the book:
00341 \verbatim
00342     J. R. Parker
00343     "Algoritms for Image Processing and Computer Vision"
00344     WILEY
00345 \endverbatim
00346 * \ingroup convolve */
00347 void imProcessCanny(const imImage* src_image, imImage* dst_im
00348
00349 /** Calculates the number of 3x3 gaussian repetitions given th
00350 * \ingroup convolve */
00351 int imGaussianStdDev2Repetitions(float stddev);
00352
00353 /** Calculates the kernel size given the standard deviation.
00354 * \ingroup convolve */
00355 int imGaussianStdDev2KernelSize(float stddev);
00356
00357
00358 #if defined(__cplusplus)
00359 }
00360 #endif
00361
00362 #endif
```

include

im_process_pon.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Image Processing - Pontual Operations
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_PROCESS_PON_H
00009 #define __IM_PROCESS_PON_H
00010
00011 #include "im_image.h"
00012
00013 #if defined(__cplusplus)
00014 extern "C" {
00015 #endif
00016
00017
00018
00019 /** \defgroup arithm Arithmetic Operations
00020 * \par
00021 * Simple math operations for images.
00022 * \par
00023 * See \ref im_process_pon.h
00024 * \ingroup process */
00025
00026 /** Unary Arithmetic Operations.
00027 * Inverse and log may lead to math exceptions.
00028 * \ingroup arithm */
00029 enum imUnaryOp {
00030     IM_UN_EQ,    /**< equal           =      a      */
00031     IM_UN_ABS,   /**< absolute        =      |a|      */
00032     IM_UN_LESS,  /**< less            =      -a      */
00033     IM_UN_INC,   /**< increment       +=     a      */
00034     IM_UN_INV,   /**< invert           =      1/a      (#) */
00035     IM_UN_SQR,   /**< square           =      a*a      */
00036     IM_UN_SQRT,  /**< square root     =      a^(1/2)  */
00037     IM_UN_LOG,   /**< natural logarithm =      ln(a)   (#) */
00038     IM_UN_EXP,   /**< exponential      =      exp(a)  */
00039     IM_UN_SIN,   /**< sine             =      sin(a)  */
```

```

00040     IM_UN_COS,      /**< cosine           = cos(a)          */
00041     IM_UN_CONJ,    /**< complex conjugate = ar - ai*i
00042     IM_UN_CPNORM /*< complex normalization by magnitude = a /
00043 };
00044
00045 /** Apply an arithmetic unary operation. \n
00046 * Can be done in place, images must match size, does not need
00047 * \ingroup arithm */
00048 void imProcessUnArithmeticOp(const imImage* src_image, imImage
00049
00050 /** Binary Arithmetic Operations.
00051 * Inverse and log may lead to math exceptions.
00052 * \ingroup arithm */
00053 enum imBinaryOp {
00054     IM_BIN_ADD,      /**< add       = a+b          */
00055     IM_BIN_SUB,      /**< subtract   = a-b          */
00056     IM_BIN_MUL,      /**< multiply   = a*b          */
00057     IM_BIN_DIV,      /**< divide     = a/b          (#) */
00058     IM_BIN_DIFF,     /**< difference = |a-b|        */
00059     IM_BIN_POW,      /**< power      = a^b          */
00060     IM_BIN_MIN,      /**< minimum    = (a < b)? a: b */
00061     IM_BIN_MAX,      /**< maximum   = (a > b)? a: b */
00062 };
00063
00064 /** Apply a binary arithmetic operation. \n
00065 * Can be done in place, images must match size. \n
00066 * Source images must match type, destiny image can be several
00067 * \li byte -> byte, ushort, int, float
00068 * \li ushort -> ushort, int, float
00069 * \li int -> int, float
00070 * \li float -> float
00071 * \li complex -> complex
00072 * One exception is that you can combine complex with float re
00073 * \ingroup arithm */
00074 void imProcessArithmeticOp(const imImage* src_image1, const im
00075
00076 /** Apply a binary arithmetic operation with a constant value.
00077 * Can be done in place, images must match size. \n
00078 * Destiny image can be several types depending on source: \n
00079 * \li byte -> byte, ushort, int, float
00080 * \li ushort -> byte, ushort, int, float
00081 * \li int -> byte, ushort, int, float
00082 * \li float -> float
00083 * \li complex -> complex
00084 * The constant value is type casted to an appropriate type bef
00085 * \ingroup arithm */
00086 void imProcessArithmeticConstOp(const imImage* src_image, floa

```

```
00087
00088 /** Blend two images using an alpha value = [a * alpha + b * (
00089 * Can be done in place, images must match size and type.
00090 * \ingroup arithm */
00091 void imProcessBlend(const imImage* src_image1, imImage* src_im
00092
00093 /** Split a complex image into two images with real and imagin
00094 * or magnitude and phase parts (polar = 1). \n
00095 * Source image must be IM_COMPLEX, destiny images must be IM_
00096 * \ingroup arithm */
00097 void imProcessSplitComplex(const imImage* src_image, imImage*
00098
00099 /** Merges two images as the real and imaginary parts of a com
00100 * or as magnitude and phase parts (polar = 1). \n
00101 * Source images must be IM_FLOAT, destiny image must be IM_CO
00102 * \ingroup arithm */
00103 void imProcessMergeComplex(const imImage* src_image1, const im
00104
00105 /** Calculates the mean of multiple images. \n
00106 * Images must match size and type.
00107 * \ingroup arithm */
00108 void imProcessMultipleMean(const imImage** src_image_list, int
00109
00110 /** Calculates the standard deviation of multiple images. \n
00111 * Images must match size and type.
00112 * \ingroup arithm */
00113 void imProcessMultipleStdDev(const imImage** src_image_list, i
00114
00115 /** Calculates the auto-covariance of an image with the mean o
00116 * Images must match size and type. Returns zero if the counte
00117 * \ingroup arithm */
00118 int imProcessAutoCovariance(const imImage* src_image, const im
00119
00120 /** Multiplies the conjugate of one complex image with another
00121 * Images must match size. Conj(img1) * img2 \n
00122 * Can be done in-place.
00123 * \ingroup arithm */
00124 void imProcessMultiplyConj(const imImage* src_image1, const im
00125
00126
00127
00128 /** \defgroup quantize Additional Image Quantization Operation
00129 * \par
00130 * Additionally operations to the \ref imConvertColorSpace fun
00131 * \par
00132 * See \ref im_process_pon.h
00133 * \ingroup process */
```

```
00134
00135 /** Converts a RGB image to a MAP image using uniform quantiza
00136 * with an optional 8x8 ordered dither. The RGB image must hav
00137 * \ingroup quantize */
00138 void imProcessQuantizeRGBUniform(const imImage* src_image, imI
00139
00140 /** Quantizes a gray scale image in less than 256 grays using
00141 * Both images must be IM_BYTE/IM_GRAY. Can be done in place.
00142 * \ingroup quantize */
00143 void imProcessQuantizeGrayUniform(const imImage* src_image, imI
00144
00145
00146
00147 /** \defgroup histo Histogram Based Operations
00148 * \par
00149 * See \ref im_process_pon.h
00150 * \ingroup process */
00151
00152 /** Performs an histogram expansion. \n
00153 * Percentage defines an amount of pixels to include at start
00154 * If its is zero only empty counts of the histogram will be c
00155 * Images must be IM_BYTE/(IM_RGB or IM_GRAY). Can be done in
00156 * \ingroup histo */
00157 void imProcessExpandHistogram(const imImage* src_image, imImag
00158
00159 /** Performs an histogram equalization. \n
00160 * Images must be IM_BYTE/(IM_RGB or IM_GRAY). Can be done in
00161 * \ingroup histo */
00162 void imProcessEqualizeHistogram(const imImage* src_image, imIm
00163
00164
00165
00166 /** \defgroup colorproc Color Processing Operations
00167 * \par
00168 * Operations to change the color components configuration.
00169 * \par
00170 * See \ref im_process_pon.h
00171 * \ingroup process */
00172
00173 /** Split a RGB image into luma and chroma. \n
00174 * Chroma is calculated as R-Y,G-Y,B-Y. Source image must be I
00175 * luma image is IM_GRAY/IM_BYTE and chroma is IM_RGB/IM_BYTE.
00176 * Source and destiny have the same size.
00177 * \ingroup colorproc */
00178 void imProcessSplitYChroma(const imImage* src_image, imImage*
00179
00180 /** Split a RGB image into HSI planes. \n
```

```
00181 * Source image must be IM_RGB/IM_BYTE. Destiny images are all
00182 * Source and destiny have the same size. See \ref hsi .
00183 * \ingroup colorproc */
00184 void imProcessSplitHSI(const imImage* src_image, imImage* h_im
00185
00186 /** Merge HSI planes into a RGB image. \n
00187 * Source images must be IM_GRAY/IM_BYTE. Destiny image is all
00188 * Source and destiny have the same size. See \ref hsi .
00189 * \ingroup colorproc */
00190 void imProcessMergeHSI(const imImage* h_image, const imImage*
00191
00192 /** Split a multicomponent image into separate components.\n
00193 * Destiny images must be IM_GRAY. Size and data types must be
00194 * The number of destiny images must match the depth of the so
00195 * \ingroup colorproc */
00196 void imProcessSplitComponents(const imImage* src_image, imImag
00197
00198 /** Merges separate components into a multicomponent image.\n
00199 * Source images must be IM_GRAY. Size and data types must be
00200 * The number of source images must match the depth of the des
00201 * \ingroup colorproc */
00202 void imProcessMergeComponents(const imImage** src_image_list,
00203
00204 /** Normalize the color components by their sum. Example: c1 =
00205 * Destiny image must be IM_FLOAT.
00206 * \ingroup colorproc */
00207 void imProcessNormalizeComponents(const imImage* src_image, im
00208
00209 /** Replaces the source color by the destiny color. \n
00210 * The color will be type casted to the image data type. \n
00211 * The colors must have the same number of components of the i
00212 * Supports all color spaces and all data types except IM_COMP
00213 * \ingroup colorproc */
00214 void imProcessReplaceColor(const imImage* src_image, imImage*
00215
00216
00217
00218 /** \defgroup logic Logical Arithmetic Operations
00219 * \par
00220 * Logical binary math operations for images.
00221 * \par
00222 * See \ref im_process_pon.h
00223 * \ingroup process */
00224
00225 /** Logical Operations.
00226 * \ingroup logic */
00227 enum imLogicOp {
```

```
00228     IM_BIT_AND,    /**< and   =   a & b   */
00229     IM_BIT_OR,     /**< or    =   a | b   */
00230     IM_BIT_XOR     /**< xor   = ~(a | b) */
00231 };
00232
00233 /** Apply a logical operation.\n
00234 * Images must have data type IM_BYTE, IM USHORT or IM_INT. Can
00235 * \ingroup logic */
00236 void imProcessBitwiseOp(const imImage* src_image1, const imIma
00237
00238 /** Apply a logical NOT operation.\n
00239 * Images must have data type IM_BYTE, IM USHORT or IM_INT. Can
00240 * \ingroup logic */
00241 void imProcessBitwiseNot(const imImage* src_image, imImage* ds
00242
00243 /** Apply a bit mask. \n
00244 * The same as imProcessBitwiseOp but the second image is repla
00245 * Images must have data type IM_BYTE. It is valid only for AND
00246 * \ingroup logic */
00247 void imProcessBitMask(const imImage* src_image, imImage* dst_i
00248
00249 /** Extract or Reset a bit plane. For ex: 000X0000 or XXX0XXXX
00250 * Images must have data type IM_BYTE. Can be done in place.
00251 * \ingroup logic */
00252 void imProcessBitPlane(const imImage* src_image, imImage* dst_
00253
00254
00255
00256 /** \defgroup render Synthetic Image Render
00257 * \par
00258 * Renders some 2D mathematical functions as images. All the f
00259 * and supports all data types except IM_COMPLEX.
00260 * \par
00261 * See \ref im_process_pon.h
00262 * \ingroup process */
00263
00264 /** Render Funtion.
00265 * \ingroup render */
00266 typedef float (*imRenderFunc)(int x, int y, int d, float* para
00267
00268 /** Render Conditional Funtion.
00269 * \ingroup render */
00270 typedef float (*imRenderCondFunc)(int x, int y, int d, int *co
00271
00272 /** Render a synthetic image using a render function. \n
00273 * plus will make the render be added to the current image dat
00274 * or else all data will be replaced. All the render functions
```

```
00275 * Returns zero if the counter aborted.  
00276 * \ingroup render */  
00277 int imProcessRenderOp(imImage* image, imRenderFunc render_func  
00278  
00279 /** Render a sintetic image using a conditional render function  
00280 * Data will be rendered only if the condional param is true.  
00281 * Returns zero if the counter aborted.  
00282 * \ingroup render */  
00283 int imProcessRenderCondOp(imImage* image, imRenderCondFunc ren  
00284  
00285 /** Render speckle noise on existing data. Can be done in place.  
00286 * \ingroup render */  
00287 int imProcessRenderAddSpeckleNoise(const imImage* src_image, i  
00288  
00289 /** Render gaussian noise on existing data. Can be done in place.  
00290 * \ingroup render */  
00291 int imProcessRenderAddGaussianNoise(const imImage* src_image,  
00292  
00293 /** Render uniform noise on existing data. Can be done in place.  
00294 * \ingroup render */  
00295 int imProcessRenderAddUniformNoise(const imImage* src_image, i  
00296  
00297 /** Render random noise.  
00298 * \ingroup render */  
00299 int imProcessRenderRandomNoise(imImage* image);  
00300  
00301 /** Render a constant. The number of values must match the depth.  
00302 * \ingroup render */  
00303 int imProcessRenderConstant(imImage* image, float* value);  
00304  
00305 /** Render a centered wheel.  
00306 * \ingroup render */  
00307 int imProcessRenderWheel(imImage* image, int int_radius, int e  
00308  
00309 /** Render a centered cone.  
00310 * \ingroup render */  
00311 int imProcessRenderCone(imImage* image, int radius);  
00312  
00313 /** Render a centered tent.  
00314 * \ingroup render */  
00315 int imProcessRenderTent(imImage* image, int width, int height)  
00316  
00317 /** Render a ramp. Direction can be vertical (1) or horizontal (-1).  
00318 * \ingroup render */  
00319 int imProcessRenderRamp(imImage* image, int start, int end, in  
00320  
00321 /** Render a centered box.
```

```

00322 * \ingroup render */
00323 int imProcessRenderBox(imImage* image, int width, int height);
00324
00325 /** Render a centered sinc.
00326 * \ingroup render */
00327 int imProcessRenderSinc(imImage* image, float xperiod, float y
00328
00329 /** Render a centered gaussian.
00330 * \ingroup render */
00331 int imProcessRenderGaussian(imImage* image, float stddev);
00332
00333 /** Render the laplacian of a centered gaussian.
00334 * \ingroup render */
00335 int imProcessRenderLapOfGaussian(imImage* image, float stddev)
00336
00337 /** Render a centered cosine.
00338 * \ingroup render */
00339 int imProcessRenderCosine(imImage* image, float xperiod, float
00340
00341 /** Render a centered grid.
00342 * \ingroup render */
00343 int imProcessRenderGrid(imImage* image, int x_space, int y_spa
00344
00345 /** Render a centered chessboard.
00346 * \ingroup render */
00347 int imProcessRenderChessboard(imImage* image, int x_space, int
00348
00349
00350
00351 /** \defgroup tonegamut Tone Gamut Operations
00352 * \par
00353 * Operations that try to preserve the min-max interval in the
00354 * \par
00355 * See \ref im_process_pon.h
00356 * \ingroup process */
00357
00358
00359 /** Tone Gamut Operations.
00360 * \ingroup tonegamut */
00361 enum imToneGamut {
00362     IM_GAMUT_NORMALIZE, /**< normalize = (a-min) / (max-min)
00363     IM_GAMUT_POW,        /**< pow      = ((a-min) / (max-min))^gamma
00364                                         param[0]=gamma
00365     IM_GAMUT_LOG,        /**< log      = log(K * (a-min) / (max-min))
00366                                         param[0]=K      (K>0)
00367     IM_GAMUT_EXP,        /**< exp      = (exp(K * (a-min) / (max-min)) - 1) / K
00368                                         param[0]=K

```

```

00369     IM_GAMUT_INVERT,    /**< invert      = max - (a-min)
00370     IM_GAMUT_ZEROSTART, /*< zerostart = a - min
00371     IM_GAMUT_SOLARIZE,  /**< solarize   = a < level ? a: (level
00372                                param[0]=level percenta
00373                                photography solarizatio
00374     IM_GAMUT_SLICE,     /**< slice       = start < a || a > end ?
00375                                param[0]=start, param[
00376     IM_GAMUT_EXPAND,    /**< expand     = a < start ? min: a > e
00377                                param[0]=start, param[
00378     IM_GAMUT_CROP,      /**< crop       = a < start ? start: a >
00379                                param[0]=start, param[
00380     IM_GAMUT_BRIGHTCONT /*< brightcont = a < min ? min: a > m
00381                                param[0]=bright_shift
00382                                change brightness and
00383 };
00384
00385 /** Apply a gamut operation with arguments. \n
00386 * Supports all data types except IM_COMPLEX. \n
00387 * The linear operation do a special conversion when min > 0 a
00388 * IM_BYTE images have min=0 and max=255 always. \n
00389 * Can be done in place. When there is no extra params use NUL
00390 * \ingroup tonegamut */
00391 void imProcessToneGamut(const imImage* src_image, imImage* dst_
00392
00393 /** Converts from (0-1) to (0-255), crop out of bounds values.
00394 * Source image must be IM_FLOAT, and destiny image must be IM_
00395 * \ingroup tonegamut */
00396 void imProcessUnNormalize(const imImage* src_image, imImage* d
00397
00398 /** Directly converts IM_USHORT, IM_INT and IM_FLOAT into IM_B
00399 * This can also be done using \ref imConvertDataType with IM_
00400 * \ingroup tonegamut */
00401 void imProcessDirectConv(const imImage* src_image, imImage* ds
00402
00403 /** A negative effect. Uses \ref imProcessToneGamut with IM_GA
00404 * Supports all color spaces and all data types except IM_COMP
00405 * \ingroup tonegamut */
00406 void imProcessNegative(const imImage* src_image, imImage* dst_
00407
00408
00409
00410 /** \defgroup threshold Threshold Operations
00411 * \par
00412 * Operations that converts a usually IM_GRAY/IM_BYTE image in
00413 * \par
00414 * See \ref im_process_pon.h
00415 * \ingroup process */

```

```

00416
00417 /** Apply a manual threshold. \n
00418 * threshold = a <= level ? 0: value \n
00419 * Normal value is 1 but another common value is 255. Can be d
00420 * Supports all integer IM_GRAY images as source, and IM_BINAR
00421 * \ingroup threshold */
00422 void imProcessThreshold(const imImage* src_image, imImage* dst
00423
00424 /** Apply a threshold by the difference of two images. \n
00425 * threshold = a1 <= a2 ? 0: 1 \n
00426 * Can be done in place.
00427 * \ingroup threshold */
00428 void imProcessThresholdByDiff(const imImage* src_image1, const
00429
00430 /** Apply a threshold by the Hysteresis method. \n
00431 * Hysteresis thersholding of edge pixels. Starting at pixels
00432 * value greater than the HIGH threshold, trace a connected se
00433 * of pixels that have a value greater than the LOW threhsold.
00434 * Note: could not find the original source code author name.
00435 * \ingroup threshold */
00436 void imProcessHysteresisThreshold(const imImage* src_image, im
00437
00438 /** Estimates hysteresis low and high threshold levels.
00439 * \ingroup threshold */
00440 void imProcessHysteresisThresEstimate(const imImage* src_image
00441
00442 /** Calculates the threshold level for manual threshold using
00443 * Extracted from XITE, Copyright 1991, Blab, UiO \n
00444 * http://www.ifi.uio.no/~blab/Software/Xite/
00445 \verbatim
00446 Reference:
00447     S. M. Dunn & D. Harwood & L. S. Davis:
00448     "Local Estimation of the Uniform Error Threshold"
00449     IEEE Trans. on PAMI, Vol PAMI-6, No 6, Nov 1984.
00450 Comments: It only works well on images whith large objects.
00451 Author: Olav Borgli, BLAB, ifi, UiO
00452 Image processing lab, Department of Informatics, University
00453 \endverbatim
00454 * Returns the used level.
00455 * \ingroup threshold */
00456 int imProcessUniformErrThreshold(const imImage* src_image, imI
00457
00458 /** Apply a dithering on each image channel by using a difusion
00459 * It can be applied on any IM_BYTE images. It will "threshold"
00460 * source and destiny must be of the same depth.
00461 * \ingroup threshold */
00462 void imProcessDifusionErrThreshold(const imImage* src_image, i

```

```
00463
00464 /** Calculates the threshold level for manual threshold using
00465 * that should stay bellow the threshold. \n
00466 * Returns the used level.
00467 * \ingroup threshold */
00468 int imProcessPercentThreshold(const imImage* src_image, imImage*
00469
00470 /** Calculates the threshold level for manual threshold using
00471 * Returns the used level. \n
00472 * Original implementation by Flavio Szenberg.
00473 * \ingroup threshold */
00474 int imProcessOtsuThreshold(const imImage* src_image, imImage*
00475
00476 /** Calculates the threshold level for manual threshold using
00477 * Returns the used level. \n
00478 * Supports all integer IM_GRAY images as source, and IM_BINARY
00479 * \ingroup threshold */
00480 int imProcessMinMaxThreshold(const imImage* src_image, imImage*
00481
00482 /** Estimates Local Max threshold level for IM_BYTE images.
00483 * \ingroup threshold */
00484 void imProcessLocaMaxThresEstimate(const imImage* src_image, imImage*
00485
00486 /** Apply a manual threshold using an interval. \n
00487 * threshold = start_level <= a <= end_level ? 1: 0 \n
00488 * Normal value is 1 but another common value is 255. Can be done
00489 * Supports all integer IM_GRAY images as source, and IM_BINARY
00490 * \ingroup threshold */
00491 void imProcessSliceThreshold(const imImage* src_image, imImage*
00492
00493
00494 /** \defgroup effects Special Effects
00495 * \par
00496 * Operations to change image appearance.
00497 * \par
00498 * See \ref im_process_pon.h
00499 * \ingroup process */
00500
00501
00502 /** Generates a zoom in effect averaging colors inside a square
00503 * Operates only on IM_BYTE images.
00504 * \ingroup effects */
00505 void imProcessPixelate(const imImage* src_image, imImage* dst_
00506
00507 /** A simple Posterize effect. It reduces the number of colors
00508 * less significant bit planes. Can have 1 to 7 levels. See \ref
00509 * Image data type must be integer.
```

```
00510  * \ingroup effects */
00511 void imProcessPosterize(const imImage* src_image, imImage* dst
00512
00513
00514
00515 #if defined(__cplusplus)
00516 }
00517 #endif
00518
00519 #endif
```

include

im_raw.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief RAW File Format
00003  *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_RAW_H
00009 #define __IM_RAW_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** Opens a RAW image file.
00017  * \ingroup raw */
00018 imFile* imFileOpenRaw(const char* file_name, int *error);
00019
00020 /** Creates a RAW image file.
00021  * \ingroup raw */
00022 imFile* imFileNewRaw(const char* file_name, int *error);
00023
00024
00025 #if defined(__cplusplus)
00026 }
00027 #endif
00028
00029 #endif
```

include

im_util.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Utilities
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_UTIL_H
00009 #define __IM_UTIL_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015
00016 /** \defgroup util Utilities
00017 * \par
00018 * See \ref im_util.h
00019 * @{
00020 */
00021
00022 #define IM_MIN(_a, _b) (_a < _b? _a: _b)
00023 #define IM_MAX(_a, _b) (_a > _b? _a: _b)
00024
00025 /** @} */
00026
00027
00028 /** \defgroup str String Utilities
00029 * \par
00030 * See \ref im_util.h
00031 * \ingroup util */
00032
00033 /** Check if the two strings are equal.
00034 * \ingroup str */
00035 int imStrEqual(const char* str1, const char* str2);
00036
00037 /** Calculate the size of the string but limited to max_len.
00038 * \ingroup str */
00039 int imStrNLen(const char* str, int max_len);
```

```
00040
00041 /** Check if the data is a string.
00042 * \ingroup str */
00043 int imStrCheck(const void* data, int count);
00044
00045
00046
00047 /** \defgroup imageutil Image Utilities
00048 * \par
00049 * See \ref im_util.h
00050 * \ingroup imagerep */
00051
00052 /** Returns the size of the data buffer.
00053 * \ingroup imageutil */
00054 int imImageContentSize(int width, int height, int color_mode, int
00055
00056 /** Returns the size of one line of the data buffer. \n
00057 * This depends if the components are packed. If packed includ
00058 * \ingroup imageutil */
00059 int imImageLineSize(int width, int color_mode, int data_type);
00060
00061 /** Returns the number of elements of one line of the data buf
00062 * This depends if the components are packed. If packed includ
00063 * \ingroup imageutil */
00064 int imImageLineCount(int width, int color_mode);
00065
00066 /** Check the combination color_mode+data_type.
00067 * \ingroup imageutil */
00068 int imImageCheckFormat(int color_mode, int data_type);
00069
00070
00071
00072 /** \defgroup colorutl Color Utilities
00073 * \par
00074 * See \ref im_util.h
00075 * \ingroup util */
00076
00077 /** Encode RGB components in a long for palette usage. \n
00078 * "long" definition is compatible with the CD library definit
00079 * \ingroup colorutl */
00080 long imColorEncode(unsigned char red, unsigned char green, uns
00081
00082 /** Decode RGB components from a long for palette usage. \n
00083 * "long" definition is compatible with the CD library definit
00084 * \ingroup colorutl */
00085 void imColorDecode(unsigned char *red, unsigned char *green, u
00086
```

```
00087
00088
00089 /** \defgroup colormodeutl Color Mode Utilities
00090 * \par
00091 * See \ref im_util.h
00092 * \ingroup imagerep */
00093
00094 /** Returns the color mode name.
00095 * \ingroup colormodeutl */
00096 const char* imColorModeSpaceName(int color_mode);
00097
00098 /** Returns the number of components of the color space includ
00099 * \ingroup colormodeutl */
00100 int imColorModeDepth(int color_mode);
00101
00102 /** Returns the color space of the color mode.
00103 * \ingroup colormodeutl */
00104 #define imColorModeSpace(_cm) (_cm & 0xFF)
00105
00106 /** Check if the two color modes match. Only the color space i
00107 * \ingroup colormodeutl */
00108 #define imColorModeMatch(_cm1, _cm2) (imColorModeSpace(_cm1) =
00109
00110 /** Check if the color mode has an alpha channel.
00111 * \ingroup colormodeutl */
00112 #define imColorModeHasAlpha(_cm) (_cm & IM_ALPHA)
00113
00114 /** Check if the color mode components are packed in one plane
00115 * \ingroup colormodeutl */
00116 #define imColorModeIsPacked(_cm) (_cm & IM_PACKED)
00117
00118 /** Check if the color mode orients the image from top down to
00119 * \ingroup colormodeutl */
00120 #define imColorModeIsTopDown(_cm) (_cm & IM_TOPDOWN)
00121
00122 /** Returns the color mode of the equivalent display bitmap im
00123 * Original packing and alpha are ignored. Returns IM_RGB, IM_
00124 * \ingroup colormodeutl */
00125 int imColorModeToBitmap(int color_mode);
00126
00127 /** Check if the color mode and data_type defines a display bi
00128 * \ingroup colormodeutl */
00129 int imColorModeIsBitmap(int color_mode, int data_type);
00130
00131
00132
00133 /** \defgroup datatypeutl Data Type Utilities
```

```
00134 * \par
00135 * See \ref im_util.h
00136 * \ingroup util
00137 * @{
00138 */
00139
00140 typedef unsigned char imbyte;
00141 typedef unsigned short imushort;
00142
00143 #define IM_BYTECROP(_v) (_v < 0? 0: _v > 255? 255: _v)
00144 #define IM_CROPMAX(_v, _max) (_v < 0? 0: _v > _max? _max: _v)
00145
00146 /** @} */
00147
00148 /** Returns the size in bytes of a specified numeric data type
00149 * \ingroup datatypeutl */
00150 int imDataTypeSize(int data_type);
00151
00152 /** Returns the numeric data type name given its identifier.
00153 * \ingroup datatypeutl */
00154 const char* imDataTypeName(int data_type);
00155
00156 /** Returns the maximum value of an integer data type. For float
00157 * \ingroup datatypeutl */
00158 unsigned long imDataTypeIntMax(int data_type);
00159
00160 /** Returns the minimum value of an integer data type. For float
00161 * \ingroup datatypeutl */
00162 long imDataTypeIntMin(int data_type);
00163
00164
00165
00166 /** \defgroup bin Binary Data Utilities
00167 * \par
00168 * See \ref im_util.h
00169 * \ingroup util */
00170
00171 /** CPU Byte Orders.
00172 * \ingroup bin */
00173 enum imByteOrder
00174 {
00175     IM_LITTLEENDIAN, /**< Little Endian - The most significant byte
00176     IM_BIGENDIAN      /**< Big Endian - The most significant byte
00177 };
00178
00179 /** Returns the current CPU byte order.
00180 * \ingroup bin */
```

```
00181 int imBinCPUByteOrder(void);
00182
00183 /** Changes the byte order of an array of 2, 4 or 8 byte values
00184 * \ingroup bin */
00185 void imBinSwapBytes(void *data, int count, int size);
00186
00187 /** Changes the byte order of an array of 2 byte values.
00188 * \ingroup bin */
00189 void imBinSwapBytes2(void *data, int count);
00190
00191 /** Inverts the byte order of the 4 byte values
00192 * \ingroup bin */
00193 void imBinSwapBytes4(void *data, int count);
00194
00195 /** Inverts the byte order of the 8 byte values
00196 * \ingroup bin */
00197 void imBinSwapBytes8(void *data, int count);
00198
00199
00200
00201 /** \defgroup compress Data Compression Utilities
00202 * \par
00203 * See \ref im_util.h
00204 * \ingroup util */
00205
00206 /** Compresses the data using the ZLIB Deflate compression. \n
00207 * The destination buffer must be at least 0.1% larger than source. \n
00208 * It compresses raw byte data. zip_quality can be 1 to 9. \n
00209 * Returns the size of the compressed buffer.
00210 * \ingroup compress */
00211 int imCompressDataZ(const void* src_data, int src_size, void*
00212
00213 /** Uncompresses the data compressed with the ZLIB Deflate com-
00214 * \ingroup compress */
00215 int imCompressDataUnZ(const void* src_data, int src_size, void*
00216
00217
00218 #if defined(__cplusplus)
00219 }
00220 #endif
00221
00222 #endif
```

include

imlua.h

```
00001 #ifndef _IM_LUA_
00002 #define _IM_LUA_
00003
00004 /***** 
00005 * Initializes IMLua.
00006 \*****
00007 void imlua_open(void);
00008
00009 #endif
```

include

old_im.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002 * \brief Old API
00003 *
00004 * See Copyright Notice in im_lib.h
00005 * $Id: Exp $
00006 */
00007
00008 #ifndef __IM_OLD_H
00009 #define __IM_OLD_H
00010
00011 #if defined(__cplusplus)
00012 extern "C" {
00013 #endif
00014
00015 enum {IM_BMP, IM_PCX, IM_GIF, IM_TIF, IM_RAS, IM_SGI, IM_JPG,
00016 enum {IM_NONE = 0x0000, IM_DEFAULT = 0x0100, IM_COMPRESSED = 0
00017
00018 #define IM_ERR_READ IM_ERR_ACCESS
00019 #define IM_ERR_WRITE IM_ERR_ACCESS
00020 #define IM_ERR_TYPE IM_ERR_DATA
00021 #define IM_ERR_COMP IM_ERR_COMPRESS
00022
00023 long imEncodeColor(unsigned char red, unsigned char green, uns
00024 void imDecodeColor(unsigned char* red, unsigned char* green, u
00025 int imFileFormat(char *filename, int* format);
00026 int imImageInfo(char *filename, int *width, int *height, int *
00027 int imLoadRGB(char *filename, unsigned char *red, unsigned char
00028 int imSaveRGB(int width, int height, int format, unsigned char
00029 int imLoadMap(char *filename, unsigned char *map, long *palett
00030 int imSaveMap(int width, int height, int format, unsigned char
00031 void imRGB2Map(int width, int height, unsigned char *red, unsi
00032 void imMap2RGB(int width, int height, unsigned char *map, int
00033 void imRGB2Gray(int width, int height, unsigned char *red, uns
00034 void imMap2Gray(int width, int height, unsigned char *map, int
00035 void imResize(int src_width, int src_height, unsigned char *sr
00036 void imStretch(int src_width, int src_height, unsigned char *s
00037 typedef int (*imCallback)(char *filename);
00038 int imRegisterCallback(imCallback cb, int cb_id, int format);
00039
```

```
00040 #define IM_INTERRUPTED -1
00041 #define IM_ALL -1
00042 #define IM_COUNTER_CB 0
00043 typedef int (*imFileCounterCallback)(char *filename, int perce
00044
00045 #define IM_RESOLUTION_CB 1
00046 typedef int (*imResolutionCallback)(char *filename, double* xr
00047
00048 enum {IM_RES_NONE, IM_RES_DPI, IM_RES_DPC};
00049
00050 #define IM_GIF_TRANSPARENT_COLOR_CB 0
00051 typedef int (*imGifTranspIndex)(char *filename, unsigned char
00052
00053 #define IM_TIF_IMAGE_DESCRIPTION_CB 0
00054 typedef int (*imTiffImageDesc)(char *filename, char* img_desc)
00055
00056 #if defined(__cplusplus)
00057 }
00058 #endif
00059
00060 #endif
```

include

include Directory Reference

Files

file [im.h](#)
file [im_attrib.h](#)
file [im_attrib_flat.h](#)
file [im_binfile.h](#)
file [im_capture.h](#)
file [im_color.h](#)
file [im_colorhsi.h](#)
file [im_complex.h](#)
file [im_convert.h](#)
file [im_counter.h](#)
file [im_dib.h](#)
file [im_file.h](#)
file [im_format.h](#)
file [im_format_all.h](#)
file [im_format_avi.h](#)
file [im_format_jp2.h](#)
file [im_format_raw.h](#)
file [im_format_wmv.h](#)
file [im_image.h](#)
file [im_lib.h](#)
file [im_math.h](#)
file [im_math_op.h](#)
file [im_palette.h](#)
file [im_plus.h](#)
file [im_process.h](#)
file [im_process_ana.h](#)
file [im_process_glo.h](#)

file **im_process_loc.h**
file **im_process_pon.h**
file **im_raw.h**
file **im_util.h**
file **imlua.h**
file **old_im.h**

- a -

- abs_op() : [im_math_op.h](#)
- add_op() : [im_math_op.h](#)

- C -

- cos_op() : [im_math_op.h](#)
- crop_byte() : [im_math_op.h](#)

- d -

- diff_op() : [im_math_op.h](#)
- div_op() : [im_math_op.h](#)

- e -

- exp_op() : [im_math_op.h](#)

- i -

- imAnalyzeFindRegions() : [im_process_ana.h](#)
- imAnalyzeMeasureArea() : [im_process_ana.h](#)
- imAnalyzeMeasureCentroid() : [im_process_ana.h](#)

- imAnalyzeMeasureHoles() : [im_process_ana.h](#)
- imAnalyzeMeasurePerimArea() : [im_process_ana.h](#)
- imAnalyzeMeasurePerimeter() : [im_process_ana.h](#)
- imAnalyzeMeasurePrincipalAxis() : [im_process_ana.h](#)
- imBicubicInterpolation() : [im_math.h](#)
- imBilinearDecimation() : [im_math.h](#)
- imBilinearInterpolation() : [im_math.h](#)
- imBinCPUByteOrder() : [im_util.h](#)
- imBinFileByteOrder() : [im_binfile.h](#)
- imBinFileClose() : [im_binfile.h](#)
- imBinFileEndOfFile() : [im_binfile.h](#)
- imBinFileError() : [im_binfile.h](#)
- imBinFileNew() : [im_binfile.h](#)
- imBinFileOpen() : [im_binfile.h](#)
- imBinFilePrintf() : [im_binfile.h](#)
- imBinFileRead() : [im_binfile.h](#)
- imBinFileSeekFrom() : [im_binfile.h](#)
- imBinFileSeekOffset() : [im_binfile.h](#)
- imBinFileSeekTo() : [im_binfile.h](#)
- imBinFileSetCurrentModule() : [im_binfile.h](#)
- imBinFileSize() : [im_binfile.h](#)
- imBinFileTell() : [im_binfile.h](#)
- imBinFileWrite() : [im_binfile.h](#)
- imBinSwapBytes() : [im_util.h](#)
- imBinSwapBytes2() : [im_util.h](#)
- imBinSwapBytes4() : [im_util.h](#)
- imBinSwapBytes8() : [im_util.h](#)
- imCalcCountColors() : [im_process_ana.h](#)
- imCalcGrayHistogram() : [im_process_ana.h](#)
- imCalcHistogram() : [im_process_ana.h](#)
- imCalcHistogramStatistics() : [im_process_ana.h](#)
- imCalcHistoImageStatistics() : [im_process_ana.h](#)
- imCalcImageStatistics() : [im_process_ana.h](#)
- imCalcRMSError() : [im_process_ana.h](#)
- imCalcSNR() : [im_process_ana.h](#)
- imCalcUShortHistogram() : [im_process_ana.h](#)
- imColorCMYK2RGB() : [im_color.h](#)
- imColorDecode() : [im_util.h](#)
- imColorEncode() : [im_util.h](#)

- imColorHSI2RGB() : [im_colorhsi.h](#)
- imColorHSI2RGBbyte() : [im_colorhsi.h](#)
- imColorHSI_ImaxS() : [im_colorhsi.h](#)
- imColorHSI_Smax() : [im_colorhsi.h](#)
- imColorLab2XYZ() : [im_color.h](#)
- imColorLightness2Luminance() : [im_color.h](#)
- imColorLuminance2Lightness() : [im_color.h](#)
- imColorLuv2XYZ() : [im_color.h](#)
- imColorMax() : [im_color.h](#)
- imColorModeDepth() : [im_util.h](#)
- imColorModeIsBitmap() : [im_util.h](#)
- imColorModeSpaceName() : [im_util.h](#)
- imColorModeToBitmap() : [im_util.h](#)
- imColorQuantize() : [im_color.h](#)
- imColorReconstruct() : [im_color.h](#)
- imColorRGB2HSI() : [im_colorhsi.h](#)
- imColorRGB2HSIbyte() : [im_colorhsi.h](#)
- imColorRGB2Luma() : [im_color.h](#)
- imColorRGB2RGBNonlinear() : [im_color.h](#)
- imColorRGB2XYZ() : [im_color.h](#)
- imColorRGB2YCbCr() : [im_color.h](#)
- imColorTransfer2Linear() : [im_color.h](#)
- imColorTransfer2Nonlinear() : [im_color.h](#)
- imColorXYZ2Lab() : [im_color.h](#)
- imColorXYZ2Luv() : [im_color.h](#)
- imColorXYZ2RGB() : [im_color.h](#)
- imColorYCbCr2RGB() : [im_color.h](#)
- imColorZero() : [im_color.h](#)
- imCompressDataUnZ() : [im_util.h](#)
- imCompressDataZ() : [im_util.h](#)
- imConvertColorSpace() : [im_convert.h](#)
- imConvertDataType() : [im_convert.h](#)
- imConvertMapToRGB() : [im_convert.h](#)
- imConvertPacking() : [im_convert.h](#)
- imConvertToBitmap() : [im_convert.h](#)
- imCounterBegin() : [im_counter.h](#)
- imCounterEnd() : [im_counter.h](#)
- imCounterInc() : [im_counter.h](#)
- imCounterSetCallback() : [im_counter.h](#)

- imCounterTotal() : **im_counter.h**
- imDataBitGet() : **im_math_op.h**
- imDataBitSet() : **im_math_op.h**
- imDataTypeIntMax() : **im_util.h**
- imDataTypeIntMin() : **im_util.h**
- imDataTypeName() : **im_util.h**
- imDataTypeSize() : **im_util.h**
- imDibCaptureScreen() : **im_dib.h**
- imDibCopyClipboard() : **im_dib.h**
- imDibCreate() : **im_dib.h**
- imDibCreateCopy() : **im_dib.h**
- imDibCreateReference() : **im_dib.h**
- imDibCreateSection() : **im_dib.h**
- imDibDecodeToBitmap() : **im_dib.h**
- imDibDecodeToMap() : **im_dib.h**
- imDibDecodeToRGBA() : **im_dib.h**
- imDibDestroy() : **im_dib.h**
- imDibEncodeFromBitmap() : **im_dib.h**
- imDibEncodeFromMap() : **im_dib.h**
- imDibEncodeFromRGBA() : **im_dib.h**
- imDibFromHBitmap() : **im_dib.h**
- imDibIsClipboardAvailable() : **im_dib.h**
- imDibLineGetPixelFunc() : **im_dib.h**
- imDibLineSetPixelFunc() : **im_dib.h**
- imDibLoadFile() : **im_dib.h**
- imDibLogicalPalette() : **im_dib.h**
- imDibPasteClipboard() : **im_dib.h**
- imDibSaveFile() : **im_dib.h**
- imDibToHBitmap() : **im_dib.h**
- imFileClose() : **im.h**
- imFileGetAttribute() : **im.h**
- imFileGetAttributeList() : **im.h**
- imFileGetInfo() : **im.h**
- imFileGetPalette() : **im.h**
- imFileHandle() : **im.h**
- imFileLineBufferCount() : **im_file.h**
- imFileLineBufferInc() : **im_file.h**
- imFileLineBufferRead() : **im_file.h**
- imFileLineBufferWrite() : **im_file.h**

- imFileSizeAligned() : **im_file.h**
- imFileLoadBitmap() : **im_image.h**
- imFileLoadImage() : **im_image.h**
- imFileNew() : **im.h**
- imFileNewRaw() : **im_raw.h**
- imFileOpen() : **im.h**
- imFileOpenRaw() : **im_raw.h**
- imFileReadImageData() : **im.h**
- imFileReadImageInfo() : **im.h**
- imFileSaveImage() : **im_image.h**
- imFileSetAttribute() : **im.h**
- imFileSetInfo() : **im.h**
- imFileSetPalette() : **im.h**
- imFileWriteImageData() : **im.h**
- imFileWriteImageInfo() : **im.h**
- imFormatCanWriteImage() : **im.h**
- imFormatCompressions() : **im.h**
- imFormatInfo() : **im.h**
- imFormatList() : **im.h**
- imFormatRegister() : **im_format.h**
- imFormatRegisterAVI() : **im_format_avi.h**
- imFormatRegisterJP2() : **im_format_jp2.h**
- imFormatRegisterWMV() : **im_format_wmv.h**
- imGaussianStdDev2KernelSize() : **im_process_loc.h**
- imGaussianStdDev2Repetitions() : **im_process_loc.h**
- imImageCheckFormat() : **im_util.h**
- imImageClear() : **im_image.h**
- imImageClone() : **im_image.h**
- imImageCopy() : **im_image.h**
- imImageCopyAttributes() : **im_image.h**
- imImageCopyData() : **im_image.h**
- imImageCreate() : **im_image.h**
- imImageDataSize() : **im_util.h**
- imImageDestroy() : **im_image.h**
- imImageDuplicate() : **im_image.h**
- imImageGetAttribute() : **im_image.h**
- imImageGetAttributeList() : **im_image.h**
- imImageInit() : **im_image.h**
- imImageIsBitmap() : **im_image.h**

- imImageLineCount() : [**im_util.h**](#)
- imImageLineSize() : [**im_util.h**](#)
- imImageLoad() : [**im_image.h**](#)
- imImageLoadBitmap() : [**im_image.h**](#)
- imImageMakeBinary() : [**im_image.h**](#)
- imImageMatch() : [**im_image.h**](#)
- imImageMatchColor() : [**im_image.h**](#)
- imImageMatchColorSpace() : [**im_image.h**](#)
- imImageMatchDataType() : [**im_image.h**](#)
- imImageMatchSize() : [**im_image.h**](#)
- imImageReshape() : [**im_image.h**](#)
- imImageSetAttribute() : [**im_image.h**](#)
- imImageSetBinary() : [**im_image.h**](#)
- imImageSetPalette() : [**im_image.h**](#)
- imMinMax() : [**im_math.h**](#)
- imPaletteBlackBody() : [**im_palette.h**](#)
- imPaletteBlue() : [**im_palette.h**](#)
- imPaletteBlueIce() : [**im_palette.h**](#)
- imPaletteCian() : [**im_palette.h**](#)
- imPaletteFindColor() : [**im_palette.h**](#)
- imPaletteFindNearest() : [**im_palette.h**](#)
- imPaletteGray() : [**im_palette.h**](#)
- imPaletteGreen() : [**im_palette.h**](#)
- imPaletteHighContrast() : [**im_palette.h**](#)
- imPaletteHotIron() : [**im_palette.h**](#)
- imPaletteHues() : [**im_palette.h**](#)
- imPaletteMagenta() : [**im_palette.h**](#)
- imPaletteRainbow() : [**im_palette.h**](#)
- imPaletteRed() : [**im_palette.h**](#)
- imPaletteUniform() : [**im_palette.h**](#)
- imPaletteUniformIndex() : [**im_palette.h**](#)
- imPaletteUniformIndexHalftoned() : [**im_palette.h**](#)
- imPaletteYellow() : [**im_palette.h**](#)
- imProcessAddMargins() : [**im_process_loc.h**](#)
- imProcessArithmeticConstOp() : [**im_process_pon.h**](#)
- imProcessArithmeticOp() : [**im_process_pon.h**](#)
- imProcessAutoCorrelation() : [**im_process_glo.h**](#)
- imProcessAutoCovariance() : [**im_process_pon.h**](#)
- imProcessBinMorphClose() : [**im_process_loc.h**](#)

- imProcessBinMorphConvolve() : [**im_process_loc.h**](#)
- imProcessBinMorphDilate() : [**im_process_loc.h**](#)
- imProcessBinMorphErode() : [**im_process_loc.h**](#)
- imProcessBinMorphOpen() : [**im_process_loc.h**](#)
- imProcessBinMorphOutline() : [**im_process_loc.h**](#)
- imProcessBinMorphThin() : [**im_process_loc.h**](#)
- imProcessBitMask() : [**im_process_pon.h**](#)
- imProcessBitPlane() : [**im_process_pon.h**](#)
- imProcessBitwiseNot() : [**im_process_pon.h**](#)
- imProcessBitwiseOp() : [**im_process_pon.h**](#)
- imProcessBlend() : [**im_process_pon.h**](#)
- imProcessCalcRotateSize() : [**im_process_loc.h**](#)
- imProcessCanny() : [**im_process_loc.h**](#)
- imProcessCompassConvolve() : [**im_process_loc.h**](#)
- imProcessConvolve() : [**im_process_loc.h**](#)
- imProcessConvolveRep() : [**im_process_loc.h**](#)
- imProcessCrop() : [**im_process_loc.h**](#)
- imProcessCrossCorrelation() : [**im_process_glo.h**](#)
- imProcessDiffOfGaussianConvolve() : [**im_process_loc.h**](#)
- imProcessDiffOfGaussianConvolveRep() : [**im_process_loc.h**](#)
- imProcessDifusionErrThreshold() : [**im_process_pon.h**](#)
- imProcessDirectConv() : [**im_process_pon.h**](#)
- imProcessDistanceTransform() : [**im_process_glo.h**](#)
- imProcessEqualizeHistogram() : [**im_process_pon.h**](#)
- imProcessExpandHistogram() : [**im_process_pon.h**](#)
- imProcessFFT() : [**im_process_glo.h**](#)
- imProcessFFTraw() : [**im_process_glo.h**](#)
- imProcessFillHoles() : [**im_process_ana.h**](#)
- imProcessFlip() : [**im_process_loc.h**](#)
- imProcessGaussianConvolve() : [**im_process_loc.h**](#)
- imProcessGaussianConvolveRep() : [**im_process_loc.h**](#)
- imProcessGrayMorphClose() : [**im_process_loc.h**](#)
- imProcessGrayMorphConvolve() : [**im_process_loc.h**](#)
- imProcessGrayMorphDilate() : [**im_process_loc.h**](#)
- imProcessGrayMorphErode() : [**im_process_loc.h**](#)
- imProcessGrayMorphGradient() : [**im_process_loc.h**](#)
- imProcessGrayMorphOpen() : [**im_process_loc.h**](#)
- imProcessGrayMorphTopHat() : [**im_process_loc.h**](#)
- imProcessGrayMorphWell() : [**im_process_loc.h**](#)

- imProcessHoughLines() : [**im_process_glo.h**](#)
- imProcessHoughLinesDraw() : [**im_process_glo.h**](#)
- imProcessHysteresisThresEstimate() : [**im_process_pon.h**](#)
- imProcessHysteresisThreshold() : [**im_process_pon.h**](#)
- imProcessIFFT() : [**im_process_glo.h**](#)
- imProcessLapOfGaussianConvolve() : [**im_process_loc.h**](#)
- imProcessLocalMaxThreshold() : [**im_process_loc.h**](#)
- imProcessLocaMaxThresEstimate() : [**im_process_pon.h**](#)
- imProcessMeanConvolve() : [**im_process_loc.h**](#)
- imProcessMedianConvolve() : [**im_process_loc.h**](#)
- imProcessMergeComplex() : [**im_process_pon.h**](#)
- imProcessMergeComponents() : [**im_process_pon.h**](#)
- imProcessMergeHSI() : [**im_process_pon.h**](#)
- imProcessMinMaxThreshold() : [**im_process_pon.h**](#)
- imProcessMirror() : [**im_process_loc.h**](#)
- imProcessMultipleMean() : [**im_process_pon.h**](#)
- imProcessMultipleStdDev() : [**im_process_pon.h**](#)
- imProcessMultiplyConj() : [**im_process_pon.h**](#)
- imProcessNegative() : [**im_process_pon.h**](#)
- imProcessNormalizeComponents() : [**im_process_pon.h**](#)
- imProcessOtsuThreshold() : [**im_process_pon.h**](#)
- imProcessPercentThreshold() : [**im_process_pon.h**](#)
- imProcessPerimeterLine() : [**im_process_ana.h**](#)
- imProcessPixelate() : [**im_process_pon.h**](#)
- imProcessPosterize() : [**im_process_pon.h**](#)
- imProcessPrune() : [**im_process_ana.h**](#)
- imProcessQuantizeGrayUniform() : [**im_process_pon.h**](#)
- imProcessQuantizeRGBUniform() : [**im_process_pon.h**](#)
- imProcessRadial() : [**im_process_loc.h**](#)
- imProcessRangeContrastThreshold() : [**im_process_loc.h**](#)
- imProcessRangeConvolve() : [**im_process_loc.h**](#)
- imProcessRankClosestConvolve() : [**im_process_loc.h**](#)
- imProcessRankMaxConvolve() : [**im_process_loc.h**](#)
- imProcessRankMinConvolve() : [**im_process_loc.h**](#)
- imProcessReduce() : [**im_process_loc.h**](#)
- imProcessReduceBy4() : [**im_process_loc.h**](#)
- imProcessRegionalMaximum() : [**im_process_glo.h**](#)
- imProcessRenderAddGaussianNoise() : [**im_process_pon.h**](#)
- imProcessRenderAddSpeckleNoise() : [**im_process_pon.h**](#)

- imProcessRenderAddUniformNoise() : [im_process_pon.h](#)
- imProcessRenderBox() : [im_process_pon.h](#)
- imProcessRenderChessboard() : [im_process_pon.h](#)
- imProcessRenderCondOp() : [im_process_pon.h](#)
- imProcessRenderCone() : [im_process_pon.h](#)
- imProcessRenderConstant() : [im_process_pon.h](#)
- imProcessRenderCosine() : [im_process_pon.h](#)
- imProcessRenderGaussian() : [im_process_pon.h](#)
- imProcessRenderGrid() : [im_process_pon.h](#)
- imProcessRenderLapOfGaussian() : [im_process_pon.h](#)
- imProcessRenderOp() : [im_process_pon.h](#)
- imProcessRenderRamp() : [im_process_pon.h](#)
- imProcessRenderRandomNoise() : [im_process_pon.h](#)
- imProcessRenderSinc() : [im_process_pon.h](#)
- imProcessRenderTent() : [im_process_pon.h](#)
- imProcessRenderWheel() : [im_process_pon.h](#)
- imProcessReplaceColor() : [im_process_pon.h](#)
- imProcessResize() : [im_process_loc.h](#)
- imProcessRotate() : [im_process_loc.h](#)
- imProcessRotate180() : [im_process_loc.h](#)
- imProcessRotate90() : [im_process_loc.h](#)
- imProcessRotateKernel() : [im_process_loc.h](#)
- imProcessSliceThreshold() : [im_process_pon.h](#)
- imProcessSobelConvolve() : [im_process_loc.h](#)
- imProcessSplitComplex() : [im_process_pon.h](#)
- imProcessSplitComponents() : [im_process_pon.h](#)
- imProcessSplitHSI() : [im_process_pon.h](#)
- imProcessSplitYChroma() : [im_process_pon.h](#)
- imProcessSwapQuadrants() : [im_process_glo.h](#)
- imProcessThreshold() : [im_process_pon.h](#)
- imProcessThresholdByDiff() : [im_process_pon.h](#)
- imProcessToneGamut() : [im_process_pon.h](#)
- imProcessUnArithmeticOp() : [im_process_pon.h](#)
- imProcessUniformErrThreshold() : [im_process_pon.h](#)
- imProcessUnNormalize() : [im_process_pon.h](#)
- imProcessZeroCrossing() : [im_process_loc.h](#)
- imStrCheck() : [im_util.h](#)
- imStrEqual() : [im_util.h](#)
- imStrNLen() : [im_util.h](#)

- imVersion() : **im_lib.h**
- imVersionDate() : **im_lib.h**
- imVersionNumber() : **im_lib.h**
- imVideoCaptureConnect() : **im_capture.h**
- imVideoCaptureCreate() : **im_capture.h**
- imVideoCaptureDestroy() : **im_capture.h**
- imVideoCaptureDeviceCount() : **im_capture.h**
- imVideoCaptureDeviceDesc() : **im_capture.h**
- imVideoCaptureDialogCount() : **im_capture.h**
- imVideoCaptureDialogDesc() : **im_capture.h**
- imVideoCaptureDisconnect() : **im_capture.h**
- imVideoCaptureFrame() : **im_capture.h**
- imVideoCaptureGetAttribute() : **im_capture.h**
- imVideoCaptureGetAttributeList() : **im_capture.h**
- imVideoCaptureGetImageSize() : **im_capture.h**
- imVideoCaptureLive() : **im_capture.h**
- imVideoCaptureOneFrame() : **im_capture.h**
- imVideoCaptureReloadDevices() : **im_capture.h**
- imVideoCaptureResetAttribute() : **im_capture.h**
- imVideoCaptureSetAttribute() : **im_capture.h**
- imVideoCaptureSetImageSize() : **im_capture.h**
- imVideoCaptureShowDialog() : **im_capture.h**
- imZeroOrderDecimation() : **im_math.h**
- imZeroOrderInterpolation() : **im_math.h**
- inv_op() : **im_math_op.h**

- | -

- less_op() : **im_math_op.h**
- log_op() : **im_math_op.h**

- m -

- max_op() : **im_math_op.h**

- min_op() : **im_math_op.h**
- mul_op() : **im_math_op.h**

- p -

- pow_op() : **im_math_op.h**

- S -

- sin_op() : **im_math_op.h**
- sqr_op() : **im_math_op.h**
- sqrt_op() : **im_math_op.h**
- sub_op() : **im_math_op.h**

[All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumeration values](#) | [Defines](#)

- imAttribTableCallback : [im_attrib_flat.h](#)
- imBinMemoryFileName : [im_binfile.h](#)
- imCounterCallback : [im_counter.h](#)
- imDib : [im_dib.h](#)
- imDibLineGetPixel : [im_dib.h](#)
- imDibLineSetPixel : [im_dib.h](#)
- imFormatFunc : [im_format.h](#)
- imImage : [im_image.h](#)
- imRenderCondFunc : [im_process_pon.h](#)
- imRenderFunc : [im_process_pon.h](#)
- imStats : [im_process_ana.h](#)

[All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumeration values](#) | [Defines](#)

- imBinaryOp : [im_process_pon.h](#)
- imBinFileModule : [im_binfile.h](#)
- imByteOrder : [im_util.h](#)
- imCastMode : [im_convert.h](#)
- imColorModeConfig : [im.h](#)
- imColorSpace : [im.h](#)
- imComplex2Real : [im_convert.h](#)
- imDataType : [im.h](#)
- imErrorCodes : [im.h](#)
- imGammaFactor : [im_convert.h](#)
- imLogicOp : [im_process_pon.h](#)
- imToneGamut : [im_process_pon.h](#)
- imUnaryOp : [im_process_pon.h](#)

- i -

- IM_ALPHA : [im.h](#)
- IM_BIGENDIAN : [im_util.h](#)
- IM_BIN_ADD : [im_process_pon.h](#)
- IM_BIN_DIFF : [im_process_pon.h](#)
- IM_BIN_DIV : [im_process_pon.h](#)
- IM_BIN_MAX : [im_process_pon.h](#)
- IM_BIN_MIN : [im_process_pon.h](#)
- IM_BIN_MUL : [im_process_pon.h](#)
- IM_BIN_POW : [im_process_pon.h](#)
- IM_BIN_SUB : [im_process_pon.h](#)
- IM_BINARY : [im.h](#)
- IM_BIT_AND : [im_process_pon.h](#)
- IM_BIT_OR : [im_process_pon.h](#)
- IM_BIT_XOR : [im_process_pon.h](#)
- IM_BYTE : [im.h](#)
- IM_CAST_DIRECT : [im_convert.h](#)
- IM_CAST_FIXED : [im_convert.h](#)
- IM_CAST_MINMAX : [im_convert.h](#)
- IM_CFLOAT : [im.h](#)
- IM_CMYK : [im.h](#)
- IM_ERR_ACCESS : [im.h](#)
- IM_ERR_COMPRESS : [im.h](#)
- IM_ERR_COUNTER : [im.h](#)
- IM_ERR_DATA : [im.h](#)
- IM_ERR_FORMAT : [im.h](#)
- IM_ERR_MEM : [im.h](#)
- IM_ERR_NONE : [im.h](#)
- IM_ERR_OPEN : [im.h](#)
- IM_FLOAT : [im.h](#)
- IM_GAMUT_BRIGHTCONT : [im_process_pon.h](#)
- IM_GAMUT_CROP : [im_process_pon.h](#)

- IM_GAMUT_EXP : [im_process_pon.h](#)
- IM_GAMUT_EXPAND : [im_process_pon.h](#)
- IM_GAMUT_INVERT : [im_process_pon.h](#)
- IM_GAMUT_LOG : [im_process_pon.h](#)
- IM_GAMUT_NORMALIZE : [im_process_pon.h](#)
- IM_GAMUT_POW : [im_process_pon.h](#)
- IM_GAMUT_SLICE : [im_process_pon.h](#)
- IM_GAMUT_SOLARIZE : [im_process_pon.h](#)
- IM_GAMUT_ZEROSTART : [im_process_pon.h](#)
- IM_GRAY : [im.h](#)
- IM_INT : [im.h](#)
- IM_IOCUSTOM0 : [im_binfile.h](#)
- IM_LAB : [im.h](#)
- IM_LITTLEENDIAN : [im_util.h](#)
- IM_LUV : [im.h](#)
- IM_MAP : [im.h](#)
- IM_MEMFILE : [im_binfile.h](#)
- IM_PACKED : [im.h](#)
- IM_RAWFILE : [im_binfile.h](#)
- IM_RGB : [im.h](#)
- IM_STREAM : [im_binfile.h](#)
- IM_SUBFILE : [im_binfile.h](#)
- IM_TOPDOWN : [im.h](#)
- IM_UN_ABS : [im_process_pon.h](#)
- IM_UN_CONJ : [im_process_pon.h](#)
- IM_UN_COS : [im_process_pon.h](#)
- IM_UN_CPNORM : [im_process_pon.h](#)
- IM_UN_EQ : [im_process_pon.h](#)
- IM_UN_EXP : [im_process_pon.h](#)
- IM_UN_INC : [im_process_pon.h](#)
- IM_UN_INV : [im_process_pon.h](#)
- IM_UN_LESS : [im_process_pon.h](#)
- IM_UN_LOG : [im_process_pon.h](#)
- IM_UN_SIN : [im_process_pon.h](#)
- IM_UN_SQR : [im_process_pon.h](#)
- IM_UN_sqrt : [im_process_pon.h](#)
- IM USHORT : [im.h](#)
- IM_XYZ : [im.h](#)
- IM_YCBCR : [im.h](#)

- cdPutBitmap : [im_image.h](#)
- IM_VERSION_NUMBER : [im_lib.h](#)
- imColorModeHasAlpha : [im_util.h](#)
- imColorModeIsPacked : [im_util.h](#)
- imColorModeIsTopDown : [im_util.h](#)
- imColorModeMatch : [im_util.h](#)
- imColorModeSpace : [im_util.h](#)