
IM Image Representation, Storage, Capture and Processing

Version 3.1

(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.

IM Project Management:

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.

IM has been used in Tecgraf for many theses and dissertations. Check the Publications in Tecgraf's web site <http://www.tecgraf.puc-rio.br/>.

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, FreeBSD 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.**

We host **IM** support features at **LuaForge**. It provides us Tracker, Lists, News, CVS and Files. The **IM** page at **LuaForge** is available at: <http://luaforge.net/projects/imtoolkit/>.

The discussion list is available at:

<http://lists.luaforge.net/mailman/listinfo/imtoolkit-users>.

You can also submit *Bugs*, *Feature Requests* and *Support Requests* at: http://luaforge.net/tracker/?group_id=86.

Source code, pre-compiled binaries and samples can be downloaded at: http://luaforge.net/frs/?group_id=86.

The CVS can be browsed at: http://luaforge.net/scm/?group_id=86.

If you want us to develop a specific feature for the toolkit, Tecgraf is available for partnerships and cooperation. Please contact **tcg AT tecgraf.puc-rio.br**.

Lua documentation and resources can be found at <http://www.lua.org/>.

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 Scuri*
- *Antonio Nabuco Tartarini*
- *Diego Fernandes Nehab*
- *Erick de Moura Ferreira*
- *Carolina Alfaro*

We also thank the developers of the third party libraries:

- 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](#))
- Jason Perkins ([Premake](#))
- Marc Alexander Lehmann ([libLZF](#))
- (to many others that contribute to these library, keeping them free and updated)

The IM toolkit distribution includes the some third party libraries that are not developed by Tecgraf. Their license are also free and have the same freedom as the [Tecgraf Library License](#). You can read the respective licenses in the files: [zlib.txt](#), [libpng.txt](#), [libjpeg.txt](#), [libtiff.txt](#), [libjasper.txt](#), [liblzf.txt](#).

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. A poster illustrating the toolkit is available in Adobe Acrobat format ([poster.pdf](#) ~1.4Mb)

The documentation is also available in Adobe Acrobat ([im.pdf](#) ~900Kb) 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

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

The spirit of this license is that you are free to use the libraries for any purpose at no cost without having to ask us. The only requirement is that if you do use them, 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 libraries are 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 conditions here presented.

The Tecgraf products under this license are: [IUP](#), [CD](#) and [IM](#).

Copyright © 1994-2005 [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.

IM Download

The main download site is the **LuaForge** site available at:

http://luaforge.net/project/showfiles.php?group_id=86

When LuaForge is offline, the **Tecgraf Download Center** is activated to offer a mirror site.

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

Before downloading any precompiled binaries, you should read before the [Tecgraf Library Download Tips](#).

Some other files are available directly at the **IM** download folder:

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

History

Version 3.1 (12 Dez 2005)

- New Download, Discussion List, Submission of Bugs, Support Requests and Feature Requests, are now available thanks to the LuaForge site.
- New Binding for Lua 5
- New support for alpha in `imImage`.
- New organization of the documentation.
- New in ICON format the `TransparencyIndex` is used to for **IM_MAP** images without an alpha channel.
- New video capture functions: `imVideoCaptureFormatCount`, `imVideoCaptureGetFormat` and `imVideoCaptureSetFormat`, to access the available capture video formats.
- New functions `imFileLoadImageFrame` and `imFileLoadBitmapFrame` to reuse the image data when loading.
- New function `imFileImageSave`.
- New function `imImageCreateBased`.
- New `imProcessInsert`.
- New compression functions `imCompressDataLZF` and `imCompressDataUnLZF`, using libLZF.
- New module for `imBinFile`, **IM_FILEHANDLE** that allows to access an already opened file using the system file handle as file name. Thanks to Frederico Abraham.
- Changed in JPEG file format YcbCr are now automatically converted to RGB when loaded. RGB images were already automatically converted to YCbCr when saved. Now this behavior can be controlled by the **AutoYCbCr** attribute.
- Changed the `imAnalyzeFindRegions` to include an additional parameter that control if regions touching the border are computed or not. The function `imProcessPrune` now will only eliminate the regions in the selected size range.
- Changed third party libraries, updated to newest versions: libExif, libTIFF, libPNG and zlib. Added OLD JPEG support in libTIFF.
- Changed optimization flags to ON when building the library in all

platforms.

- Changed **imProcessPerimeterLine**, **imAnalyzeMeasurePerimeter**, **imAnalyzeMeasurePerimArea**, **imAnalyzeMeasureCentroid** and **imAnalyzeMeasurePrincipalAxis** to consider pixels that touch the borders.
- Changed macro name **cdPutBitmap** to **imPutBitmap**.
- Changed function names **imImageLoad** and **imImageLoadBitmap**, to **imFileImageLoad** and **imFileImageLoadBitmap**.
- Fixed overflow in **imCalcImageStatistics** fo **IM_INT** and **IM_USHORT** images.
- Fixed error management in system file I/O in UNIX.
- Fixed some small defines for 64-bits compatibility in **libExif**, **libPNG** and **libJPEG**.
- Fixed incorrect interpretation of 16 bit data from PNG files.
- Fixed **imFileReadImageInfo** can be called many times with the same index that will return the correct result without accessing the file again.
- Fixed small bug in sample **iupglcap**.
- Fixed bug in TIFF format read for images with multiple bands in **ExtraSamples**.
- Fixed ICON format **can_sequence** was 0.
- Fixed **imProcessMergeHSI** and **imProcessSplitHSI** documentation, and implementation for **IM_BYTE** images.
- Fixed **imProcessRangeContrastThreshold**, **imProcessLocalMaxThreshold** and **imProcessRankClosestConvolve** when processing near the border.
- Fixed invalid file permissions in UNIX when saving a new file.
- Fixed name for **imProcessLocalMaxThresEstimate**.
- Fixed **imProcessReduceBy4** for images with odd width and/or height.
- Fixed **imAttribTableSet** when replacing an attribute (thanks to Takeshi Mitsunaga).
- Fixed memory leaks in **imConvertToBitmap** and **imConvertDataType** (thanks to Takeshi Mitsunaga).
- Fixed **imProcessZeroCrossing** for the last pixel column (thanks to Takeshi Mitsunaga). Also fixed for some crossings that were lost.
- Fixed **imProcessGrayMorphConvolve** for **IM_FLOAT** images with **IM_FLOAT** kernel (thanks to Takeshi Mitsunaga).

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 **iupg1cap**.
- **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

- MOV (using QuickTime SDK and QT4Linux)
- DICOM
- TIFF Annotations, DNG, IPTC and EXIF tags
- Linux Capture (using Video4Linux)
- Use libavcodec and libavformat in Linux
- AVI using libavifile in Linux (UNIX ?)
- MPEG-2 (using MSSG?)

For the Processing library:

- Fast Gaussian convolution using symmetry
- 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
- Support for the Intel® Integrated Performance Primitives
- JPEG and TIFF Thumbnails
- Formats: FLI, DV, FPX (Flash Pix), EXR (Industrial Light & Magic High Dynamic Range Format), MNG
- ECW
- GeoJasper?
- OpenML?
- WIA and TWAIN?
- Other scientific formats. FITS, VICAR, SEGY

Suggestions? im@tecgraf.puc-rio.br

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 comparison 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, Open Source
- 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.

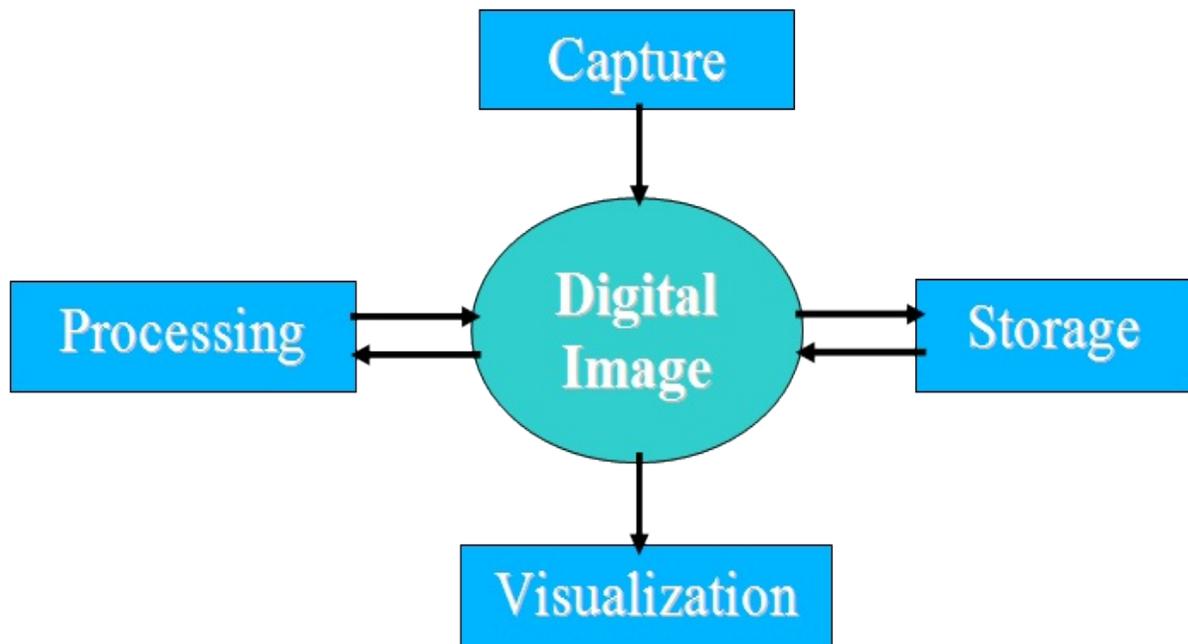
Although OpenCV has many resources, its code is very hard to reuse. The simplicity of the IM code, mainly the image processing routines, make it a good reference to be reused by other applications extracting only the code needed with little changes. And can be used as an complement to learn image processing 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 contains **Overview**, **Guide**, **Samples** and **Reference** sections for each one of the IM concepts.

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.

Building Applications

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

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

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).

The download files list includes the [Tecgraf/PUC-Rio Library Download Tips](#) document, with a description of all the available binaries.

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.

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.

In **IM**'s main source directory there is a file named *make_uname* (*make_uname.bat* in Windows) that build the libraries using **Tecmake**. To build the **IM** libraries for Windows using Visual C 7.0 for example, just execute *make_uname.bat vc7* in the source folder.

But we also provide a stand alone makefile for Linux systems and a Visual Studio workspace with the respective projects. The stand alone makefile is created using [Premake](#) and a configuration file in lua called "premake.lua".

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. Theses 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 [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 `imPutBitmap` can be used. It is defined as:

```
#define imPutBitmap(_image, _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax)
{
    if (_image->color_space == IM_RGB)
    {
        if (image->has_alpha)
            cdPutImageRectRGBA(_image->width, _image->height,
                                (unsigned char*)_image->data[0],
                                (unsigned char*)_image->data[1],
                                (unsigned char*)_image->data[2],
                                (unsigned char*)_image->data[3],
                                _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax);
        else
            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->data[1],
                           _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	<->	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 [Image Representation / Conversion](#). For example:

```

imConvertPacking(image->data[0], gl_data, image->width, image->h
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,
}

```

If you just want to save your OpenGL buffer then you can use:

```

glPixelStorei(GL_PACK_ALIGNMENT, 1); /* data alignment must be 1
glReadPixels(x, y, width, height, GL_RGB, GL_UNSIGNED_BYTE, (GLV

ifile = imFileNew(filename, format, &error);
error = imFileWriteImageInfo(ifile, width, height, IM_RGB|IM_PAC
error = imFileWriteImageData(ifile, gl_data);
imFileClose(ifile);

```

You can also put glReadPixels and imFileWriteImageInfo/imFileWriteImageData inside a loop to create an animation.

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 close 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 its counter callback, now there are a few more parameters and a user data pointer. See [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 [Image Storage](#).

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.

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](#), or download it with some makefiles [im_view.zip](#).

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>.

Lua Samples

To retrieve information from an image file:

```
local ifile, error = im.FileOpen(file_name)
local format, compression, image_count = ifile:GetInfo()
local format_desc = im.FormatInfo(format)
for i = 1, image_count do
    local width, height, color_mode, data_type, error = ifile:ReadImage(i)
end
ifile:Close()
```

To edit pixels in an image and save the changes:

```
local image = im.ImageLoad(filename)

local r = image[0]
local g = image[1]
local b = image[2]

for row = 0, image:Height() - 1, 10 do
    for column = 0, image:Width() - 1, 10 do
        r[row][column] = 0
    end
end
```

```
                g[row][column] = 0
                b[row][column] = 0
            end
        end
    end

    image:Save("indexing.bmp", "BMP")
```

To render noise:

```
local image = im.ImageCreate(500, 500, im.RGB, im.BYTE) im.ProcessNoise(
image:Save("noise.bmp", "BMP")
```

To render using the CD library:

```
local image = im.ImageCreate(500, 500, im.RGB, im.BYTE)
local canvas = image:cdCreateCanvas() -- Creates a CD_IMAGERGB

cd.Activate(canvas)
cd.Clear()
cd.Font(3, 1, 24)
cd.Text(100, 100, "Test")
cd.Line(0, 0, 100, 100)
cd.KillCanvas(canvas)

image:Save("new.bmp", "BMP")
```

Check the file samples_lua5.tar.gz for several samples in Lua. For some of them you will need also the CD and the IUP libraries to compile and link the application.

Lua Binding Guide

Overview

IMLua was developed to make all functionalities of the IM library available to Lua programmers. To use the IMLua bindings, your executable must be linked with the "imlua" library, and you must call the initialization function **imlua_open** declared in the header file **imlua.h**, as seen in the example below:

in Lua5
<pre>#include <lua.h> #include <lualib.h> #include <lauxlib.h> #include <imlua.h> void main(void) { lua_State *L = lua_open(); luaopen_string(L); luaopen_math(L); luaopen_io(L); imlua_open(L); lua_dofile("myprog.lua"); lua_close(L); }</pre>

The **imlua_open()** function registers all IM functions and constants your Lua program will need. The use of the IMLua functions in Lua is generally identical to their equivalents in C. Nevertheless, there are several exceptions due to differences between the two languages. Notice that, as opposed to C, in which the flags are combined with the bitwise operator OR, in Lua the flags are added arithmetically.

The other secondary libraries also have their initialization functions declared in **imlua.h** and each one have a separate library to be linked with the application. See [IM Lua 5 Binding](#) reference.

The IMLua dynamic libraries are also compatible with the Lua "require" function.

Function Names and Definitions

In Lua, because of the name space "im" all the functions and definitions have their names prefix changed. The general rule is quite simple:

```
imXxx    -> im.Xxx
IM_XXX   -> im.XXX
imFileXXX(ifile,... -> ifile:XXX(...)
imImageXXX(image,... -> image:XXX(...
```

Garbage Collection

All the objects are garbage collected by the Lua garbage collector.

Integration with CDLua

In IMLUA there is an additional library providing simple functions to map the imImage structure to the cdBitmap structure. And some facilities to draw an image in a CD canvas. See [IM Lua 5 Binding](#) reference.

Color values and palettes can be created and used transparently in both libraries. Palettes and color values are 100% compatible between CD and IM.

Image Capture Overview

The capture support is designed for live video, it is not for passive digital cameras that 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. These are called devices.

The capture functions allows you to:

- list the available devices
- connect to a device
- configure the device
- retrieve an image

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. You can connect to more than one device at the same time.

Once connected the user can retrieve frames from the device any time. This can be done with one function call, or inside a closed loop for several frames, or inside an idle function to periodically update the screen. The user is not notified when a new frame is available, but every time the user retrieve a frame, if successful, it is a new frame, old frames are discarded when a new frame arrives.

Currently it is implemented only in Microsoft Windows.

Image Capture Guide

Using

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`.

Building

You should include the `<im_capture.h>` header and link with the "im_capture.lib" library. This library is independent of all IM libraries.

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

To compile the capture source code you will need the Direct X 9 SDK. Notice that since Direct X uses COM, `CoInitialize(NULL)` is called when the devices are enumerated.

For more information on Direct X capture, i.e. Direct Show see:

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

Image Capture Samples

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](#)

Image Capture

Detailed Description

Functions to capture images from live video devices.

See [im_capture.h](#)

Modules

[Windows Attributes Names](#)

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 imVideoCaptureDialogCount (imVideoCapture *vc)
int imVideoCaptureShowDialog (imVideoCapture *vc,
int dialog, void *parent)
const char * imVideoCaptureDialogDesc (imVideoCapture *vc,
int dialog)
int imVideoCaptureFormatCount (imVideoCapture *vc)
int imVideoCaptureGetFormat (imVideoCapture *vc, int
format, int *width, int *height, char *desc)
int imVideoCaptureSetFormat (imVideoCapture *vc, int
format)
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.

```
im.VideoCaptureDeviceCount() -> count: number [in Lua 5]
```

const char* imVideoCaptureDeviceDesc (int *device*)

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

```
im.VideoCaptureDeviceDesc(device: number) -> desc: string [in Lua 5]
```

int imVideoCaptureReloadDevices (void)

Reload the device list. The devices can be dynamically removed or added to the system. Returns the number of available devices.

```
im.imVideoCaptureReloadDevices() -> count: number [in Lua 5]
```

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.

```
im.VideoCaptureCreate() -> vc: imVideoCapture [in Lua 5]
```

void imVideoCaptureDestroy (imVideoCapture * *vc*)

Destroys a imVideoCapture object.

```
im.VideoCaptureDestroy(vc: imVideoCapture) [in Lua 5]
```

```
vc:Destroy() [in Lua 5]
```

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

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 connected it will disconnect first.

Use -1 to return the current connected device, in this case returns -1 if not connected.

Returns zero if failed.

```
vc:Connect([device: number]) -> ret: number [in Lua 5]
```

void imVideoCaptureDisconnect (imVideoCapture *	<i>vc</i>)
--	-------------------------	--------------------

Disconnect from a capture device.

```
vc:Disconnect() [in Lua 5]
```

int imVideoCaptureDialogCount (imVideoCapture *	<i>vc</i>)
--	-------------------------	--------------------

Returns the number of available configuration dialogs.

```
vc:DialogCount() -> count: number [in Lua 5]
```

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.
dialog can be from 0 to **imVideoCaptureDialogCount**.
Returns zero if failed.

```
vc:ShowDialog(dialog: number, parent: userdata) -> error: number [:
```

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

Returns the description of a configuration dialog. dialog can be from 0 to **imVideoCaptureDialogCount**.

```
vc:DialogDesc(dialog: number) -> desc: string [in Lua 5]
```

```
int imVideoCaptureFormatCount ( imVideoCapture * vc )
```

Returns the number of available video formats.
Returns zero if failed.

```
vc:FormatCount() -> error: number [in Lua 5]
```

int imVideoCaptureGetFormat	(imVideoCapture * <i>vc</i>,
	int <i>format</i>,
	int * <i>width</i>,
	int * <i>height</i>,
	char * <i>desc</i>
)

Returns information about the video format.

format can be from 0 to [imVideoCaptureFormatCount](#).

desc should be of size 10.

The image size is usually the maximum size for that format. Other sizes [imVideoCaptureSetImageSize](#).

Returns zero if failed.

```
vc:GetFormat(format: number) -> error: number, width: number, height: number
```

int imVideoCaptureSetFormat (imVideoCapture *	<i>vc,</i>
	int	<i>format</i>
)		

Changes the video format of the connected device.

Should NOT work for DV devices. Use [imVideoCaptureSetImageSize](#) only.

Use -1 to return the current format, in this case returns -1 if failed.

When the format is changed in the dialog, for some formats the returned format is the preferred format, not the current format.

This will not affect color_mode of the capture image.

Returns zero if failed.

```
vc:SetFormat(format: number) -> error: number [in Lua 5]
```

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

Returns the current image size of the connected device.

width and height returns 0 if not connected.

```
vc:GetImageSize() -> width: number, height: number [in Lua 5]
```

int imVideoCaptureSetImageSize (imVideoCapture *	<i>vc,</i>

	int	<i>width,</i>
	int	<i>height</i>
)	

Changes the image size of the connected device.

Similar to [imVideoCaptureSetFormat](#), but changes only the size.

Valid sizes can be obtained with [imVideoCaptureGetFormat](#).

Returns zero if failed.

```
vc:SetImageSize(width: number, height: number) -> error: number [in Lua]
```

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

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.

```
vc:Frame(image: imImage, timeout: number) -> error: number [in Lua]
```

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

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.

```
vc:OneFrame(image: imImage) -> error: number [in Lua 5]
```

int imVideoCaptureLive (imVideoCapture *	<i>vc,</i>
	int	<i>live</i>
)		

Start capturing.

Use -1 to return the current state.

Returns zero if failed.

```
vc:Live(live: number) -> error: number [in Lua 5]
```

int imVideoCaptureResetAttribute (imVideoCapture *	<i>vc,</i>
	const char *	<i>attrib,</i>
	int	<i>fauto</i>
)		

Resets a camera or video attribute to the default value or to the automatic mode.
Not all attributes support automatic modes.

Returns zero if failed.

```
vc:ResetAttribute(attrib: string, fauto: number) -> error: number [in Lua 5]
```

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

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

```
vc:GetAttribute(attrib: string) -> error: number, percent: number [in Lua 5]
```

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 val
Returns zero if failed.

```
vc:SetAttribute(attrib: string, percent: number) -> error: number |
```

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

Returns a list of the description of the valid attributes.

```
vc:GetAttributeList() -> attrib_list: table of strings [in Lua 5]
```

Windows Attributes Names

[Image Capture]

VideoBrightness - Specifies the brightness, also called the black level.

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 temperature.

VideoBacklightCompensation - Specifies the backlight compensation.

VideoGain - Specifies the gain adjustment.

CameraPanAngle - Specifies the camera's pan angle. To 100 rotate right.

CameraTiltAngle - Specifies the camera's tilt angle. To 100 rotate up.

CameraRollAngle - Specifies the camera's roll angle. To 100 rotate clockwise.

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 distance in centimeters.

FlipHorizontal - Specifies the video will be flipped in the horizontal.

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 Overview

Width and Height

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.

Color Space

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. It can have a maximum of 256 colors. Each value is an index into a RGB palette.

IM_GRAY usually means luma (nonlinear Luminance), but it can represent any other intensity value that is not necessarily related to color.

IM_BINARY is a subset of the IM_GRAY color space, and it has only 2 colors black and white. Each value can be 0 or 1. But for practical reasons we use one byte to store it.

The other color spaces are standard CIE color spaces, except CMYK that does not have a clear definition without other parameters to complement it.

Data Type

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 (waste space but keep processing simple).

Color Mode Flags

To avoid defining another image parameter we also use a parameter called **color_mode** that it is composed by the **color_space** plus some **flags**, i.e. **color_mode = color_space + flags**. The flags are binary combined with the color space, for example `color_mode = IM_RGB | IM_XXX`. And several flags can be combined in the same `color_mode`.

There are 3 flags:

- IM_ALPHA
- IM_PACKED
- IM_TOPDOWN

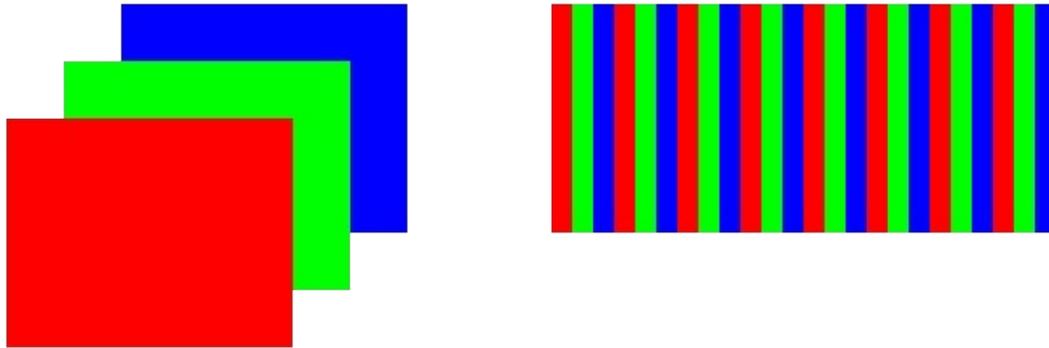
When a flag is absent the opposite definition is assumed. For simplicity we define some macros that help handling the color mode:

- `imColorModeSpace`
- `imColorModeHasAlpha`
- `imColorModeIsPacked`
- `imColorModeIsTopDown`

Color Components Packaging (**IM_PACKED** or **unpacked**)

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 `rgbrgrgb...`) 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 users define their own image structure that can have a packed or an separated organization. The following picture illustrates the difference between the two options:



(flag not defined)

IM_PACKED

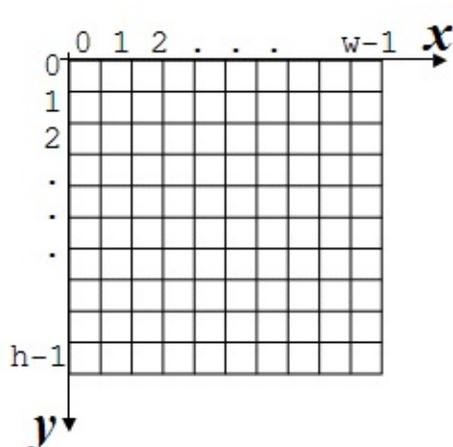
Separated and Packed RGB Components

Alpha Channel (`IM_ALPHA` or no alpha)

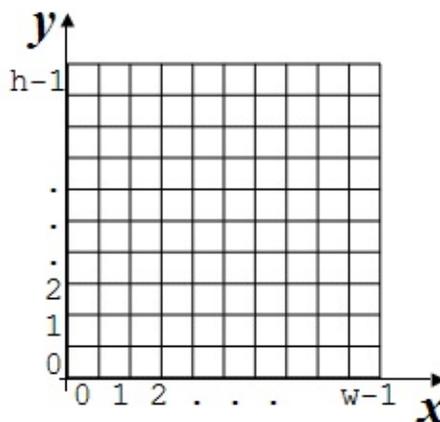
An extra component, the **alpha** channel, may be present. The number of components is then increased by one. Its organization follows the rules of packed and unpacked components.

Orientation (`IM_TOPDOWN` or bottom up)

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.



IM_TOPDOWN



(flag not defined)

Top Down and Bottom Up Orientations

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

Raw Data Buffer

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 following the specified packing.

For example, if a RGB image is 4x4 pixels it will have the following organization in memory:

RRRRRRRRRRRRRRRRGGGGGGGGGGGGGGGGBBBBBBBBBBBBBBBB - for non packed
 0 1 2 3 0 1 2 3 0 1 2 3

RGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGB - for packed cc
0 1 2 3

In bold we visually marked some lines of data.

imImage

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 buffers.

For the high level image processing functions we created a structure called **imImage** that eliminates the extra flags and assume bottom up orientation and separated components. Alpha channel is supported as an extra component.

The **imImage** structure is defined using 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**.

Image Representation Guide

Raw Data Buffer

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

```
int width, height, color_mode, data_type;
int size = imImageDataSize(width, height, color_mode, data_type)
void* buffer = malloc(size);
```

So if the data type is IM_FLOAT, we could write:

```
float* idata = (float*)buffer;
```

Then to locate the pixel at line y, column x, component d simply write:

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

But notice that this code will return values at different pixel locations for top down and bottom up orientations.

imImage

To use the **imImage** structure you must include the <im_image.h> header.

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

```
int width, height, color_space, data_type, palette_count;
long *palette;
void* buffer

imImage* image;

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

The `imImageInit` function allow you to initialize an **imImage** structure with an user allocated buffer. This is very useful if 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 do `data[0] = NULL` before calling the destroy function then the raw data buffer will not be destroyed.

The **imImage** data buffer is allocated like the raw data buffer.

The separated color components are arranged one after another, but we access the data through an array of pointers each one starting at the beginning of each color component. So `image->data[0]` contains a pointer to all the data, and `image->data[1]` is a short cut to the second component and so on. With this you can use `image->data[0]` as a starting point for all the data, or use it as the first component.

```
count = width*height;
unsigned char* idata = (unsigned char*)image->data[0];
for (int i = 0; i < count; i++)
{
    idata[i] = 255;
}
```

or

```
for (int d = 0; d < image->depth; d++)
{
    unsigned char* idata = (unsigned char*)image->data[d];

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            int offset = y * width + x;

            idata[offset] = 255;
        }
    }
}
```

The **imImage** structure contains all the image information obtained from a

file, because it also has support for alpha, attributes and 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 into the graphics display. For Bitmap images the color space must be `IM_RGB`, `IM_MAP`, `IM_GRAY` or `IM_BINARY`, and the data type must be `IM_BYTE`.

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

Image Representation 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_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 about IUP see <http://www.tecgraf.puc-rio.br/iup> and more about CD see <http://www.tecgraf.puc-rio.br/cd>.

You can view the source code here: [im_view.c](#), or download it with some makefiles [im_view.zip](#).

Image Representation

Detailed Description

See [im.h](#)

Modules

Raw Data Conversion Utilities

imImage

Raw Data Utilities

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](#).

Enumerator:

`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
00026     IM_FLOAT,  /**< "float". 4 bytes single precision IEEE floa
00027     IM_CFLOAT /**< complex "float". 2 float values in sequenc
00028 };
```

enum `imColorSpace`

Image color mode color space descriptors (first byte).
See also [Color Mode Utilities](#).

Enumerator:

`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 int
`IM_BINARY` Indexed by 2 colors: black (0) and white (1) (`data_type`
`IM_CMYK` Cyan, 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, near
IM_LUV CIE L*u*v*. L* is Lightness (nonlinear Luminance, near
IM_XYZ CIE XYZ. Linear Light Tristimulus, Y is linear Luminance

```

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,     /**< Cyan, Magenta, Yellow and Black (nonlinear)
00040     IM_YCBCR,   /**< ITU-R 601 Y'CbCr. Y' is luma (nonlinear Lur
00041     IM_LAB,      /**< CIE L*a*b*. L* is Lightness (nonlinear Lum:
00042     IM_LUV,     /**< CIE L*u*v*. L* is Lightness (nonlinear Lum:
00043     IM_XYZ      /**< CIE XYZ. Linear Light Tristimulus, Y is lin
00044 };
  
```

enum *imColorModeConfig*

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

See also [Color Mode Utilities](#).

Enumerator:

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

```

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

imImage

[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 NULL before calling imImageDestroy.

See [im_image.h](#)

Modules

[Image Conversion](#)

Data Structures

struct [_imlImage](#)

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

Defines

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

Typedefs

```
typedef \_imlImage imlImage
```

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)

imlImage * **imlImageCreateBased** (**imlImage** *image, int width, int height, int color_space, int data_type)

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

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

Define Documentation

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

Value:

```
{  
    if (_image->color_space == IM_RGB)  
    {  
        if (image->has_alpha)  
            cdPutImageRectRGBA(_image->width, _image->height,  
                                (unsigned char*)_image->data[0],  
                                (unsigned char*)_image->data[1],  
                                (unsigned char*)_image->data[2],  
                                (unsigned char*)_image->data[3],  
                                _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax);  
        else  
            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->palette,  
                           _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax);  
}
```

Utility macro to draw the image in a CD library canvas. Works only for data

IM_BYTE, and color spaces: IM_RGB, IM_MAP, IMGGRAY and IM_BINA

Typedef Documentation

```
typedef struct _iImage iImage
```

iImage Structure Definition.

An image representation than supports all the color spaces, but planes are always unpacked and the orientation is always bottom up.

Function Documentation

imlImage* imlImageCreate (int	<i>width</i> ,
	int	<i>height</i> ,
	int	<i>color_space</i> ,
	int	<i>data_type</i>
)	

Creates a new image.

```
im.ImageCreate(width: number, height: number, color_space: number,
```

imlImage* imlImageInit (int	<i>width</i> ,
	int	<i>height</i> ,
	int	<i>color_space</i> ,
	int	<i>data_type</i> ,
	void *	<i>data_buffer</i> ,
	long *	<i>palette</i> ,
	int	<i>palette_count</i>
)	

Initializes the image structure but does not allocates image data.

imlImage* imlImageCreateBased (imlImage *	<i>image</i> ,
	int	<i>width</i> ,
	int	<i>height</i> ,
	int	<i>color_space</i> ,
	int	<i>data_type</i>
)	

Creates a new image based on an existing one.

If the additional parameters are -1, the given image parameters are used.

The image attributes always are copied.

```
im.ImageCreateBased(image: imImage, [width: number], [height: number])
```

The additional parameters in Lua should be nil, and they can also be further

```
void imImageDestroy ( imImage * image )
```

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

```
im.ImageDestroy(image: imImage) [in Lua 5]
```

```
image:Destroy() [in Lua 5]
```

```
void imImageAddAlpha ( imImage * image )
```

Adds an alpha channel plane.

```
image:AddAlpha() [in Lua 5]
```

```
void imImageReshape ( imImage * image,  
int width,  
int height  
)
```

Changes the buffer size. Reallocate internal buffers if the new size is larger than the original.

```
image:Reshape(width: number, height: number) [in Lua 5]
```

```
void imImageCopy ( const imImage * src_image,  
imImage * dst_image
```

)

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

```
image:Copy(dst_image: imImage) [in Lua 5]
```

void imImageCopyData (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image</i>
)		

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

```
image:CopyData(dst_image: imImage) [in Lua 5]
```

imImage* imImageDuplicate (const imImage * *image*)

Creates a copy of the image.

```
image:Duplicate() -> new_image: imImage [in Lua 5]
```

imImage* imImageClone (const imImage * *image*)

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

```
image:Clone() -> new_image: imImage [in Lua 5]
```

void imImageSetAttribute (imImage *	<i>image,</i>
	const char *	<i>attrib,</i>
	int	<i>data_type,</i>
	int	<i>count,</i>
	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.

```
image:SetAttribute(attrib: string, data_type: number, data: table)
```

If data_type is IM_BYTE, as_string can be used as data.

const void*	imlImageGetAttribute (const <i>imlImage</i> *	<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.

```
image:GetAttribute(attrib: string, [as_string: boolean]) -> data: t
```

If data_type is IM_BYTE, as_string can be used to return a string instead

void	imlImageGetAttributeList (const <i>imlImage</i> *	<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.

```
image:GetAttributeList() -> data: table of strings [in Lua 5]
```

```
void imImageClear ( imImage * image )
```

Sets all image data to zero.

```
image:Clear() [in Lua 5]
```

```
int imImageIsBitmap ( const imImage * image )
```

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.

```
image:IsBitmap() -> is_bitmap: boolean [in Lua 5]
```

```
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.

```
image:SetPalette(palette: imPalette) [in Lua 5]
```

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

Copies the image attributes from src to dst.

```
image:CopyAttributes(dst_image: imImage) [in Lua 5]
```

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

```
const imImage * image2
```

```
)
```

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

```
image:MatchSize(image2: imImage) -> match: boolean [in Lua 5]
```

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

```
const imImage * image2
```

```
)
```

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

```
image:MatchColor(image2: imImage) -> match: boolean [in Lua 5]
```

```
int imImageMatchDataType ( const imImage * image1,
```

```
const imImage * image2
```

```
)
```

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

```
image:MatchDataType(image2: imImage) -> match: boolean [in Lua 5]
```

```
int imImageMatchColorSpace ( const imImage * image1,
```

```
const imImage * image2
```

```
)
```

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

```
image:MatchColorSpace(image2: imImage) -> match: boolean [in Lua 5]
```

```
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.

```
image:Match(image2: imImage) -> match: boolean [in Lua 5]
```

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

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

```
image:SetBinary() [in Lua 5]
```

```
void imImageMakeBinary ( imImage * image )
```

Changes a gray data into a binary data, done in-place.

```
image:MakeBinary() [in Lua 5]
```

Image Conversion

[imlImage]

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 absolute, 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 absolute, int cast_mode)

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

Enumerator:

`IM_CAST_MINMAX` scan for min and max values

`IM_CAST_FIXED` use predefined 0-max values, see [Color Manip](#)

`IM_CAST_DIRECT` direct type cast the value. Only byte and usho

```
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

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

Changes the image data type, using a complex2real conversion, a gamma correction, or a cast. When demoting the data type the function will scan for min/max values. 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.

```
im.ConvertDataType(src_image: imImage, dst_image: imImage, cpx2real: float, gamma: float, abssolute: int, cast_mode: int) -> err
```

int	imConvertColorSpace (const imImage *	src_image ,
		imImage *	dst_image
)		

Converts one color space to another. Images must be of the same size and color mode. 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 white. RGB to Map uses the median cut implementation from the free IJG JPEG library. Thomas G. Lane.

All other color space conversions assume sRGB and CIE definitions.

Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.

```
im.ConvertColorSpace(src_image: imImage, dst_image: imImage) -> err
```

int	imConvertToBitmap (const imImage *	<i>src_image,</i>
		imImage *	<i>dst_image,</i>
		int	<i>cpx2real,</i>
		float	<i>gamma,</i>
		int	<i>abssolute,</i>
		int	<i>cast_mode</i>
)		

Converts the image to its bitmap equivalent, uses **imConvertColorSpace**
Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.

```
im.ConvertToBitmap(src_image: imImage, dst_image: imImage, cpx2real:
```

Raw Data Utilities

[Image Representation]

Detailed Description

See [im_util.h](#)

Functions

int **imImageDataSize** (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 imImageDataSize ( int width,  
                     int height,  
                     int color_mode,  
                     int data_type  
                     )
```

Returns the size of the data buffer.

```
im.ImageDataSize(width: number, height: number, color_mode: number,
```

```
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 comp

```
im.ImageLineSize(width: number, color_mode: number, data_type: numl
```

```
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 comp
includes only one.

```
im.ImageLineCount(width: number, color_mode: number) -> linecount:
```

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

Check if the combination color_mode+data_type is valid.

```
im.ImageCheckFormat(color_mode: number, data_type: number) -> check
```

Raw Data Conversion Utilities

[Image Representation]

Detailed Description

Utilities for raw data buffers.

See [im_convert.h](#)

Functions

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)

Function Documentation

void imConvertPacking (const void *	<i>src_data,</i>
	void *	<i>dst_data,</i>
	int	<i>width,</i>
	int	<i>height,</i>
	int	<i>depth,</i>
	int	<i>data_type,</i>
	int	<i>src_is_packed</i>
)		

Changes the packing of the data buffer.

void imConvertMapToRGB (unsigned char *	<i>data,</i>
	int	<i>count,</i>
	int	<i>depth,</i>
	int	<i>packed,</i>
	long *	<i>palette,</i>
	int	<i>palette_count</i>
)		

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.

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

Functions

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

Define Documentation

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

Returns the color space of the color mode.

```
im.ColorModeSpace(color_mode: number) -> color_space: number [in L]
```

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

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

```
im.ColorModeMatch(color_mode1: number, color_mode2: number) -> match: boolean
```

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

Check if the color mode has an alpha channel.

```
im.ColorModeHasAlpha(color_mode: number) -> has_alpha: boolean [in L]
```

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

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

```
im.ColorModeIsPacked(color_mode: number) -> is_packed: boolean [in L]
```

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

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

```
im.ColorModeIsTopDown(color_mode: number) -> is_top_down: boolean [in L]
```

Function Documentation

const char* imColorModeSpaceName (int *color_mode*)

Returns the color mode name.

```
im.ColorModeSpaceName(color_mode: number) -> name: string [in Lua 5]
```

int imColorModeDepth (int *color_mode*)

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

```
im.ColorModeDepth(color_mode: number) -> depth: number [in Lua 5]
```

int imColorModeToBitmap (int *color_mode*)

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

```
im.ColorModeToBitmap(color_mode: number) -> color_space: number [in Lua 5]
```

**int imColorModelsBitmap (int *color_mode*,
int *data_type*
)**

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

```
im.ColorModeIsBitmap(color_mode: number, data_type: number) -> is_bitmap: boolean [in Lua 5]
```

Image Storage Overview

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.

The abstract model we use has the following structure:

Format Identifier
Compression
Image Count
Image Information: parameters, attributes, palette
Image Data
Image Information: parameters, attributes, palette
Image Data
...

The compression is usually the same for all the images in the file, but it can be changed after loading an image. For traditional file formats image count is always 1. Image information must always be loaded or saved before image data.

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.

Image Storage Guide

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.

Remember that even if all the images in the file have the same parameters you still have to call `imFileReadImageInfo` before calling `imFileReadImageData`.

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

```
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 loads 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 easier:

```

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

imImage* image = imFileLoadImage(ifile, 0, &error);

// or use imFileLoadBitmap to force a bitmap conversion

imFileClose(ifile);

```

Or the simplest version:

```

imImage* image = imFileImageLoad(file_name, 0, &error);

```

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 easier:

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

```
error = imFileSaveImage(ifile, image);
```

```
imFileClose(ifile);
```

Or the simplest version:

```
error = imFileImageSave(file_name, format, image);
```

Error Messages

Here is a sample error message display using IUP and IM error codes:

```
static void imIupErrorMessage(int error, int interactive)
{
    char* lang = IupGetLanguage();
    char *msg, *title;
    if (strcmp(lang, "ENGLISH")==0)
    {
        title = "Error";
        switch (error)
        {
            case IM_ERR_OPEN:
                msg = "Error Opening File.";
                break;
            case IM_ERR_MEM:
                msg = "Insuficient memory.";
                break;
            case IM_ERR_ACCESS:
                msg = "Error Accessing File.";
                break;
            case IM_ERR_DATA:
                msg = "Image type not Suported.";
                break;
            case IM_ERR_FORMAT:
                msg = "Invalid Format.";
                break;
            case IM_ERR_COMPRESS:
                msg = "Invalid or unsupported compression.";
                break;
            default:
                msg = "Unknown Error.";
        }
    }
    else
    {
```

```

    title = "Erro";
    switch (error)
    {
    case IM_ERR_OPEN:
        msg = "Erro Abrindo Arquivo.";
        break;
    case IM_ERR_MEM:
        msg = "Memória Insuficiente.";
        break;
    case IM_ERR_ACCESS:
        msg = "Erro Acessando Arquivo.";
        break;
    case IM_ERR_DATA:
        msg = "Tipo de Imagem não Suportado.";
        break;
    case IM_ERR_FORMAT:
        msg = "Formato Inválido.";
        break;
    case IM_ERR_COMPRESS:
        msg = "Compressão Inválida ou não Suportada.";
        break;
    default:
        msg = "Erro Desconhecido.";
    }
}

if (interactive)
    IupMessage(title, msg);
else
    printf("%s: %s", title, msg);
}

```

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 some times 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.

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 [File Format SDK](#).

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.

Image Storage 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](#)

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 Photographics Experts Group.
- "DEFLATE" - LZ77 variation (ZIP)

Modules

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

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]`".

```
im.FormatList() -> format_list: table of strings [in Lua 5]
```

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.

```
im.FormatInfo(format: string) -> error: number, desc: string, ext:
```

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. Y
color_mode and data_type are optional, use -1 to ignore them.

If you use them they will select only the allowed compressions checked I
Returns an error code.

```
im.FormatCompressions(format: string, [color_mode: number], [data_t
```

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 support the given image class at the given compress
Returns an error code.

```
im.FormatCanWriteImage(format: string, compression: string, color_r
```

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 beginning 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, use 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.

```
im.FileOpenRaw(file_name: string) -> ifile: imFile, error: number |
```

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

Creates a RAW image file.

```
im.FileNewRaw(file_name: string) -> ifile: imFile, error: number [:
```

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:

TransparencyIndex IM_BYTE (1 or N)

Comments:

If the user specifies an alpha channel, the AND mask is loaded but the file color mode will not contain the IM_ALPHA flag.

For IM_MAP images, if the user does not specifies an alpha channel the TransparencyIndex is used to initialize the AND mask when and the most repeated index with transparency will be the transparency index.

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 0.6.12.

<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:

AutoYCbCr IM_INT (1) (controls YCbCr auto conversion) default
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
jdhuf.c - added support for OJPEG_SUPPORT in libTIFF.
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
fixed small bug in "mnote-pentax-tag.h".

Comments:

Other APPx markers are ignored.
No thumbnail support.
RGB images are automatically converted to YCbCr when saved.
Also YcbCr are converted to RGB when loaded. Use AutoYCbCr=0 t

KRN - IM Kernel File Format

[File Formats]

Description

Textual format to provide 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 1.2.8.

<http://www.libpng.org>

Copyright (C) 1998-2004 Glenn Randers-Pehrson

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's.
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
Changed pngconf.h to use int instead of long in png_uint_32 an

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

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 kepted the copyright until Aldus was aquired 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 3.7.4

<http://www.remotesensing.org/libtiff/>

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 YCBCR]

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)

ADOBE_DEFLATE - Adobe LZ77 variation

SGILOG - SGI Log Luminance RLE for L and Luv (only byte, ushor

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 (

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, LZW compression is enabled. LZW Copy
libGeoTIFF can be used without XTIFF initialization. Use Handl

Changes:

"tiff_jpeg.c" - commented "downsampled_output = TRUE" in 2 pla
"tiff_ojpeg.c" for boolean type compilation.
New file "tif_config.h" to match our needs.
New file "tiff_binfile.c" that implement I/O routines using imB

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

Detailed Description

Description

Windows Copyright Microsoft Corporation.

Access to the AVI format uses Windows AVIFile library. Available in Windows Only.

When writing a new file you must use an ".avi" extension, or the Windows API will fail.

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 "-lvfw32". Old versions of Cygwin and MingW use the "-lvfw_ms32" and "-lvfw_avi32".

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 fourcc 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 2400]

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 1.701.0.

<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 we write always as JP2

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 jas_stream_create and jas_strea

Changed jp2_dec.c and jpc_cs.c to remove "uint" and "ulong" us

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.

Access to the WMV format uses Windows Media SDK. Available in 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-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 Managemen
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 previou
If not indexed by frame, random positioning may not be precise
Sequential 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

Image Storage

Detailed Description

See [im.h](#)

Modules

[File Format SDK](#)
[Image Storage](#)
[File Formats](#)

Data Structures

class [imlImageFile](#)

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

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)

Enumeration Type Documentation

enum `imErrorCodes`

File Access Error Codes

Enumerator:

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

```
imFile* imFileOpen ( const char * file_name,  
                    int * error  
                    )
```

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

```
im.FileOpen(file_name: string) -> ifile: imFile, error: number [in
```

```
imFile* imFileNew ( const char * file_name,  
                   const char * format,  
                   int * error  
                   )
```

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.

```
im.FileNew(file_name: string, format: string) -> ifile: imFile, error: number [in
```

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

Closes the file

```
im.FileClose(ifile: imFile) [in Lua 5]
```

```
ifile:Close() [in Lua 5]
```

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

Returns the internal handle. It is file format dependent.

```
ifile:Handle() -> handle: userdata [in Lua 5]
```

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

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

```
ifile:GetInfo() -> format: string, compression: string, image_count
```

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 writing.

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.

```
ifile:SetInfo(compression: string) [in Lua 5]
```

void imFileSetAttribute (imFile *	<i>ifile,</i>
	const char *	<i>attrib,</i>
	int	<i>data_type,</i>
	int	<i>count,</i>
	const void *	<i>data</i>
)		

```
)
```

Changes an extended attribute.
The data will be internally duplicated.
If data is NULL the attribute is removed.

```
ifile:SetAttribute(attrib: string, data_type: number, data: table)
```

If data_type is IM_BYTE, as_string can be used as data.

```
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.

```
ifile:GetAttribute(attrib: string, [as_string: boolean]) -> data: table
```

If data_type is IM_BYTE, as_string can be used to return a string instead

```
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.

```
ifile:GetAttributeList() -> data: table of strings [in Lua 5]
```

```
void imFileGetPalette ( imFile * ifile,
```

	long *	<i>palette,</i>
	int *	<i>palette_count</i>
)		

Returns the palette if any.

"palette" must be a 256 colors allocated array.

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

```
ifile:GetPalette() -> palette: imPalette [in Lua 5]
```

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

Changes the palette.

"palette_count" is >0 and <=256.

```
ifile:SetPalette(palette: imPalette) [in Lua 5]
```

int imFileReadImageInfo (imFile *	<i>ifile,</i>
	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

Returns an error code. index specifies the image number between 0 and

Some drivers reads only in sequence, so "index" can be ignored by the f

Any parameters can be NULL. This function must be called at least once

```
ifile:ReadImageInfo([index: number]) -> error: number, width: number
```

Default index is 0.

int	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. 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

```
ifile:WriteImageInfo(width: number, height: number, user_color_mode: number)
```

int	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: 0-255, except integer values that min-max are already between 0-255. Color mode flags contains packed, alpha and top-bottom information. If flag is -1 the file original flags are used.

Returns an error code.

```
ifile:ReadImageData(data: userdata, convert2bitmap: number, color_mode_flags: number)
```

```
int imFileWriteImageData ( imFile * ifile,  
                           void * data  
                           )
```

Writes the image data.
Returns an error code.

```
ifile:WriteImageData(data: userdata) -> error: number [in Lua 5]
```

imlImage Storage

[Image Storage]

Detailed Description

Functions to simplify the process of reading and writing imImage structures. Will also load and save the alpha planes when possible.

See [im_image.h](#)

Functions

imImage * **imFileLoadImage** (**imFile** *ifile, int index, int *error)
void **imFileLoadImageFrame** (**imFile** *ifile, int index, **imImage** *image, int *error)

imImage * **imFileLoadBitmap** (**imFile** *ifile, int index, int *error)
void **imFileLoadBitmapFrame** (**imFile** *ifile, int index, **imImage** *image, int *error)

int **imFileSaveImage** (**imFile** *ifile, const **imImage** *image)

imImage * **imFileImageLoad** (const char *file_name, int index, int *error)

imImage * **imFileImageLoadBitmap** (const char *file_name, int index, int *error)

int **imFileImageSave** (const char *file_name, const char *format, const **imImage** *image)

Function Documentation

imImage*	imFileLoadImage (imFile *	<i>ifile,</i>
		int	<i>index,</i>
		int *	<i>error</i>
)		

Loads an image from an already open file. Returns NULL if failed. This will call **imFileReadImageInfo** and **imFileReadImageData**. *index* specifies the image number between 0 and *image_count*-1. The returned image will be of the same *color_space* and *data_type* of the image in the file.

```
ifile:ImageLoad([index: number]) -> image: imImage, error: number
```

Default index is 0.

void	imFileLoadImageFrame (imFile *	<i>ifile,</i>
		int	<i>index,</i>
		imImage *	<i>image,</i>
		int *	<i>error</i>
)		

Loads an image from an already open file. Returns NULL if failed. This function assumes that the image in the file has the same parameter as the *image*. This will call **imFileReadImageInfo** and **imFileReadImageData**. *index* specifies the image number between 0 and *image_count*-1. The returned image will be of the same *color_space* and *data_type* of the image in the file.

```
ifile:ImageLoadFrame([index: number], image: imImage) -> error: number
```

Default index is 0.

imImage*	imFile *	<i>ifile,</i>
	int	<i>index,</i>
	int *	<i>error</i>
)	

Loads an image from an already open file, but forces the image to be a k
The returned imagem will be always a Bitmap image, with color_space F
GRAY or BINARY, and data_type IM_BYTE.
index specifies the image number between 0 and image_count-1.
Returns NULL if failed.

```
ifile:LoadBitmap([index: number]) -> image: imImage, error: number
```

Default index is 0.

void imFileLoadBitmapFrame (imFile *	<i>ifile,</i>
	int	<i>index,</i>
	imImage *	<i>image,</i>
	int *	<i>error</i>
)	

Loads an image from an already open file, but forces the image to be a k
This function assumes that the image in the file has the same parameter
image.
The imagem must be a Bitmap image, with color_space RGB, MAP, GR,
and data_type IM_BYTE.
index specifies the image number between 0 and image_count-1.
Returns NULL if failed.

```
ifile:LoadBitmapFrame([index: number], image: imImage) -> error: nu
```

Default index is 0.

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

Saves the image to an already open file.
 This will call **imFileWriteImageInfo** and **imFileWriteImageData**.
 Returns error code.

```
ifile:SaveImage(image: imImage) -> error: number [in Lua 5]
```

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

Loads an image from file. Open, loads and closes the file.
 index specifies the image number between 0 and image_count-1.
 Returns NULL if failed.

```
im.FileImageLoad(file_name: string, [index: number]) -> image: imIr
```

Default index is 0.

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

Loads an image from file, but forces the image to be a bitmap. Open, load
 index specifies the image number between 0 and image_count-1.
 Returns NULL if failed.

```
im.FileImageLoadBitmap(file_name: string, [index: number]) -> image
```

Default index is 0.

```
int imFileImageSave ( const char * file_name,  
                     const char * format,  
                     const imImage * image  
                     )
```

Saves the image to file. Open, saves and closes the file.
Returns error code.

```
im.FileImageLoadBitmap(file_name: string, format: string, image: im
```

File Format SDK

[Image Storage]

Detailed Description

All the file formats are based on these 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 Overview

We use the simplest model possible, a function with input data, output data and control parameters.

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.

There is no ROI (Region Of Interest) management, but you can Crop, Process, then Insert the result in the original image.

Image Processing Guide

Using

You should include one or more headers: `<im_process_ana.h>`, `<im_process_glo.h>`, `<im_process_loc.h>` and `<im_process_pon.h>`. And you must link with the "im_process.a/im_process.lib" library.

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 remember that you can always use the `imImageInit` function to initialize an **imImage** structure with your own buffer.

New Operations

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

If it is only one, then 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], total_count);
        break;
    case IM_USHORT:
        DoProc((imushort*)src_image->data[0], (imushort*)dst_image->data[0], total_count);
        break;
    case IM_INT:
        DoProc((int*)src_image->data[0], (int*)dst_image->data[0], total_count);
        break;
    case IM_FLOAT:
        DoProc((float*)src_image->data[0], (float*)dst_image->data[0], total_count);
        break;
    }
}

```

```

    case IM_CFLOAT:
        DoProc((imcfloat*)src_image->data[0], (imcfloat*)dst_image->
            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.

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 [Utilities / Counter](#).

Image Processing Samples

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](#)

Hough Lines

The Hough transform can be used to detect lines in an image. But it results are highly dependent on other operations done before and after the transform. Here you can see a small pseudo code that illustrates a possible sequence of operations to detect lines using the hough transform.

First the canny operator will isolate the borders, the threshold will mark the candidate pixels. After the transform the local maximum are isolated to detect the line parameters of the lines that have many pixels from the cadidate ones. The last operation will just draw the detected lines over the original gray scale image.

```
imProcessCanny(in, out, stddev)
imProcessHysteresisThreshold(in, out, low, high)

imProcessHoughLines(in, out)
imProcessLocalMaxThreshold(in, out, size, min)

imProcessHoughLinesDraw(in1, in2, out)
```

Or a more complete sequence using another approach:

```

gray = imImageCreate(width, height, IM_GRAY, IM_BYTE);
binary = imImageCreate(width, height, IM_BINARY, IM_BYTE);
binary2 = imImageClone(binary);

rmax = sqrt(width*width +height*height)/2;
hough_height=2*rmax+1;
hough = imImageCreate(180, hough_height, IM_GRAY, IM_INT);
hough_binary = imImageCreate(180, hough_height, IM_BINARY, IM_BY

imConvertColorSpace(rgb, gray);
imProcessPercentThreshold(gray, binary, percent=50);
imProcessBinMorphClose(binary, binary2, 3, 1);
imProcessPrune(binary2, binary, 4, size=100, 0);
imProcessFillHoles(binary, binary2, 4);
imProcessPerimeterLine(binary2, binary); // here you should hav

imProcessHoughLines(binary, hough);
imProcessLocalMaxThreshold(hough, hough_binary, 7, 100);

imProcessHoughLinesDraw(gray,hough_binary,draw_hough); // option

```

In the result of `imProcessLocalMaxThreshold` there will be several white pixels. They represent the detected lines. Defining:

$$Y = a * X + b$$

$$\cos(\theta) * X + \sin(\theta) * Y = \rho$$

The transform return values in the intervals:

```

theta = 0 .. 179
rho = -rhomax .. rhomax (origin in the center of the image)

```

where:

```

rhomax = sqrt(width*width + height*height) /2 (width and hei

```

For each (x_i, y_i) point found in the result image:

```

theta = xi;
rho = yi - rhomax;

```

then:

```

a = -cos(theta)/sin(theta);
b = (rho + (width/2)*cos(theta) + (height/2)*sin(theta))/sin(the

```

The complex formula for "b" came from the fact that we have to shift the result to the image origin at (0,0).

Image Analysis

The following pseudo code illustrates the sequence of operations to measure regions. This is also called Blob Analysis.

First the regions are isolated from background using a threshold. Then regions too small or too large are eliminated and the holes are filled in this example. After the regions are found we can start measuring properties of the regions like area and perimeter.

```
imProcessSliceThreshold(in, out, level1, level2)
imProcessPrune(in, out, connect, size1, size2)
imProcessFillHoles(in, out, connect)
imAnalyzeFindRegions(in, out, connect)
imAnalyzeMeasureArea(in, area)
imAnalyzeMeasurePerimeter(in, perim)
```

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.

Modules

Image Statistics Calculations

Image Analysis

Other Domain Transform Operations

Fourier 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

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_CFLOAT.

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** (imlImage *image, imRenderFunc render_func, char *render_name, float *param, int plus)

int **imProcessRenderCondOp** (imlImage *image, imRenderCondFunc render_cond_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** (imlImage *image, float *value)

int **imProcessRenderWheel** (imlImage *image, int internal_radius, int external_radius)

int **imProcessRenderCone** (imlImage *image, int radius)

int **imProcessRenderTent** (imlImage *image, int tent_width, int tent_height)

int **imProcessRenderRamp** (imlImage *image, int start, int end, int vert_dir)

int **imProcessRenderBox** (imlImage *image, int box_width, int box_height)

int **imProcessRenderSinc** (imlImage *image, float x_period, float y_period)

int **imProcessRenderGaussian** (imlImage *image, float stddev)

int **imProcessRenderLapOfGaussian** (imlImage *image, float stddev)

int **imProcessRenderCosine** (imlImage *image, float x_period, float y_period)

int **imProcessRenderGrid** (imlImage *image, int x_space, int y_space)

int **imProcessRenderChessboard** (imlImage *image, int x_space, int

y_space)



Typedef Documentation

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

Render Funtion.

```
render_func(x: number, y: number, d: number, param: table of number)
```

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

Render Conditional Funtion.

```
render_cond_func(x: number, y: number, d: number, param: table of r
```

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
Returns zero if the counter aborted.

```
im.ProcessRenderOp(image: imImage, render_func: function, render_name: string, plus: bool)
```

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

Render a synthetic image using a conditional render function.

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

Returns zero if the counter aborted.

```
im.ProcessRenderCondOp(image: imImage, render_cond_func: function, render_name: string, param: float)
```

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.

```
im.ProcessRenderAddSpeckleNoise(src_image: imImage, dst_image: imIr
```

```
im.ProcessRenderAddSpeckleNoiseNew(src_image: imImage, percent: nur
```

```
int imProcessRenderAddGaussianNoise ( const imImage * src_ima
                                     imImage *          dst_ima
                                     float              mean,
                                     float              stddev
                                     )
```

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

```
im.ProcessRenderAddGaussianNoise(src_image: imImage, dst_image: im.
```

```
im.ProcessRenderAddGaussianNoiseNew(src_image: imImage, mean: numbe
```

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

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

```
im.ProcessRenderAddUniformNoise(src_image: imImage, dst_image: imIr
```

```
im.ProcessRenderAddUniformNoiseNew(src_image: imImage, mean: numbe
```

```
int imProcessRenderRandomNoise ( imImage * image )
```

Render random noise.

```
im.ProcessRenderRandomNoise(image: imImage) -> counter: boolean [ir
```

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

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

```
im.ProcessRenderConstant(image: imImage, value: table of number) ->
```

```
int imProcessRenderWheel ( imImage * image,
                           int internal_radius,
                           int external_radius
                           )
```

Render a centered wheel.

```
im.ProcessRenderWheel(image: imImage, internal_radius: number, external_radius: number) ->
```

```
int imProcessRenderCone ( imImage * image,
                          int radius
                          )
```

Render a centered cone.

```
im.ProcessRenderCone(image: imImage, radius: number) -> counter: number
```

```
int imProcessRenderTent ( imImage * image,
                          int tent_width,
                          int tent_height
                          )
```

Render a centered tent.

```
im.ProcessRenderTent(image: imImage, tent_width: number, tent_height: number)
```

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

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

```
im.ProcessRenderRamp(image: imImage, start: number, end: number, vert_dir: int)
```

int imProcessRenderBox (imImage *	<i>image,</i>
	int	<i>box_width,</i>
	int	<i>box_height</i>
)		

Render a centered box.

```
im.ProcessRenderBox(image: imImage, box_width: number, box_height: number)
```

int imProcessRenderSinc (imImage *	<i>image,</i>
	float	<i>x_period,</i>
	float	<i>y_period</i>
)		

Render a centered sinc.

```
im.ProcessRenderSinc(image: imImage, x_period: number, y_period: number)
```

int imProcessRenderGaussian (imImage *	<i>image,</i>
	float	<i>stddev</i>
)		

```
    )
```

Render a centered gaussian.

```
im.ProcessRenderGaussian(image: imImage, stddev: number) -> counter
```

```
int imProcessRenderLapOfGaussian ( imImage * image,  
float stddev  
)
```

Render the laplacian of a centered gaussian.

```
im.ProcessRenderLapOfGaussian(image: imImage, stddev: number) -> counter
```

```
int imProcessRenderCosine ( imImage * image,  
float x_period,  
float y_period  
)
```

Render a centered cosine.

```
im.ProcessRenderCosine(image: imImage, x_period: number, y_period: number)
```

```
int imProcessRenderGrid ( imImage * image,  
int x_space,  
int y_space  
)
```

Render a centered grid.

```
im.ProcessRenderGrid(image: imImage, x_space: number, y_space: number)
```

```
int imProcessRenderChessboard ( imImage * image,
```

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

Render a centered chessboard.

```
im.ProcessRenderChessboard(image: imImage, x_space: number, y_space
```

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 imProcessInsert (const imlImage *src_image, const imlImage
    *region_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 <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>,
	int	<i>order</i>
)		

Only reduce the image size using the given decimation order.
Supported decimation orders:

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

```
im.ProcessReduce(src_image: imImage, dst_image: imImage, order: number)
```

```
im.ProcessReduceNew(image: imImage, order: number) -> counter: bool
```

int imProcessResize (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>,
	int	<i>order</i>
)		

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.

```
im.ProcessResize(src_image: imImage, dst_image: imImage, order: number)
```

```
im.ProcessResizeNew(image: imImage, order: number) -> counter: bool
```

void imProcessReduceBy4 (const <i>imImage</i> *	<i>src_image,</i>
	<i>imImage</i> *	<i>dst_image</i>
)	

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

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

```
im.ProcessReduceBy4(src_image: imImage, dst_image: imImage) [in Lua]
```

```
im.ProcessReduceBy4New(image: imImage) -> new_image: imImage [in Lua]
```

void imProcessCrop (const <i>imImage</i> *	<i>src_image,</i>
	<i>imImage</i> *	<i>dst_image,</i>
	int	<i>xmin,</i>
	int	<i>ymin</i>
)	

Extract a rectangular region from an image.

Images must be of the same type. Destination image size must be smaller than source image. *xmin* and *ymin* must be >0 and <size.

```
im.ProcessCrop(src_image: imImage, dst_image: imImage, xmin: number, ymin: number) [in Lua]
```

```
im.ProcessCropNew(image: imImage, xmin: number, xmax: number, ymin: number, ymax: number) [in Lua]
```

void imProcessInsert (const <i>imImage</i> *	<i>src_image,</i>
	const <i>imImage</i> *	<i>region_image,</i>
	<i>imImage</i> *	<i>dst_image,</i>
	int	<i>xmin,</i>
	int	<i>ymin</i>
)	

Insert a rectangular region in an image.

Images must be of the same type. Region image size can be larger than ymin and xmin must be >0 and <size.

Source and destiny must be of the same size. Can be done in place.

```
im.ProcessInsert(src_image: imImage, region_image: imImage, dst_image: imImage)
```

```
im.ProcessInsertNew(image: imImage, region_image: imImage, xmin: number, ymin: number)
```

void imProcessAddMargins (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>xmin,</i>
	int	<i>ymin</i>
)	

Increase the image size by adding pixels with zero value.

Images must be of the same type. Destiny image size must be greater than

```
im.ProcessAddMargins(src_image: imImage, dst_image: imImage, xmin: number, xmax: number, ymin: number, ymax: number)
```

```
im.ProcessAddMarginsNew(image: imImage, xmin: number, xmax: number, ymin: number, ymax: number)
```

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 imlImage *src_image, imlImage
        *dst_image, double cos0, double sin0, int order)
void imProcessRotate90 (const imlImage *src_image, imlImage
    *dst_image, int dir_clockwise)
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)
```

Function Documentation

void imProcessCalcRotateSize (int	<i>width,</i>
	int	<i>height,</i>
	int *	<i>new_width,</i>
	int *	<i>new_height,</i>
	double	<i>cos0,</i>
	double	<i>sin0</i>
)		

Calculates the size of the new image after rotation.

```
im.ProcessCalcRotateSize(width: number, height: number, cos0: number,
```

int imProcessRotate (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	double	<i>cos0,</i>
	double	<i>sin0,</i>
	int	<i>order</i>
)		

Rotates the image using the given interpolation order (see [imProcessRotate](#)). Images must be of the same type. The destiny size can be calculated using [imProcessCalcRotateSize](#). Returns zero if the counter aborted.

```
im.ProcessRotate(src_image: imImage, dst_image: imImage, cos0: number,
```

```
im.ProcessRotateNew(image: imImage, cos0: number, sin0: number, order: int,
```

void imProcessRotate90 (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>

```

) int dir_clockwise
)

```

Rotate the image in 90 degrees counterclockwise or clockwise. Swap columns and rows. Images must be of the same type. Destination width and height must be source width and height. Direction can be clockwise (1) or counter clockwise (-1).

```
im.ProcessRotate90(src_image: imImage, dst_image: imImage, dir_clockwise: int)
```

```
im.ProcessRotate90New(image: imImage, dir_clockwise: boolean) -> new_image: imImage
```

```

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.

```
im.ProcessRotate180(src_image: imImage, dst_image: imImage) [in Lua 5.1]
```

```
im.ProcessRotate180New(image: imImage) -> new_image: imImage [in Lua 5.1]
```

```

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.

```
im.ProcessMirror(src_image: imImage, dst_image: imImage) [in Lua 5.1]
```

```
im.ProcessMirrorNew(image: imImage) -> new_image: imImage [in Lua 5.1]
```

```

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

```

```
)
```

Apply a vertical flip. Swap lines.
Images must be of the same type and size.

```
im.ProcessFlip(src_image: imImage, dst_image: imImage) [in Lua 5]
```

```
im.ProcessFlipNew(image: imImage) -> new_image: imImage [in Lua 5]
```

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

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

```
im.ProcessRadial(src_image: imImage, dst_image: imImage, k1: number
```

```
im.ProcessRadialNew(image: imImage, k1: number, order: number) -> (
```

Additional Image Quantization Operations [Image Processing]

Detailed Description

Additionally operations to the [imConvertColorSpace](#) function.

See [im_process_pon.h](#)

Functions

void **imProcessQuantizeRGBUniform** (const **imlImage** *src_image,
imlImage *dst_image, int do_dither)

void **imProcessQuantizeGrayUniform** (const **imlImage** *src_image,
imlImage *dst_image, int grays)

Function Documentation

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

Converts a RGB image to a MAP image using uniform quantization with image must have data type IM_BYTE.

```
im.ProcessQuantizeRGBUniform(src_image: imImage, dst_image: imImage)  
im.ProcessQuantizeRGBUniformNew(src_image: imImage, do_dither: bool)
```

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

Quantizes a gray scale image in less that 256 grays using uniform quant Both images must be IM_BYTE/IM_GRAY. Can be done in place.

```
im.ProcessQuantizeGrayUniform(src_image: imImage, dst_image: imImage)  
im.ProcessQuantizeGrayUniformNew(src_image: imImage, grays: number)
```

Color Processing Operations

[Image Processing]

Detailed Description

Operations to change the color components configuration.

See [im_process_pon.h](#)

Functions

```
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_image, imlImage *dst_image)
void imProcessSplitComponents (const imlImage *src_image,
    imlImage **dst_image_list)
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)
```

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_GRAY/IM_BYTE and luma image is IM_GRAY/IM_BYTE and chroma is IM_RGB/IM_BYTE. Source and destiny must have the same size.

```
im.ProcessSplitYChroma(src_image: imImage, y_image: imImage, chroma:  
im.ProcessSplitYChromaNew(src_image: imImage) -> y_image: imImage,
```

```
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,IM_FLOAT. Destiny images a Source images must normalized to 0-1 if type is IM_FLOAT ([imProcessCoordinate System Conversions](#) for a definition of the color conversion) Source and destiny must have the same size.

```
im.ProcessSplitHSI(src_image: imImage, h_image: imImage, s_image: :  
im.ProcessSplitHSINew(src_image: imImage) -> h_image: imImage, s_ir
```

```
void imProcessMergeHSI ( const imlImage * h_image,
```

	const <i>imImage</i> *	<i>s_image</i>,
	const <i>imImage</i> *	<i>i_image</i>,
	<i>imImage</i> *	<i>dst_image</i>
)	

Merge HSI planes into a RGB image.

Source images must be IM_GRAY/IM_FLOAT. Destiny image can be IM_...
Source and destiny must have the same size. See [HSI Color Coordinate](#)
color conversion.

```
im.ProcessMergeHSI(h_image: imImage, s_image: imImage, i_image: imImage, dst_image: imImage)
```

```
im.ProcessMergeHSINew(h_image: imImage, s_image: imImage, i_image: imImage, dst_image: imImage)
```

void imProcessSplitComponents (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> **	<i>dst_image_list</i>
)		

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.

```
im.ProcessSplitComponents(src_image: imImage, dst_image_list: table of imImage)
```

```
im.ProcessSplitComponentsNew(src_image: imImage) -> dst_image_list
```

void imProcessMergeComponents (const <i>imImage</i> **	<i>src_image_list</i>,
	<i>imImage</i> *	<i>dst_image</i>
)		

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.

```
im.ProcessMergeComponents(src_image_list: table of imImage, dst_image: imImage)
```

```
im.ProcessMergeComponentsNew(src_image_list: table of imImage) -> c
```

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

Normalize the color components by their sum. Example: $c1 = c1/(c1+c2)$
Destiny image must be IM_FLOAT.

```
im.ProcessNormalizeComponents(src_image: imImage, dst_image: imImage)
```

```
im.ProcessNormalizeComponentsNew(src_image: imImage) -> new_image:
```

```
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_CFLOAT.

```
im.ProcessReplaceColor(src_image: imImage, dst_image: imImage, src_color: table of float, dst_color: table of float)
```

```
im.ProcessReplaceColorNew(src_image: imImage, src_color: table of float, dst_color: table of float)
```

Histogram Based Operations

[Image Processing]

Detailed Description

See [im_process_pon.h](#)

Functions

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

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

Function Documentation

void imProcessExpandHistogram (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>,
	float	<i>percent</i>
)		

Performs an histogram expansion.

Percentage defines an amount of pixels to include at start and end. If its histogram will be considered.

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

```
im.ProcessExpandHistogram(src_image: imImage, dst_image: imImage, percent: number)
```

```
im.ProcessExpandHistogramNew(src_image: imImage, percent: number) -> new_image: imImage
```

void imProcessEqualizeHistogram (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>
)		

Performs an histogram equalization.

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

```
im.ProcessEqualizeHistogram(src_image: imImage, dst_image: imImage)
```

```
im.ProcessEqualizeHistogramNew(src_image: imImage) -> new_image: imImage
```

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 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_level)
void imProcessThreshold (const imlImage *src_image, imlImage
    *dst_image, int level, int value)
void imProcessThresholdByDiff (const imlImage *src_image1, const
    imlImage *src_image2, imlImage *dst_image)
void imProcessHysteresisThreshold (const imlImage *src_image,
    imlImage *dst_image, int low_thres, int high_thres)
void imProcessHysteresisThresEstimate (const imlImage *image, int
    *low_level, int *high_level)
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 imProcessLocalMaxThresEstimate (const imlImage *image, int
    *level)
void imProcessSliceThreshold (const imlImage *src_image, imlImage
    *dst_image, int start_level, int end_level)
```

Function Documentation

int imProcessRangeContrastThreshold (const <i>imlImage</i> *	<i>src_image</i>
	<i>imlImage</i> *	<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 destination.
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.

`im.ProcessRangeContrastThreshold(src_image: imImage, dst_image: imImage)`

`im.ProcessRangeContrastThresholdNew(image: imImage, kernel_size: int, min_range: int)`

int imProcessLocalMaxThreshold (const <i>imlImage</i> *	<i>src_image</i>,
	<i>imlImage</i> *	<i>dst_image</i>,
	int	<i>kernel_size</i>,
	int	<i>min_level</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 destination.

```
im.ProcessLocalMaxThreshold(src_image: imImage, dst_image: imImage,
im.ProcessLocalMaxThresholdNew(image: imImage, kernel_size: number,
```

void imProcessThreshold (const <i>imImage</i> *	<i>src_image,</i>
	<i>imImage</i> *	<i>dst_image,</i>
	int	<i>level,</i>
	int	<i>value</i>
)		

Apply a manual threshold.

threshold = a <= level ? 0: value

Normal value is 1 but another common value is 255. Can be done in place
Supports all integer IM_GRAY images as source, and IM_BINARY as de

```
im.ProcessThreshold(src_image: imImage, dst_image: imImage, level:
```

```
im.ProcessThresholdNew(src_image: imImage, level: number, value: nu
```

void imProcessThresholdByDiff (const <i>imImage</i> *	<i>src_image1,</i>
	const <i>imImage</i> *	<i>src_image2,</i>
	<i>imImage</i> *	<i>dst_image</i>
)		

Apply a threshold by the difference of two images.

threshold = a1 <= a2 ? 0: 1

Can be done in place.

```
im.ProcessThresholdByDiff(src_image1: imImage, src_image2: imImage,
```

```
im.ProcessThresholdByDiffNew(src_image1: imImage, src_image2: imIma
```

void imProcessHysteresisThreshold (const <i>imImage</i> *	<i>src_image,</i>
	<i>imImage</i> *	<i>dst_image,</i>

	int	<i>low_thres,</i>
	int	<i>high_thres</i>
)	

Apply a threshold by the Hysteresis method.

Hysteresis thresholding of edge pixels. Starting at pixels with a value greater than the LOW threshold and then testing pixels that have a value greater than the LOW threshold.

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

```
im.ProcessHysteresisThreshold(src_image: imImage, dst_image: imImage)
```

```
im.ProcessHysteresisThresholdNew(src_image: imImage, low_thres: num)
```

void imProcessHysteresisThresEstimate (const imImage *	<i>image,</i>
	int *	<i>low_level,</i>
	int *	<i>high_level,</i>
)	

Estimates hysteresis low and high threshold levels.

Useful for [imProcessHysteresisThreshold](#).

```
im.ProcessHysteresisThresEstimate(image: imImage) -> low_level: num
```

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

Calculates the threshold level for manual threshold using an uniform error

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.

```
im.ProcessUniformErrThreshold(src_image: imImage, dst_image: imImage)
```

```
im.ProcessUniformErrThresholdNew(src_image: imImage) -> level: number
```

void imProcessDifusionErrThreshold (const <i>imImage</i> *	<i>src_image,</i>
	<i>imImage</i> *	<i>dst_image,</i>
	int	<i>level</i>
)		

Apply a dithering on each image channel by using a difusion error method. It can be applied on any IM_BYTE images. It will "threshold" each channel must be of the same depth.

```
im.ProcessDifusionErrThreshold(src_image: imImage, dst_image: imImage)
```

```
im.ProcessDifusionErrThresholdNew(src_image: imImage, level: number)
```

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

Calculates the threshold level for manual threshold using a percentage calculation. Returns the used level.

```
im.ProcessPercentThreshold(src_image: imImage, dst_image: imImage,
```

```
im.ProcessPercentThresholdNew(src_image: imImage, percent: number)
```

int imProcessOtsuThreshold (const <i>imImage</i> *	<i>src_image,</i>
	<i>imImage</i> *	<i>dst_image</i>

```
)
```

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

Original implementation by Flavio Szenberg.

```
im.ProcessOtsuThreshold(src_image: imImage, dst_image: imImage) ->
```

```
im.ProcessOtsuThresholdNew(src_image: imImage) -> level: number, ne
```

```
int imProcessMinMaxThreshold ( const imImage * src_image,  
                               imImage * 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 de

```
im.ProcessMinMaxThreshold(src_image: imImage, dst_image: imImage) .
```

```
im.ProcessMinMaxThresholdNew(src_image: imImage) -> level: number,
```

```
void imProcessLocalMaxThresEstimate ( const imImage * image,  
                                       int * level  
                                       )
```

Estimates Local Max threshold level for IM_BYTE images.

```
im.ProcessLocalMaxThresEstimate(image: imImage) -> level: number [:
```

```
void imProcessSliceThreshold ( const imImage * src_image,  
                               imImage * dst_image,  
                               int start_level,  
                               int end_level  
                               )
```

Apply a manual threshold using an interval.

$\text{threshold} = \text{start_level} \leq a \leq \text{end_level} ? 1 : 0$

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

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

```
im.ProcessSliceThreshold(src_image: imImage, dst_image: imImage, start_level: number, end_level: number)
```

```
im.ProcessSliceThresholdNew(src_image: imImage, start_level: number, end_level: number)
```

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_CPXNORM  
}  
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 do_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 exceptio

Enumerator:

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

```
00029 {
00030     IM_UN_EQL,    /**< 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_CPXNORM /**< complex normalization by magnitude = a / cpxi
```

```
00043 };
```

enum imBinaryOp

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

Enumerator:

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

```
00056 {
00057     IM_BIN_ADD,    /**< add          =    a+b          */
00058     IM_BIN_SUB,    /**< subtract       =    a-b          */
00059     IM_BIN_MUL,    /**< multiply      =    a*b          */
00060     IM_BIN_DIV,    /**< divide        =    a/b          (#) */
00061     IM_BIN_DIFF,   /**< difference    =    |a-b|         */
00062     IM_BIN_POW,    /**< power         =    a^b          */
00063     IM_BIN_MIN,    /**< minimum       =    (a < b)? a: b */
00064     IM_BIN_MAX,    /**< maximum       =    (a > b)? a: b */
00065 };
```

Function Documentation

void imProcessUnArithmeticOp (const <i>imlImage</i> *	<i>src_image</i>,
	<i>imlImage</i> *	<i>dst_image</i>,
	int	<i>op</i>
)		

Apply an arithmetic unary operation.

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

```
im.ProcessUnArithmeticOp(src_image: imImage, dst_image: imImage, op
```

```
im.ProcessUnArithmeticOpNew(image: imImage, op: number) -> new_image
```

void imProcessArithmeticOp (const <i>imlImage</i> *	<i>src_image1</i>,
	const <i>imlImage</i> *	<i>src_image2</i>,
	<i>imlImage</i> *	<i>dst_image</i>,
	int	<i>op</i>
)		

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 de

- byte -> byte, ushort, int, float
- ushort -> ushort, int, float
- int -> int, float
- float -> float
- complex -> complex One exception is that you can combine comple

```
im.ProcessArithmeticOp(src_image1: imImage, src_image2: imImage, dst
```

```
im.ProcessArithmeticOpNew(image1: imImage, image2: imImage, op: nur
```

The New function will create a new image of the same type of the source

void imProcessArithmeticConstOp (const <i>imImage</i> *	<i>src_image</i>,
	float	<i>src_const</i>,
	<i>imImage</i> *	<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

```
im.ProcessArithmeticConstOp(src_image: imImage, src_const: number, dst_image: imImage, op: string)  
im.ProcessArithmeticConstOpNew(image: imImage, src_const: number, op: string)
```

void imProcessBlend (const <i>imImage</i> *	<i>src_image1</i>,
	<i>imImage</i> *	<i>src_image2</i>,
	<i>imImage</i> *	<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.

```
im.ProcessBlend(src_image1: imImage, src_image2: imImage, dst_image: imImage, alpha: number)  
im.ProcessBlendNew(image1: imImage, image2: imImage, alpha: number)
```

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

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

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

```
im.ProcessSplitComplex(src_image: imImage, dst_image1: imImage, dst_image2: imImage) -> dst_image1, dst_image2
im.ProcessSplitComplexNew(image: imImage, do_polar: boolean) -> dst_image1, dst_image2
```

void imProcessMergeComplex (const <i>imlImage</i> *	<i>src_image1</i>,
	const <i>imlImage</i> *	<i>src_image2</i>,
	<i>imlImage</i> *	<i>dst_image</i>,
	int	<i>polar</i>
)		

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_CFLOAT.

```
im.ProcessMergeComplex(src_image1: imImage, src_image2: imImage, dst_image: imImage) -> dst_image
im.ProcessMergeComplexNew(image1: imImage, image2: imImage) -> new_image
```

void imProcessMultipleMean (const <i>imlImage</i> **	<i>src_image_list</i>,
	int	<i>src_image_count</i>,
	<i>imlImage</i> *	<i>dst_image</i>
)		

Calculates the mean of multiple images.

Images must match size and type.

```
im.ProcessMultipleMean(src_image_list: table of imImage, dst_image: imImage)
im.ProcessMultipleMeanNew(src_image_list: table of imImage) -> new imImage
```

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

Calculates the standard deviation of multiple images. Images must match size and type. Use **imProcessMultipleMean** to calculate the mean image.

```
im.ProcessMultipleStdDev(src_image_list: table of imImage, mean_image: imImage, dst_image: imImage)
im.ProcessMultipleStdDevNew(src_image_list: table of imImage, mean_image: imImage) -> new imImage
```

int imProcessAutoCovariance (const imImage *	src_image,
	const imImage *	mean_image,
	imImage *	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.

```
im.ProcessAutoCovariance(src_image: imImage, mean_image: imImage, dst_image: imImage)
im.ProcessAutoCovarianceNew(src_image: imImage, mean_image: imImage) -> new imImage
```

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

Multiplies the conjugate of one complex image with another complex image. Images must match size. $\text{Conj}(\text{img1}) * \text{img2}$
Can be done in-place.

```
im.ProcessMultiplyConj(src_image1: imImage, src_image2: imImage, dest: imImage)
```

```
im.ProcessMultiplyConjNew(src_image1: imImage, src_image2: imImage, dest: imImage)
```

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

```
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 do_reset)
```

Enumeration Type Documentation

enum `imLogicOp`

Logical Operations.

Enumerator:

`IM_BIT_AND` and = a & b

`IM_BIT_OR` or = a | b

`IM_BIT_XOR` xor = ~(a | b)

```
00292     {
00293     IM_BIT_AND,    /**< and  =  a & b  */
00294     IM_BIT_OR,     /**< or   =  a | b  */
00295     IM_BIT_XOR    /**< xor  = ~(a | b) */
00296 };
```

Function Documentation

void imProcessBitwiseOp (const <i>imlImage</i> *	<i>src_image1,</i>
	const <i>imlImage</i> *	<i>src_image2,</i>
	<i>imlImage</i> *	<i>dst_image,</i>
	int	<i>op</i>
)		

Apply a logical operation.

Images must have data type IM_BYTE, IM_USHORT or IM_INT. Can be

```
im.ProcessBitwiseOp(src_image1: imImage, src_image2: imImage, dst_:
```

```
im.ProcessBitwiseOpNew(src_image1: imImage, src_image2: imImage, op
```

void imProcessBitwiseNot (const <i>imlImage</i> *	<i>src_image,</i>
	<i>imlImage</i> *	<i>dst_image</i>
)		

Apply a logical NOT operation.

Images must have data type IM_BYTE, IM_USHORT or IM_INT. Can be place.

```
im.ProcessBitwiseNot(src_image: imImage, dst_image: imImage) [in Lu
```

```
im.ProcessBitwiseNotNew(src_image: imImage) -> new_image: imImage |
```

void imProcessBitMask (const <i>imlImage</i> *	<i>src_image,</i>
	<i>imlImage</i> *	<i>dst_image,</i>
	unsigned char	<i>mask,</i>
	int	<i>op</i>
)		

Apply a bit mask.

The same as `imProcessBitwiseOp` but the second image is replaced by a mask. Images must have data type `IM_BYTE`. It is valid only for AND, OR and XOR.

```
im.ProcessBitMask(src_image: imImage, dst_image: imImage, mask: string)
```

```
im.ProcessBitMaskNew(src_image: imImage, mask: string, op: number)
```

In Lua, mask is a string with 0s and 1s, for example: "11001111".

void imProcessBitPlane (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	int	<i>plane,</i>
	int	<i>do_reset</i>
)	

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.

```
im.ProcessBitPlane(src_image: imImage, dst_image: imImage, plane: number)
```

```
im.ProcessBitPlaneNew(src_image: imImage, plane: number, do_reset: boolean)
```

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 **imlImage** *src_image, **imlImage** *dst_image, int op, float *param)

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

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

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

Enumeration Type Documentation

enum `imToneGamut`

Tone Gamut Operations.

Enumerator:

<code>IM_GAMUT_NORMALIZE</code>	<code>normalize = (a-min) / (max-min)</code> (desti
<code>IM_GAMUT_POW</code>	<code>pow = ((a-min) / (max-min))^gamma * param[0]=gamma</code>
<code>IM_GAMUT_LOG</code>	<code>log = log(K * (a-min) / (max-min) + 1); param[0]=K (K>0)</code>
<code>IM_GAMUT_EXP</code>	<code>exp = (exp(K * (a-min) / (max-min)) - 1 param[0]=K</code>
<code>IM_GAMUT_INVERT</code>	<code>invert = max - (a-min)</code>
<code>IM_GAMUT_ZEROSTART</code>	<code>zerostart = a - min</code>
<code>IM_GAMUT_SOLARIZE</code>	<code>solarize = a < level ? a: (level * (max-r param[0]=level percentage (0-100) rel photography solarization effect.</code>
<code>IM_GAMUT_SLICE</code>	<code>slice = start < a a > end ? min: binar param[0]=start, param[1]=end, param </code>
<code>IM_GAMUT_EXPAND</code>	<code>expand = a < start ? min: a > end ? m: param[0]=start, param[1]=end</code>
<code>IM_GAMUT_CROP</code>	<code>crop = a < start ? start: a > end ? end param[0]=start, param[1]=end</code>
<code>IM_GAMUT_BRIGHTCONT</code>	<code>brightcont = a < min ? min: a > max ? param[0]=bright_shift (-100%..+100%), change brightness and contrast simult</code>

```
00480      {
00481  IM_GAMUT_NORMALIZE, /**< normalize = (a-min) / (max-min)
00482  IM_GAMUT_POW,      /**< pow      = ((a-min) / (max-min))^
00483                                     param[0]=gamma
00484  IM_GAMUT_LOG,      /**< log      = log(K * (a-min) / (max
```

```

00485          param[0]=K      (K>0)
00486  IM_GAMUT_EXP,      /**< exp      = (exp(K * (a-min) / (ma
00487          param[0]=K
00488  IM_GAMUT_INVERT,    /**< invert    = max - (a-min)
00489  IM_GAMUT_ZEROSTART, /**< zerostart = a - min
00490  IM_GAMUT_SOLARIZE,  /**< solarize = a < level ? a: (leve
00491          param[0]=level percent
00492          photography solarizat:
00493  IM_GAMUT_SLICE,     /**< slice      = start < a || a > end ?
00494          param[0]=start, param
00495  IM_GAMUT_EXPAND,    /**< expand    = a < start ? min: a >
00496          param[0]=start, param
00497  IM_GAMUT_CROP,      /**< crop      = a < start ? start: a
00498          param[0]=start, param
00499  IM_GAMUT_BRIGHTCONT /**< brightcont = a < min ? min: a >
00500          param[0]=bright_shift
00501          change brightness and
00502 };

```

Function Documentation

void imProcessToneGamut (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>,
	int	<i>op</i>,
	float *	<i>param</i>
)		

Apply a gamut operation with arguments.

Supports all data types except IM_CFLOAT.

The linear operation do a special conversion when 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.

```
im.ProcessToneGamut(src_image: imImage, dst_image: imImage, op: number, param: table)
```

```
im.ProcessToneGamutNew(src_image: imImage, op: number, param: table) -> new_image: imImage
```

void imProcessUnNormalize (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>
)		

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

```
im.ProcessUnNormalize(src_image: imImage, dst_image: imImage) [in place]
```

```
im.ProcessUnNormalizeNew(src_image: imImage) -> new_image: imImage
```

void imProcessDirectConv (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>
)		

Directly converts IM_USHORT, IM_INT and IM_FLOAT into IM_BYTE images.
This can also be done using [imConvertDataType](#) with IM_CAST_DIRECT.

```
im.ProcessDirectConv(src_image: imImage, dst_image: imImage) [in Lua]
im.ProcessDirectConvNew(src_image: imImage) -> new_image: imImage [in Lua]
```

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

A negative effect. Uses [imProcessToneGamut](#) with IM_GAMUT_INVERSE for non MAP images.

Supports all color spaces and all data types except IM_CFLOAT.

```
im.ProcessNegative(src_image: imImage, dst_image: imImage) [in Lua]
im.ProcessNegativeNew(src_image: imImage) -> new_image: imImage [in Lua]
```

Convolution Operations

[Image Processing]

Detailed Description

See [im_process_loc.h](#)

Functions

```
int imProcessConvolve (const imImage *src_image, imImage
    *dst_image, const imImage *kernel)
int imProcessConvolveRep (const imImage *src_image, imImage
    *dst_image, const imImage *kernel, int count)
int imProcessCompassConvolve (const imImage *src_image,
    imImage *dst_image, imImage *kernel)
void imProcessRotateKernel (imImage *kernel)
int imProcessDiffOfGaussianConvolve (const imImage *src_image,
    imImage *dst_image, float stddev1, float stddev2)
int imProcessDiffOfGaussianConvolveRep (const imImage
    *src_image, imImage *dst_image, float stddev1, float stddev2)
int imProcessLapOfGaussianConvolve (const imImage *src_image,
    imImage *dst_image, float stddev)
int imProcessMeanConvolve (const imImage *src_image, imImage
    *dst_image, int kernel_size)
int imProcessGaussianConvolveRep (const imImage *src_image,
    imImage *dst_image, float stddev)
int imProcessGaussianConvolve (const imImage *src_image,
    imImage *dst_image, float stddev)
int imProcessSobelConvolve (const imImage *src_image, imImage
    *dst_image)
void imProcessZeroCrossing (const imImage *src_image, imImage
    *dst_image)
void imProcessCanny (const imImage *src_image, imImage
    *dst_image, float stddev)
int imGaussianStdDev2Repetitions (float stddev)
int imGaussianStdDev2KernelSize (float stddev)
```

	imImage *	kernel
)	

Convolve with a kernel rotating it 8 times and getting the absolute maximum. Kernel must be square.

The rotation is implemented only for kernel sizes 3x3, 5x5 and 7x7.

Supports all data types except IM_CFLOAT. Returns zero if the counter is zero. If the kernel image attribute "Description" exists it is used by the counter.

```
im.ProcessCompassConvolve(src_image: imImage, dst_image: imImage, kernel: imImage, counter: int)
```

```
im.ProcessCompassConvolveNew(image: imImage, kernel: imImage) -> counter
```

void imProcessRotateKernel (imImage * kernel)

Utility function to rotate a kernel one time.

```
im.ProcessRotateKernel(kernel: imImage) [in Lua 5]
```

int imProcessDiffOfGaussianConvolve (const imImage *	src_image
	imImage *	dst_image
	float	stddev1,
	float	stddev2
)	

Difference(Gaussian1, Gaussian2).

Supports all data types, but if source is IM_BYTE or IM_USHORT destination is IM_BYTE.

```
im.ProcessDiffOfGaussianConvolve(src_image: imImage, dst_image: imImage, stddev1: float, stddev2: float)
```

```
im.ProcessDiffOfGaussianConvolveNew(image: imImage, stddev1: float, stddev2: float)
```

int imProcessDiffOfGaussianConvolveRep (const imImage *	src_image
	imImage *	dst_image
	float	stddev

	float	<i>stddev</i>
)	

Difference(Gaussian1, Gaussian2) using gaussian repetitions.

Supports all data types, but if source is IM_BYTE or IM_USHORT destination

```
im.ProcessDiffOfGaussianConvolveRep(src_image: imImage, dst_image:
```

```
im.ProcessDiffOfGaussianConvolveRepNew(image: imImage, stddev1: num
```

int imProcessLapOfGaussianConvolve (const <i>imImage</i> *	<i>src_image</i>
	<i>imImage</i> *	<i>dst_image</i>
	float	<i>stddev</i>
)	

Convolution with a laplacian of a gaussian kernel.

Supports all data types, but if source is IM_BYTE or IM_USHORT destination

```
im.ProcessLapOfGaussianConvolve(src_image: imImage, dst_image: imImage,
```

```
im.ProcessLapOfGaussianConvolveNew(image: imImage, stddev: number)
```

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

Convolution with a kernel full of "1"s inside a circle.

Supports all data types.

```
im.ProcessMeanConvolve(src_image: imImage, dst_image: imImage, kernel
```

```
im.ProcessMeanConvolveNew(image: imImage, kernel_size: number) -> c
```

int imProcessGaussianConvolveRep (const <i>imImage</i> *	<i>src_image</i>,

	imlImage *	<i>dst_image,</i>
	float	<i>stddev</i>
)	

Convolution with a gaussian kernel. The gaussian is obtained by repetition.
Supports all data types.

```
im.ProcessGaussianConvolveRep(src_image: imImage, dst_image: imImage)
```

```
im.ProcessGaussianConvolveRepNew(image: imImage, stddev: number) ->
```

int imProcessGaussianConvolve (const imlImage *	<i>src_image,</i>
	imlImage *	<i>dst_image,</i>
	float	<i>stddev</i>
)		

Convolution with a float gaussian kernel.
Supports all data types.

```
im.ProcessGaussianConvolve(src_image: imImage, dst_image: imImage,
```

```
im.ProcessGaussianConvolveNew(image: imImage, stddev: number) -> cc
```

int imProcessSobelConvolve (const imlImage *	<i>src_image,</i>
	imlImage *	<i>dst_image</i>
)		

Magnitude of the sobel convolution.
Supports all data types except IM_CFLOAT.

```
im.ProcessSobelConvolve(src_image: imImage, dst_image: imImage) ->
```

```
im.ProcessSobelConvolveNew(image: imImage) -> counter: boolean, new
```

void imProcessZeroCrossing (const imlImage *	<i>src_image,</i>

	imImage *	<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 used after a second derivative, laplace.

Extracted from XITE, Copyright 1991, Blab, UiO

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

```
im.ProcessZeroCrossing(src_image: imImage, dst_image: imImage) [in
```

```
im.ProcessZeroCrossingNew(image: imImage) -> new_image: imImage [ir
```

void imProcessCanny (const imImage *	<i>src_image,</i>
	imImage *	<i>dst_image,</i>
	float	<i>stddev</i>
)	

First part of the Canny edge detector. Includes the gaussian filtering and 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
"Algorithms for Image Processing and Computer Vision"
WILEY
```

```
im.ProcessCanny(src_image: imImage, dst_image: imImage, stddev: nur
```

```
im.ProcessCannyNew(image: imImage, stddev: number) -> new_image: ir
```

int imGaussianStdDev2Repetitions (float	<i>stddev</i>)
---	--------------	----------------------	----------

Calculates the number of 3x3 gaussian repetitions given the standard de

```
im.GaussianStdDev2Repetitions(stddev: number) -> count: number [in
```

```
int imGaussianStdDev2KernelSize ( float stddev )
```

Calculates the kernel size given the standard deviation.

```
im.GaussianStdDev2KernelSize(stddev: number) -> kernel_size: number
```

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

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

Rank convolution using the median value.

Returns zero if the counter aborted.

Supports all data types except IM_CFLOAT. Can be applied on color image.

```
im.ProcessMedianConvolve(src_image: imImage, dst_image: imImage, kernel_size: int) -> int
```

```
im.ProcessMedianConvolveNew(image: imImage, kernel_size: number) -> int
```

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

Rank convolution using (maximum-minimum) value.

Returns zero if the counter aborted.

Supports all data types except IM_CFLOAT. Can be applied on color image.

```
im.ProcessRangeConvolve(src_image: imImage, dst_image: imImage, kernel_size: int) -> int
```

```
im.ProcessRangeConvolveNew(image: imImage, kernel_size: number) -> int
```

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

Rank convolution using the closest maximum or minimum value.

Returns zero if the counter aborted.

Supports all data types except IM_CFLOAT. Can be applied on color image.

```
im.ProcessRankClosestConvolve(src_image: imImage, dst_image: imImage, kernel_size: number)
```

```
im.ProcessRankClosestConvolveNew(image: imImage, kernel_size: number)
```

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_CFLOAT. Can be applied on color image.

```
im.ProcessRankMaxConvolve(src_image: imImage, dst_image: imImage, kernel_size: number)
```

```
im.ProcessRankMaxConvolveNew(image: imImage, kernel_size: number)
```

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_CFLOAT. Can be applied on color image.

```
im.ProcessRankMinConvolve(src_image: imImage, dst_image: imImage, kernel_size: number)
```

```
im.ProcessRankMinConvolveNew(image: imImage, kernel_size: number)
```

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

int	imProcessBinMorphConvolve (const <i>imImage</i> *	<i>src_image</i>,
		<i>imImage</i> *	<i>dst_image</i>,
		const <i>imImage</i> *	<i>kernel</i>,
		int	<i>hit_white</i>,
		int	<i>iter</i>
)		

Base binary morphology convolution.

Images are all IM_BINARY. Kernel is IM_INT. Use kernel size odd for be
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 z
Almost all the binary morphology operations use this function.

If the kernel image attribute "Description" exists it is used by the counter.

```
im.ProcessBinMorphConvolve(src_image: imImage, dst_image: imImage,
```

```
im.ProcessBinMorphConvolveNew(image: imImage, kernel: imImage, hit_
```

int	imProcessBinMorphErode (const <i>imImage</i> *	<i>src_image</i>,
		<i>imImage</i> *	<i>dst_image</i>,
		int	<i>kernel_size</i>,
		int	<i>iter</i>
)		

Binary morphology convolution with a kernel full of "1"s and hit white.

```
im.ProcessBinMorphErode(src_image: imImage, dst_image: imImage, ke
```

```
im.ProcessBinMorphErodeNew(image: imImage, kernel_size: number, ite
```

```
int imProcessBinMorphDilate ( const imlImage * src_image,
                             imlImage * dst_image,
                             int kernel_size,
                             int iter
                             )
```

Binary morphology convolution with a kernel full of "0"s and hit black.

```
im.ProcessBinMorphDilate(src_image: imImage, dst_image: imImage, ke
im.ProcessBinMorphDilateNew(image: imImage, kernel_size: number, it
```

```
int imProcessBinMorphOpen ( const imlImage * src_image,
                             imlImage * dst_image,
                             int kernel_size,
                             int iter
                             )
```

Erode+Dilate. When iteration is more than one it means Erode+Erode+E

```
im.ProcessBinMorphOpen(src_image: imImage, dst_image: imImage, kern
im.ProcessBinMorphOpenNew(image: imImage, kernel_size: number, ite
```

```
int imProcessBinMorphClose ( const imlImage * src_image,
                              imlImage * dst_image,
                              int kernel_size,
                              int iter
                              )
```

Dilate+Erode.

```
im.ProcessBinMorphClose(src_image: imImage, dst_image: imImage, ke
im.ProcessBinMorphCloseNew(image: imImage, kernel_size: number, ite
```

```

int imProcessBinMorphOutline ( const imlImage * src_image,
                               imlImage * dst_image,
                               int kernel_size,
                               int iter
                               )

```

Erode+Difference.

The difference from the source image is applied only once.

```

im.ProcessBinMorphOutline(src_image: imImage, dst_image: imImage, l
im.ProcessBinMorphOutlineNew(image: imImage, kernel_size: number, :

```

```

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

```

Thins the supplied binary image using Rosenfeld's parallel thinning algo
Reference:

"Efficient Binary Image Thinning using Neighborhood Maps"

by Joseph M. Cychosz, 3ksnn64@ecn.purdue.edu

in "Graphics Gems IV", Academic Press, 1994

```

im.ProcessBinMorphThin(src_image: imImage, dst_image: imImage) [in
im.ProcessBinMorphThinNew(image: imImage) -> new_image: imImage [in

```

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 *</code>	<code><i>src_image</i>,</code>
	<code>imlImage *</code>	<code><i>dst_image</i>,</code>
	<code>const imlImage *</code>	<code><i>kernel</i>,</code>
	<code>int</code>	<code><i>ismax</i></code>
<code>)</code>		

Base gray morphology convolution.

Supports all data types except IM_CFLOAT. Can be applied on color image. 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 the same kernel. If the kernel image attribute "Description" exists it is used by the counter.

```
im.ProcessGrayMorphConvolve(src_image: imImage, dst_image: imImage,
```

```
im.ProcessGrayMorphConvolveNew(image: imImage, kernel: imImage, isr
```

<code>int imProcessGrayMorphErode (</code>	<code>const imlImage *</code>	<code><i>src_image</i>,</code>
	<code>imlImage *</code>	<code><i>dst_image</i>,</code>
	<code>int</code>	<code><i>kernel_size</i></code>
<code>)</code>		

Gray morphology convolution with a kernel full of "0"s and use minimum

```
im.ProcessGrayMorphErode(src_image: imImage, dst_image: imImage, ke
```

```
im.ProcessGrayMorphErodeNew(image: imImage, kernel_size: number) ->
```

<code>int imProcessGrayMorphDilate (</code>	<code>const imlImage *</code>	<code><i>src_image</i>,</code>
	<code>imlImage *</code>	<code><i>dst_image</i>,</code>
	<code>int</code>	<code><i>kernel_size</i></code>

```
)
```

Gray morphology convolution with a kernel full of "0"s and use maximum

```
im.ProcessGrayMorphDilate(src_image: imImage, dst_image: imImage, kernel_size: number) ->  
im.ProcessGrayMorphDilateNew(image: imImage, kernel_size: number) ->
```

```
int imProcessGrayMorphOpen ( const imImage * src_image,  
                             imImage * dst_image,  
                             int kernel_size  
                             )
```

Erode+Dilate.

```
im.ProcessGrayMorphOpen(src_image: imImage, dst_image: imImage, kernel_size: number) ->  
im.ProcessGrayMorphOpenNew(image: imImage, kernel_size: number) ->
```

```
int imProcessGrayMorphClose ( const imImage * src_image,  
                              imImage * dst_image,  
                              int kernel_size  
                              )
```

Dilate+Erode.

```
im.ProcessGrayMorphClose(src_image: imImage, dst_image: imImage, kernel_size: number) ->  
im.ProcessGrayMorphCloseNew(image: imImage, kernel_size: number) ->
```

```
int imProcessGrayMorphTopHat ( const imImage * src_image,  
                               imImage * dst_image,  
                               int kernel_size  
                               )
```

Open+Difference.

```
im.ProcessGrayMorphTopHat(src_image: imImage, dst_image: imImage, kernel_size: number)
im.ProcessGrayMorphTopHatNew(image: imImage, kernel_size: number)
```

int	imProcessGrayMorphWell (const imImage *	src_image,
		imImage *	dst_image,
		int	kernel_size
)		

Close+Difference.

```
im.ProcessGrayMorphWell(src_image: imImage, dst_image: imImage, kernel_size: number)
im.ProcessGrayMorphWellNew(image: imImage, kernel_size: number) ->
```

int	imProcessGrayMorphGradient (const imImage *	src_image,
		imImage *	dst_image,
		int	kernel_size
)		

Difference(Erode, Dilate).

```
im.ProcessGrayMorphGradient(src_image: imImage, dst_image: imImage, kernel_size: number)
im.ProcessGrayMorphGradientNew(image: imImage, kernel_size: number)
```

Fourier Transform Operations

[Image Processing]

Detailed Description

All Fourier transforms use FFTW library version 2.1.5. Although there are newer versions, we build binaries only to version 2 because it is small and as fast as newer versions. Source code to use FFTW version 3 is available.

FFTW Copyright Matteo Frigo, Steven G. Johnson and the MIT.

<http://www.fftw.org>

See "fftw.h"

Must link with "im_fftw" library.

The FFTW lib has a GPL license. The license of the "im_fftw" 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** *image, int inverse, int center, int normalize)

void **imProcessSwapQuadrants** (**imlImage** *image, int center2origin)

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. Destination image must be of type complex.

```
im.ProcessFFT(src_image: imImage, dst_image: imImage) [in Lua 5]
```

```
im.ProcessFFTNew(image: imImage) -> new_image: imImage [in Lua 5]
```

```
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.

```
im.ProcessIFFT(src_image: imImage, dst_image: imImage) [in Lua 5]
```

```
im.ProcessIFFTNew(image: imImage) -> new_image: imImage [in Lua 5]
```

```
void imProcessFFTraw ( imImage * image,  
                      int inverse,  
                      int center,  
                      )
```

	int	<i>normalize</i>
)		

Raw in-place FFT (forward or inverse).

The lowest frequency can be centered after forward, or can be restored to original position after inverse. The result can be normalized after the transform by $\sqrt{w \cdot h}$ [1] or by $(w \cdot h)^{-1}$. Images must be of the same size and both must be of type complex.

```
im.ProcessFFTraw(image: imImage, inverse: number, center: number, r
```

void imProcessSwapQuadrants (imImage *	<i>image,</i>
	int	<i>center2origin</i>
)		

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. 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.

```
im.ProcessSwapQuadrants(image: imImage, center2origin: number) [in
```

Other Domain Transform Operations [Image Processing]

Detailed Description

Hough, Distance.

See [im_process_glo.h](#)

Functions

```
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

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

Hough Lines Transform.

It will detect white lines in a black background. So the source image must have the white lines of interest enhanced. The better the threshold with the line detection.

The destiny image must have IM_GRAY, IM_INT, width=180, height=2*radius image diagonal/2.

The hough transform defines " $\cos(\theta) * X + \sin(\theta) * Y = \rho$ " and 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/>

```
im.ProcessHoughLines(src_image: imImage, dst_image: imImage) -> counter
```

```
im.ProcessHoughLinesNew(image: imImage) -> counter: boolean, new_image
```

int imProcessHoughLinesDraw (const <i>imImage</i> *	<i>src_image</i>,
	const <i>imImage</i> *	<i>hough_points</i>,
	<i>imImage</i> *	<i>dst_image</i>
)		

Draw detected hough lines.

The source image must be IM_GRAY and IM_BYTE. The destiny image must be the source image for in place processing.

The hough points image is a hough transform image that was thresholded with a 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 c
Returns the number of detected lines.

```
im.ProcessHoughLinesDraw(src_image: imImage, hough_points: imImage,  
im.ProcessHoughLinesDrawNew(image: imImage, hough_points: imImage)
```

```
void imProcessCrossCorrelation ( const imImage * src_image1,  
const imImage * src_image2,  
imImage * dst_image  
)
```

Calculates the Cross Correlation in the frequency domain.

$CrossCorr(a,b) = IFFT(Conj(FFT(a))*FFT(b))$

Images must be of the same size and only destiny image must be of type

```
im.ProcessCrossCorrelation(src_image1: imImage, src_image2: imImage
```

```
im.ProcessCrossCorrelationNew(image1: imImage, image2: imImage) ->
```

```
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 de
must be of type complex.

```
im.ProcessAutoCorrelation(src_image: imImage, dst_image: imImage) |
```

```
im.ProcessAutoCorrelationNew(image: imImage) -> new_image: imImage
```

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

Calculates the Distance Transform of a binary image using an approximate euclidian distance.

Each white pixel in the binary image is assigned a value equal to its distance to the nearest black pixel.

Uses a two-pass algorithm incrementally calculating the distance.

Source image must be IM_BINARY, destination must be IM_FLOAT.

```
im.ProcessDistanceTransform(src_image: imImage, dst_image: imImage)
```

```
im.ProcessDistanceTransformNew(image: imImage) -> new_image: imImage
```

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

Marks all the regional maximum of the distance transform.

source is IM_GRAY/IM_FLOAT destination in IM_BINARY.

We consider maximum all connected pixel values that have smaller pixels around it.

```
im.ProcessRegionalMaximum(src_image: imImage, dst_image: imImage)
```

```
im.ProcessRegionalMaximumNew(image: imImage) -> new_image: imImage
```

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 <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>,
	int	<i>box_size</i>
)	

Generates a zoom in effect averaging colors inside a square region. Operates only on IM_BYTE images.

```
im.ProcessPixelate(src_image: imImage, dst_image: imImage, box_size: int) -> imImage  
im.ProcessPixelateNew(src_image: imImage, box_size: number) -> new_imImage
```

void imProcessPosterize (const <i>imImage</i> *	<i>src_image</i>,
	<i>imImage</i> *	<i>dst_image</i>,
	int	<i>level</i>
)	

A simple Posterize effect. It reduces the number of colors in the image to a specified number of bit planes. Can have 1 to 7 levels. See [imProcessBitMask](#). Image data type must be integer.

```
im.ProcessPosterize(src_image: imImage, dst_image: imImage, level: int) -> imImage  
im.ProcessPosterizeNew(src_image: imImage, level: number) -> new_imImage
```

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 *image)
void imCalcHistogram (const unsigned char *data, int count,
unsigned long *histo, int cumulative)
void imCalcUShortHistogram (const unsigned short *data,
int count, unsigned long *histo, int cumulative)
void imCalcGrayHistogram (const imImage *image,
unsigned long *histo, int cumulative)
void imCalcImageStatistics (const imImage *image, imStats
*stats)
void imCalcHistogramStatistics (const imImage *image,
imStats *stats)
void imCalcHistImageStatistics (const imImage *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 two images (Root Mean Square Error)

```
im.CalcRMSError(image1: imImage, image2: imImage) -> rms: number [:
```

```
float imCalcSNR ( const imImage * src_image,  
                 const imImage * noise_image  
                 )
```

Calculates the SNR of an image and its noise (Signal Noise Ratio).

```
im.CalcSNR(src_image: imImage, noise_image: imImage) -> snr: number
```

```
unsigned long imCalcCountColors ( const imImage * image )
```

Count the number of different colors in an image.

Image must be IM_BYTE, but all color spaces except IM_CMYK.

```
im.CalcCountColors(image: imImage) -> count: number [in Lua 5]
```

```
void imCalcHistogram ( const unsigned char * data,  
                      int count,  
                      unsigned long * histo,  
                      int cumulative  
                      )
```

Calculates the histogram of a IM_BYTE data.

Histogram is always 256 positions long.

When cumulative is different from zero it calculates the cumulative histogram.

```
im.CalcHistogram(image: imImage, plane: number, cumulative: number)
```

Where plane is the depth plane to calculate the histogram.

The returned table is zero indexed. image can be IM_USHORT or IM_BYTE

void imCalcUShortHistogram (const unsigned short *	<i>data,</i>
	int	<i>count,</i>
	unsigned long *	<i>histo,</i>
	int	<i>cumulative</i>
)		

Calculates the histogram of a IM_USHORT data.

Histogram is always 65535 positions long.

When cumulative is different from zero it calculates the cumulative histogram.

Use [imCalcHistogram](#) in Lua.

void imCalcGrayHistogram (const imImage *	<i>image,</i>
	unsigned long *	<i>histo,</i>
	int	<i>cumulative</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 cumulative is different from zero it calculates the cumulative histogram.

```
im.CalcGrayHistogram(image: imImage, cumulative: number) -> histo:
```

--	--	--

void imCalcImageStatistics (const <i>imImage</i> *	<i>image</i>,
	<i>imStats</i> *	<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[1]=green stats, ...

Supports all data types except IM_CFLOAT.

```
im.CalcImageStatistics(image: imImage) -> stats: table [in Lua 5]
```

Table contains the following fields: max, min, positive, negative, zeros, mean, stddev. The same as the **imStats** structure.

void imCalcHistogramStatistics (const <i>imImage</i> *	<i>image</i>,
	<i>imStats</i> *	<i>stats</i>
)		

Calculates the statistics about the image histogram data.

There is one stats for each depth plane. For ex: stats[0]=red stats, stats[1]=green stats, ...

Only IM_BYTE images are supported.

```
im.CalcHistogramStatistics(image: imImage) -> stats: table [in Lua
```

void imCalcHistImageStatistics (const <i>imImage</i> *	<i>image</i>,
	int *	<i>median</i>,
	int *	<i>mode</i>
)		

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.

```
im.CalcHistoImageStatistics(image: imImage) -> median: number, mode
```

Image Analysis

[Image Processing]

Detailed Description

See [im_process_ana.h](#)

Functions

int **imAnalyzeFindRegions** (const **imImage** *src_image, **imImage** *dst_image, int connect, int touch_border)

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 *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

int	imAnalyzeFindRegions (const <i>imImage</i> *	<i>src_image</i>,
		<i>imImage</i> *	<i>dst_image</i>,
		int	<i>connect</i>,
		int	<i>touch_border</i>
)		

Find white regions in binary image.

Result is IM_GRAY/IM_USHORT type. Regions can be 4 connected or 8

Returns the number of regions found. Background is marked as 0.

Regions touching the border are considered only if touch_border=1.

```
im.AnalyzeFindRegions(src_image: imImage, dst_image: imImage, connect
```

```
im.AnalyzeFindRegionsNew(image: imImage, connect: number, touch_bo
```

void	imAnalyzeMeasureArea (const <i>imImage</i> *	<i>image</i>,
		int *	<i>area</i>
)		

Measure the actual area of all regions. Holes are not included.

This is the number of pixels of each region.

Source image is IM_GRAY/IM_USHORT type (the result of **imAnalyzeF**
area has size the number of regions.

```
im.AnalyzeMeasureArea(image: imImage, [region_count: number]) -> a
```

The returned table is zero indexed.

void	imAnalyzeMeasurePerimArea (const <i>imImage</i> *	<i>image</i>,
		float *	<i>perimarea</i>

)

Measure the polygonal area limited by the perimeter line of all regions. Notice that some regions may have polygonal area zero.

Source image is IM_GRAY/IM_USHORT type (the result of **imAnalyzeF** perimarea has size the number of regions.

```
im.AnalyzeMeasurePerimArea(image: imImage, [region_count: number])
```

The returned table is zero indexed.

void imAnalyzeMeasureCentroid (const <i>imImage</i> *	<i>image,</i>
	const int *	<i>area,</i>
	int	<i>region_count,</i>
	float *	<i>cx,</i>
	float *	<i>cy</i>
)		

Calculate the centroid position of all regions. Holes are not included.

Source image is IM_GRAY/IM_USHORT type (the result of **imAnalyzeF** area, cx and cy have size the number of regions. If area is NULL will be

```
im.AnalyzeMeasureCentroid(image: imImage, [area: table of numbers],
```

The returned tables are zero indexed.

void imAnalyzeMeasurePrincipalAxis (const <i>imImage</i> *	<i>image,</i>
	const int *	<i>area,</i>
	const float *	<i>cx,</i>
	const float *	<i>cy,</i>
	const int	<i>region_cou</i>
	float *	<i>major_slop</i>
	float *	<i>major_leng</i>
	float *	<i>minor_slop</i>

	float *	<i>minor_leng</i>
)	

Calculate the principal major axis slope of all regions.

Source image is IM_GRAY/IM_USHORT type (the result of **imAnalyzeF** data has size the number of regions. If area or centroid are NULL will be Principal (major and minor) axes are defined to be those axes that pass

```
im.AnalyzeMeasurePrincipalAxis(image: imImage, [area: table of numl
-> major_slope: table of numbers, ma:
```

The returned tables are zero indexed.

void imAnalyzeMeasureHoles (const imImage *	<i>image,</i>
	int	<i>connect,</i>
	int *	<i>holes_count,</i>
	int *	<i>area,</i>
	float *	<i>perim</i>
)	

Measure the number and area of holes of all regions.

Source image is IM_USHORT type (the result of **imAnalyzeFindRegion** area and perim has size the number of regions, if some is NULL it will be

```
im.AnalyzeMeasureHoles(image: imImage, connect: number, [region_co
```

The returned tables are zero indexed.

void imAnalyzeMeasurePerimeter (const imImage *	<i>image,</i>
	float *	<i>perim</i>
)	

Measure the total perimeter of all regions (external and internal).

Source image is IM_GRAY/IM_USHORT type (the result of imAnalyzeFir It uses a half-pixel inter distance for 8 neighbors in a perimeter of a 4 c

region.

This function can also be used to measure line length.
perim has size the number of regions.

```
im.AnalyzeMeasurePerimeter(image: imImage) -> perim: table of number
```

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

Isolates the perimeter line of gray integer images. Background is defined black (0).

It just checks if at least one of the 4 connected neighbors is non zero. Its borders are extended with zeros.

```
im.ProcessPerimeterLine(src_image: imImage, dst_image: imImage) [ir
```

```
im.ProcessPerimeterLineNew(image: imImage) -> new_image: imImage [i
```

```
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 a binary images. Regions can be 4 connected or
Can be done in-place. end_size can be zero to ignore big objects.

```
im.ProcessPrune(src_image: imImage, dst_image: imImage, connect: nu
```

```
im.ProcessPruneNew(image: imImage, connect: number, start_size: nur
```

```
void imProcessFillHoles ( const imImage * src_image,
```

	imImage *	<i>dst_image,</i>
	int	<i>connect</i>
)	

Fill holes inside white regions.

Source and destiny are a binary images. Regions can be 4 connected or
Can be done in-place.

```
im.ProcessFillHoles(src_image: imImage, dst_image: imImage, connect
```

```
im.ProcessFillHolesNew(image: imImage, connect: number) -> new_image
```

IM Modules

Here is a list of all modules:

- **Image Capture**
 - **Windows Attributes Names**
- **Image Representation**
 - **Raw Data Conversion Utilities**
 - **imImage**
 - **Image Conversion**
 - **Raw Data Utilities**
 - **Color Mode Utilities**
- **Image Storage**
 - **File Format SDK**
 - **imImage Storage**
 - **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**

- **Other Domain Transform Operations**
- **Fourier 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**
 - **Library Management**
 - **Math Utilities**
 - **Palette Generators**
 - **String Utilities**
 - **Color Utilities**
 - **Data Type Utilities**
 - **Binary Data Utilities**
 - **Data Compression Utilities**
 - **IM Lua 5 Binding**

Utilities

Detailed Description

See [im_util.h](#)

Modules

Binary File Access
Color Manipulation
Complex Numbers
Counter
Windows DIB
Library Management
Math Utilities
Palette Generators
String Utilities
Color Utilities
Data Type Utilities
Binary Data Utilities
Data Compression Utilities
IM Lua 5 Binding

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)
```

Library Management [Utilities]

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) 1994-2005 Tecgraf/PUC-Rio  
and PETROBRAS S/A"
```

```
#define IM_VERSION "3.1.0"
```

```
#define IM_VERSION_DATE "2005/12/12"
```

```
#define IM_VERSION_NUMBER 301000
```

Functions

const char * **imVersion** (void)

const char * **imVersionDate** (void)

int **imVersionNumber** (void)

Define Documentation

```
#define IM_VERSION_NUMBER 301000
```

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.

```
im.Version() -> version: string [in Lua 5]
```

const char* imVersionDate (void)

Returns the library current version release date.

```
im.VersionDate() -> date: string [in Lua 5]
```

int imVersionNumber (void)

Returns the library current version number.

```
im.VersionNumber() -> version: number [in Lua 5]
```

IM Lua 5 Binding

[Utilities]

Detailed Description

Binding for the Lua 5 scripting language. Works with versions 5.0 and 5.1.

Lua 5.0.2 Copyright (C) 1994-2004 Tecgraf, PUC-Rio

Lua 5.1 Copyright (C) 1994-2005 Lua.org, PUC-Rio

R. Ierusalimschy, L. H. de Figueiredo & W. Celes

<http://www.lua.org>

The name of the functions were changed because of the namespace "im" and because of the object orientation.

As a general rule use:

```
imXxx    -> im.Xxx
IM_XXX   -> im.XXX
imFileXXX(ifile,... -> ifile:XXX(...)
imImageXXX(image,... -> image:XXX(...
```

All the objects are garbage collected by the Lua garbage collector.

See [imlua.h](#)

Functions

```
int implua_open (lua_State *L)
int implua_open_capture (lua_State *L)
int implua_open_process (lua_State *L)
int implua_open_fftw (lua_State *L)
int implua_open_cd (lua_State *L)
```

Function Documentation

```
int imlua_open ( lua_State * L )
```

Initializes the Lua binding of the main IM library.
You must link the application with the "imlua5" library.

```
int imlua_open_capture ( lua_State * L )
```

Initializes the Lua binding of the capture library.
You must link the application with the "imlua_capture5" library.

```
int imlua_open_process ( lua_State * L )
```

Initializes the Lua binding of the process library.
You must link the application with the "imlua_process5" library.

```
int imlua_open_fftw ( lua_State * L )
```

Initializes the Lua binding of the fourier transform library.
You must link the application with the "imlua_fftw5" library.

```
int imlua_open_cd ( lua_State * L )
```

Initializes the Lua binding of additional functions for the CD library.
CD library is a 2D graphics library.

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

You must link the application with the "imlua_cd5" library.

The following functions are defined:

`bitmap:imImageCreate()` -> `image: imImage` [in Lua 5]
Creates an `imImage` from a `cdBitmap`.

`image:cdCreateBitmap()` -> `bitmap: cdBitmap` [in Lua 5]
Creates a `cdBitmap` from an `imImage`.
The `imImage` must be a bitmap image, see `imImageIsBitmap`.

`image:cdInitBitmap()` -> `bitmap: cdBitmap` [in Lua 5]
Creates a `cdBitmap` from an `imImage`, but reuses image data.
When the `cdBitmap` is destroyed the data is preserved.

`image:cdPutImageRect(x: number, y: number, w: number, h: number, x1: number, y1: number)`
Draws the `imImage` into the active `cdCanvas`.
The `imImage` must be a bitmap image, see `imImageIsBitmap`.

`image:wdPutImageRect(x: number, y: number, w: number, h: number, x1: number, y1: number)`
Draws the `imImage` into the active `cdCanvas` using world coordinates.
The `imImage` must be a bitmap image, see `imImageIsBitmap`.

`image:cdGetImage(x: number, y: number)` [in Lua 5]
Retrieve the `imImage` data from the active `cdCanvas`.
The `imImage` must be a `IM_RGB/IM_BYTE` image.

`image:cdCreateCanvas([res: number])` -> `canvas: cdCanvas` [in Lua 5]
Creates a `cdCanvas` using the `CD_IMAGERGB` driver. Resolution is optional.
The `imImage` must be a `IM_RGB/IM_BYTE` image.

Color Utilities

[Utilities]

Detailed Description

See [im_util.h](#)

Functions

long **imColorEncode** (unsigned char red, unsigned char green,
unsigned char blue)

void **imColorDecode** (unsigned char *red, unsigned char *green,
unsigned char *blue, long color)

Function Documentation

long imColorEncode (unsigned char	<i>red,</i>
	unsigned char	<i>green,</i>
	unsigned char	<i>blue</i>
)		

Encode RGB components in a long for palette usage.
"long" definition is compatible with the CD library definition.

```
im.ColorEncode(red: number, green: number, blue: number) -> color:
```

void imColorDecode (unsigned char *	<i>red,</i>
	unsigned char *	<i>green,</i>
	unsigned char *	<i>blue,</i>
	long	<i>color</i>
)		

Decode RGB components from a long for palette usage.
"long" definition is compatible with the CD library definition.

```
im.ColorDecode(color: lightuserdata) -> red: number, green: number,
```

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

[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 purposes.
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 purposes.

```
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 quantize
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 &	<i>value,</i>
	const T &	<i>max</i>
)	[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 rec
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	<i>Y,</i>
	const T	<i>Cb,</i>
	const T	<i>Cr,</i>
	T &	<i>R,</i>
	T &	<i>G,</i>
	T &	<i>B,</i>
	const T &	<i>zero,</i>
	const T &	<i>max</i>
)	[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 + 1.402f * (Cr - zero));  
00136     float g = float(Y - 0.344f * (Cb - zero) - 0.714f * (Cr - zero));  
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 }
```

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, YCrCb 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	L,
	const float	a,
	const float	b,
	float &	X,
	float &	Y,
	float &	Z
)	[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 u1 = 0.1978f, v1 = 0.4683f;
00353     if (L != 0)
00354     {
00355         u1 = u / (6.5f * L) + 0.1978f;
00356         v1 = v / (6.5f * L) + 0.4683f;
00357     }
00358
00359     X = ((9 * u1) / (4 * v1)) * Y;
00360     Z = ((12 - 3 * u1 - 20 * v1) / (4 * v1)) * 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	<i>RL</i>,
	const float	<i>GL</i>,
	const float	<i>BL</i>,
	float &	<i>R</i>,

```

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 defined along the cube diagonal. It ranges from 0 (black) to 1 (white).

HS are the polar coordinates of a plane normal to "I".

"S" is the normal distance from the diagonal of the RGB cube. It ranges from 0 to S_{max} .

"H" is the angle starting from the red vector, given in degrees.

This is not a new color space, this is exactly the same gammut as RGB. Since it is still a cube, S_{max} 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 (here in radians) and I.

```
float imColorHSI_ImaxS ( float  h,  
                        double cosh,  
                        double sinh  
                        )
```

Returns I where S is maximum given H (here 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).

Palette Generators

[Utilities]

Detailed Description

Creates several standard palettes

In Lua, to create a palette you can call `im.PaletteCreate`.

```
im.PaletteCreate([count: number]) -> pal: imPalette [in Lua 5]
```

Default count is 256. IMLua and CDLua palettes are 100% compatible.

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 all palette entries and finds the minimum euclidian square distance. If the given color it returns immediately.

```
im.PaletteFindNearest(pal: imPalette, color: lightuserdata) -> index
```

```
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 success the exact match in the palette else search for the first color that fits in the

```
im.PaletteFindColor(pal: imPalette, color: lightuserdata, tol: number)
```

```
long* imPaletteGray ( void )
```

Creates a palette of gray scale values. The colors are arranged from black to white.

```
im.PaletteGray() -> pal: imPalette [in Lua 5]
```



long* imPaletteRed (void)

Creates a palette of a gradient of red colors. The colors are arranged from black to pure red.

```
im.PaletteRed() -> pal: imPalette [in Lua 5]
```

long* imPaletteGreen (void)

Creates a palette of a gradient of green colors. The colors are arranged from black to pure green.

```
im.PaletteGreen() -> pal: imPalette [in Lua 5]
```

long* imPaletteBlue (void)

Creates a palette of a gradient of blue colors. The colors are arranged from black to pure blue.

```
im.PaletteBlue() -> pal: imPalette [in Lua 5]
```

long* imPaletteYellow (void)

Creates a palette of a gradient of yellow colors. The colors are arranged from black to pure yellow.

```
im.PaletteYellow() -> pal: imPalette [in Lua 5]
```

long* imPaletteMagenta (void)

Creates a palette of a gradient of magenta colors. The colors are arranged from black to pure magenta.

```
im.PaletteMagenta() -> pal: imPalette [in Lua 5]
```

long* imPaletteCian (void)

Creates a palette of a gradient of cian colors. The colors are arranged from black to pure cian.

```
im.PaletteCian() -> pal: imPalette [in Lua 5]
```

long* imPaletteRainbow (void)

Creates a palette of rainbow colors. The colors are arranged in the light wave length spectrum order (starting from purple).

```
im.PaletteRainbow() -> pal: imPalette [in Lua 5]
```

long* imPaletteHues (void)

Creates a palette of hues with maximum saturation.

```
im.PaletteHues() -> pal: imPalette [in Lua 5]
```

long* imPaletteBlueIce (void)

Creates a palette of a gradient of blue colors. The colors are arranged from pure blue to white.

```
im.PaletteBlueIce() -> pal: imPalette [in Lua 5]
```

long* imPaletteHotIron (void)

Creates a palette of a gradient from black to white passing trough red and orange.

```
im.PaletteHotIron() -> pal: imPalette [in Lua 5]
```

long* imPaletteBlackBody (void)

Creates a palette of a gradient from black to white passing through red and yellow.

```
im.PaletteBlackBody() -> pal: imPalette [in Lua 5]
```

long* imPaletteHighContrast (void)

Creates a palette with high contrast colors.

```
im.PaletteHighContrast() -> pal: imPalette [in Lua 5]
```

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.

```
im.PaletteUniform() -> pal: imPalette [in Lua 5]
```

int imPaletteUniformIndex (long *color*)

Returns the index of the correspondent RGB color of an uniform palette.

```
im.PaletteUniformIndex(color: lightuserdata) -> index: number [in Lua 5]
```

int imPaletteUniformIndexHalftoned (long <i>color</i>,
int <i>x</i>,
int <i>y</i>
)

Returns the index of the correspondent RGB color of an uniform palette. the index in a halftone matrix. The spatial position used by the halftone n

```
im.PaletteUniformIndexHalftoned(color: lightuserdata, x: number, y
```

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.

Enumerator:

`IM_LITTLEENDIAN` Little Endian - The most significant byte is on the right
`IM_BIGENDIAN` Big Endian - The most significant byte is on the left

```
00212 {  
00213     IM_LITTLEENDIAN, /**< Little Endian - The most significant  
00214     IM_BIGENDIAN      /**< Big Endian - The most significant byte  
00215 };
```

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

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)

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.

```
im.DataTypeSize(data_type: number) -> size: number [in Lua 5]
```

const char* imDataTypeName (int *data_type*)

Returns the numeric data type name given its identifier.

```
im.DataTypeName(data_type: number) -> name: string [in Lua 5]
```

unsigned long imDataTypeIntMax (int *data_type*)

Returns the maximum value of an integer data type. For floating point returns 0.

```
im.DataTypeIntMax(data_type: number) -> int_max: number [in Lua 5]
```

long imDataTypeIntMin (int *data_type*)

Returns the minimum value of an integer data type. For floating point returns 0.

```
im.DataTypeIntMin(data_type: number) -> int_min: number [in Lua 5]
```

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	<i>width,</i>
	int	<i>height,</i>
	T *	<i>map,</i>
	float	<i>x1,</i>
	float	<i>y1,</i>
	float	<i>box_width,</i>
	float	<i>box_height,</i>
	TU	<i>Dummy</i>
)	[inline]	

Does Zero Order Decimation (Mean).

```
00046 {
00047     int x0,x1,y0,y1;
00048     (void)Dummy;
00049
00050     x0 = (int)floor(x1 - box_width/2.0 - 0.5) + 1;
00051     y0 = (int)floor(y1 - box_height/2.0 - 0.5) + 1;
00052     x1 = (int)floor(x1 + box_width/2.0 - 0.5);
00053     y1 = (int)floor(y1 + 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>x1,</i>
	float	<i>y1,</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(x1 - box_width/2.0 - 0.5) + 1;
00095      y0 = (int)floor(y1 - box_height/2.0 - 0.5) + 1;
00096      x1 = (int)floor(x1 + box_width/2.0 - 0.5);
00097      y1 = (int)floor(y1 + 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>x1,</i>
	float	<i>y1</i>
) [inline]	

Does Zero Order Interpolation (Nearest Neighborhood).

```

00147 {
00148     int x0 = (int)(x1-0.5f);
00149     int y0 = (int)(y1-0.5f);
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>x1,</i>
	float	<i>y1</i>
) [inline]	

Does Bilinear Interpolation.

```

00159 {
00160     int x0 = (int)(x1-0.5f);
00161     int y0 = (int)(y1-0.5f);
00162     int x1 = x0+1;
00163     int y1 = y0+1;
00164
00165     float t = x1 - (x0+0.5f);
00166     float u = y1 - (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 f1l = map[y0*width + x0];
00174   T fh1 = map[y0*width + x1];
00175   T flh = map[y1*width + x0];
00176   T fhh = map[y1*width + x1];
00177
00178   return (T)((fhh - flh - fh1 + f1l) * u * t +
00179              (fh1 - f1l) * t +
00180              (flh - f1l) * u +
00181              f1l);
00182 }

```

template<class T, class TU>

T imBicubicInterpolation (int	<i>width,</i>
	int	<i>height,</i>
	T *	<i>map,</i>
	float	<i>x1,</i>
	float	<i>y1,</i>
	TU	<i>Dummy</i>
)	[inline]	

Does Bicubic Interpolation.

```

00188 {
00189   (void)Dummy;
00190
00191   int x0 = (int)(x1-0.5f);
00192   int y0 = (int)(y1-0.5f);
00193   int x1 = x0-1;
00194   int x2 = x0+2;
00195   int y1 = y0-1;
00196   int y2 = y0+2;
00197
00198   float t = x1 - (x0+0.5f);
00199   float u = y1 - (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<=(TU)0? (TU)0: Value<=(TU)255? Value: (T
00263 else
00264     return (T)(Value);
00265 }

```

template<class T>

void imMinMax (const T *	<i>map,</i>
	int	<i>count,</i>
	T &	<i>min,</i>
	T &	<i>max</i>
) [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 }

```

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.

Binary File Access

[Utilities]

Detailed Description

These functions are very useful 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_FILEHANDLE, 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.

Enumerator:

<code>IM_RAWFILE</code>	System dependent file I/O Rotines.
<code>IM_STREAM</code>	Standard Ansi C Stream I/O Rotines.
<code>IM_MEMFILE</code>	Uses a memory buffer.
<code>IM_SUBFILE</code>	It is a sub file. FileName is a <code>imBinFile*</code> pointer fr
<code>IM_FILEHANDLE</code>	System dependent file I/O Rotines, but FileName
<code>IM_IOCUSTOM0</code>	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 imBinFi
00108     IM_FILEHANDLE,/**< System dependent file I/O Rotines, but I
00109     IM_IOCUSTOM0 /**< Other registered modules starts from hei
00110 };
```

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.

```
int imBinFileByteOrder ( imBinFile * bfile,  
                          int pByteOrder  
                          )
```

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.

ATTENTION: 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 beginning 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.

Data Compression Utilities

[Utilities]

Detailed Description

Deflate compression support uses zlib version 1.2.3.

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

Copyright (C) 1995-2004 Jean-loup Gailly and Mark Adler

LZF compression support uses libLZF version 1.51.

<http://liblzf.plan9.de/>

Copyright (C) 2000-2005 Marc Alexander Lehmann 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)

int **imCompressDataLZF** (const void *src_data, int src_size, void *dst_data, int dst_size, int zip_quality)

int **imCompressDataUnLZF** (const void *src_data, int src_size, void *dst_data, int dst_size)

Function Documentation

int imCompressDataZ (const void *	<i>src_data,</i>
	int	<i>src_size,</i>
	void *	<i>dst_data,</i>
	int	<i>dst_size,</i>
	int	<i>zip_quality</i>
)		

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 or zero if failed.

int imCompressDataUnZ (const void *	<i>src_data,</i>
	int	<i>src_size,</i>
	void *	<i>dst_data,</i>
	int	<i>dst_size</i>
)		

Uncompresses the data compressed with the ZLIB Deflate compression.
Returns zero if failed.

int imCompressDataLZF (const void *	<i>src_data,</i>
	int	<i>src_size,</i>
	void *	<i>dst_data,</i>
	int	<i>dst_size,</i>
	int	<i>zip_quality</i>
)		

)

Compresses the data using the libLZF compression.
Returns the size of the compressed buffer or zero if failed.

int imCompressDataUnLZF (const void *	<i>src_data,</i>
	int	<i>src_size,</i>
	void *	<i>dst_data,</i>
	int	<i>dst_size</i>
)		

Uncompresses the data compressed with the libLZF compression.
Returns zero if failed.

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 * userData,  
imCounterCallback callback,  
void * userData )
```

Changes the counter callback. Returns old callback.
User data is changed only if not NULL.

```
int imCounterBegin ( const char * title )
```

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 counter )
```

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 counter )
```

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 counter,  
int total,  
void * userData )
```

```
const char * message  
)
```

Sets the total increments of a count.

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

```
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)
```

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 purposes.
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 image,  
const HPALETTE hPalette  
)
```

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:

<code>_imBinMemoryFileName</code>	<i>Memory File I/O Filename</i>
<code>_imDib</code>	<i>Windows DIB Structure</i>
<code>_imFile</code>	<i>Image File Format Base (SDK Use Only)</i>
<code>_imImage</code>	<i>ImImage Structure Definition</i>
<code>_imStats</code>	
<code>imAttribArray</code>	<i>Attributes Table</i>
<code>imAttribTable</code>	<i>Attributes Table</i>
<code>imcfloat</code>	<i>Complex Float Data Type</i>
<code>imFormat</code>	<i>Image File Format Driver (SDK Use Only)</i>
<code>imImageFile</code>	<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*) [inline]

Copies the contents of the given table into this table.

```
00047 { imAttribTableCopyFrom(pTable, table.pTable); }
```

void imAttribTable::Set (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 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 * name,  
int * data_type = 0,  
int * count = 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 * user_data,  
imAttribTableCallback attrib_func  
) 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_attr.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. $\text{size} += \text{reallocate} * \text{size}$.

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 image_index
    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 unpack/pack to/from 1 byte. When reading converts n packed bits to 1 byte (unpack). If n>1 will also expand to 0-255. When writing converts 1 byte to 1 bit (pack). If negative will only expand to 0-255 (no unpack or pack).

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) {}
```

<code>imcfloat::imcfloat (</code>	<code>const float & <i>r</i>,</code>
	<code>const float & <i>i</i></code>
<code>)</code>	<code>[inline]</code>

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](#)

_imlImage Struct Reference

[imlImage]

Detailed Description

An image representation than supports all the color spaces, but planes are always unpacked and the orientation is always bottom up.

Data Fields

```
int width  
int height  
int color_space  
int data_type  
int has_alpha  
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 _imlImage::width

Number of columns. image:Width() -> width: number [in Lua 5].

int _imlImage::height

Number of lines. image:Height() -> height: number [in Lua 5].

int _imlImage::color_space

Color space descriptor. image:ColorSpace() -> color_space: number [in Lua 5].

int _imlImage::data_type

Data type descriptor. image:DataType() -> data_type: number [in Lua 5].

int _imlImage::has_alpha

Indicates that there is an extra channel with alpha. image:HasAlpha() -> has_alpha: number [in Lua 5].

It will not affect the secondary parameters, i.e. the number of planes will be in fact depth+1.

It is always 0 unless imlImageAddAlpha is called, this is done in image load functions.

int _imlImage::depth

Number of planes (ColorSpaceDepth)

int _imlImage::line_size

Number of bytes per line in one plane (width * DataTypeSize)

int _imlImage::plane_size

Number of bytes per plane. (line_size * height)

int _imlImage::size

Number of bytes occupied by the image (plane_size * depth)

int _imlImage::count

Number of pixels (width * height)

void _imlImage::data**

Image data organized as a 2D matrix with several planes.

But plane 0 is also a pointer to the full data.

The remaining planes are: $\text{data}[i] = \text{data}[0] + i * \text{plane_size}$

In Lua, indexing is possible using: `image[plane][row][column]`

long* _imlImage::palette

Color palette. `image:GetPalette()` -> `palette: imPalette [in Lua 5]`.

Used when `depth=1`. Otherwise is NULL.

int _imlImage::palette_count

The palette is always 256 colors allocated, but can have less colors used.

void* _imlImage::attrib_table

in fact is an **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_colorhsi.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]	<i>IM Lua 5 Binding</i>
old_im.h [code]	<i>Old API</i>

include

im.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im.h](#),v 1.5 2005/12/12 13:31:08 scuri 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

[im_attrib.h](#),v 1.1 2005/04/02 22:07:00 scuri 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

[im_attrib_flat.h](#),v 1.1 2005/04/02 22:07:00 scuri 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

[im_binfile.h](#),v 1.2 2005/07/15 19:58:06 scuri 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_FILEHANDLE, 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

[im_capture.h](#),v 1.5 2005/12/11 23:41:25 scuri 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 imVideoCaptureDialogCount (imVideoCapture *vc)
int imVideoCaptureShowDialog (imVideoCapture *vc,
int dialog, void *parent)
const char * imVideoCaptureDialogDesc (imVideoCapture *vc,
int dialog)
int imVideoCaptureFormatCount (imVideoCapture *vc)
int imVideoCaptureGetFormat (imVideoCapture *vc, int
format, int *width, int *height, char *desc)
int imVideoCaptureSetFormat (imVideoCapture *vc, int
format)
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

[im_color.h](#),v 1.1 2005/04/02 22:07:00 scuri 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:

```
( _w > 0.008856f? \
    powf(_w, 1.0f/3.0f): \
    7.787f * _w + 0.16f/1.16f)
```

```
#define IM_GWLAB ( _w )
```

Value:

```
( _w > 0.20689f? \
    powf(_w, 3.0f): \
    0.1284f * (_w - 0.16f/1.16f))
```

include

im_colorhsi.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_colorhsi.h](#),v 1.2 2005/07/15 19:59:21 scuri Exp

[Go to the source code of this file.](#)

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)

include

im_complex.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_complex.h](#),v 1.1 2005/04/02 22:07:00 scuri 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

[im_convert.h](#),v 1.3 2005/12/10 00:54:39 scuri 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 absolute, 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 absolute, 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

[im_counter.h](#),v 1.1 2005/04/02 22:07:00 scuri 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

[im_dib.h](#),v 1.1 2005/04/02 22:07:00 scuri 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

[im_file.h](#),v 1.2 2005/08/10 02:12:16 scuri 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

[im_format.h](#),v 1.1 2005/04/02 22:07:00 scuri 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

[im_format_all.h](#),v 1.8 2005/12/07 20:28:27 scuri 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

[im_format_avi.h](#),v 1.4 2005/12/06 00:58:09 scuri 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

[im_format_jp2.h](#),v 1.2 2005/04/14 19:35:39 scuri 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

[im_format_raw.h](#),v 1.1 2005/04/02 22:07:00 scuri 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

[im_format_wmv.h](#),v 1.2 2005/08/10 02:12:16 scuri 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

[im_image.h](#),v 1.7 2005/12/12 13:31:08 scuri Exp

[Go to the source code of this file.](#)

Data Structures

struct [_imlImage](#)

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

Defines

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

Typedefs

```
typedef \_imlImage imlImage
```

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)

imlImage * **imlImageCreateBased** (**imlImage** *image, int width, int height, int color_space, int data_type)

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

void **imlImageAddAlpha** (**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

```
    imImage *image2)
int imImageMatchColor (const imImage *image1, const
imImage *image2)
int imImageMatchDataType (const imImage *image1, const
imImage *image2)
int imImageMatchColorSpace (const imImage *image1,
const imImage *image2)
int imImageMatch (const imImage *image1, const imImage
*image2)
void imImageSetBinary (imImage *image)
void imImageMakeBinary (imImage *image)
imImage * imFileLoadImage (imFile *ifile, int index, int *error)
void imFileLoadImageFrame (imFile *ifile, int index, imImage
*image, int *error)
imImage * imFileLoadBitmap (imFile *ifile, int index, int *error)
void imFileLoadBitmapFrame (imFile *ifile, int index, imImage
*image, int *error)
int imFileSaveImage (imFile *ifile, const imImage *image)
imImage * imFileImageLoad (const char *file_name, int index, int
*error)
imImage * imFileImageLoadBitmap (const char *file_name, int index,
int *error)
int imFileImageSave (const char *file_name, const char
*format, const imImage *image)
```

include

im_lib.h File Reference

Detailed Description

See Copyright Notice in this file.

Id

[im_lib.h](#),v 1.6 2005/12/13 18:58:44 scuri Exp

[Go to the source code of this file.](#)

Defines

```
#define IM_AUTHOR "Antonio Scuri"
```

```
#define IM_COPYRIGHT "Copyright (C) 1994-2005 Tecgraf/PUC-Rio  
and PETROBRAS S/A"
```

```
#define IM_VERSION "3.1.0"
```

```
#define IM_VERSION_DATE "2005/12/12"
```

```
#define IM_VERSION_NUMBER 301000
```

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

[im_math.h](#),v 1.2 2005/05/19 19:34:35 uid20021 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

[im_math_op.h](#),v 1.2 2005/12/05 20:44:36 scuri 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)
    imbyte pow_op (const imbyte &v1, const imbyte &v2)
    imushort pow_op (const imushort &v1, const imushort &v2)
    int pow_op (const int &v1, const int &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.

```
00105 {  
00106     return (T1)pow(v1, v2);  
00107 }
```

```
template<class T>
```

```
T abs_op ( const T & v ) [inline]
```

Generic Absolute.

```
00112 {  
00113     if (v <= 0)  
00114         return -1*v;  
00115     return v;  
00116 }
```

```
template<class T>
```

```
T less_op ( const T & v ) [inline]
```

Generic Less.

```
00121 {  
00122     return -1*v;  
00123 }
```

```
template<class T>
```

```
T sqr_op ( const T & v ) [inline]
```

Generic Square.

```
00128 {  
00129     return v*v;  
00130 }
```

template<class T>

T sqrt_op (const T & v) [inline]

Generic Square Root.

```
00140 {  
00141     return (T)sqrt(v);  
00142 }
```

template<class T>

T exp_op (const T & v) [inline]

Generic Exponential.

```
00152 {  
00153     return (T)exp(v);  
00154 }
```

template<class T>

T log_op (const T & v) [inline]

Generic Logarithm.

```
00164 {  
00165 //   if (v <= 0) return (T)IM_NEARINF;  
00166     return (T)log(v);  
00167 }
```

template<class T>

T sin_op (const T & v) [inline]

Generic Sine.

```
00183 {  
00184     return (T)sin(v);  
00185 }
```

template<class T>

```
T cos_op ( const T & v ) [inline]
```

Generic Cosine.

```
00201 {  
00202     return (T)cos(v);  
00203 }
```

```
void imDataBitSet ( imbyte * data,  
                    int      index,  
                    int      bit  
                    ) [inline]
```

Sets a bit in an array.

```
00207 {  
00208     if (bit)  
00209         data[index / 8] |= (0x01 << (7 - (index % 8)));  
00210     else  
00211         data[index / 8] &= ~(0x01 << (7 - (index % 8)));  
00212 }
```

```
int imDataBitGet ( imbyte * data,  
                   int      index  
                   ) [inline]
```

Gets a bit from an array.

```
00216 {
```

```
00217     return (data[index / 8] >> (7 - (index % 8))) & 0x01;
00218 }
```

include

im_palette.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_palette.h](#),v 1.4 2005/12/12 13:31:08 scuri 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

[im_plus.h](#),v 1.1 2005/04/02 22:07:00 scuri Exp

[Go to the source code of this file.](#)

Data Structures

class [imlImageFile](#)

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

include

im_process_ana.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_process_ana.h](#),v 1.6 2005/12/11 23:41:25 scuri 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** *image)

void **imCalcHistogram** (const unsigned char *data, int count, unsigned long *histo, int cumulative)

void **imCalcUShortHistogram** (const unsigned short *data, int count, unsigned long *histo, int cumulative)

void **imCalcGrayHistogram** (const **imImage** *image, unsigned long *histo, int cumulative)

void **imCalcImageStatistics** (const **imImage** *image, **imStats** *stats)

void **imCalcHistogramStatistics** (const **imImage** *image, **imStats** *stats)

void **imCalcHistImageStatistics** (const **imImage** *image, int *median, int *mode)

int **imAnalyzeFindRegions** (const **imImage** *src_image, **imImage** *dst_image, int connect, int touch_border)

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 *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)
```

include

im_process_glo.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_process_glo.h](#),v 1.4 2005/12/12 20:29:00 scuri Exp

[Go to the source code of this file.](#)

Functions

```
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)
void imProcessFFT (const imlImage *src_image, imlImage *dst_image)
void imProcessIFFT (const imlImage *src_image, imlImage
    *dst_image)
void imProcessFFTraw (imlImage *image, int inverse, int center, int
    normalize)
void imProcessSwapQuadrants (imlImage *image, int center2origin)
```

include

im_process_loc.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_process_loc.h](#),v 1.6 2005/12/13 18:35:02 scuri 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 imProcessInsert (const imlImage *src_image, const imlImage
    *region_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_clockwise)
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 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)
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)
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 imImage *src_image,  
    imImage *dst_image, int kernel_size)  
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_level)  
int imProcessConvolve (const imImage *src_image, imImage  
    *dst_image, const imImage *kernel)  
int imProcessConvolveRep (const imImage *src_image, imImage  
    *dst_image, const imImage *kernel, int count)  
int imProcessCompassConvolve (const imImage *src_image,  
    imImage *dst_image, imImage *kernel)  
void imProcessRotateKernel (imImage *kernel)  
int imProcessDiffOfGaussianConvolve (const imImage *src_image,  
    imImage *dst_image, float stddev1, float stddev2)  
int imProcessDiffOfGaussianConvolveRep (const imImage  
    *src_image, imImage *dst_image, float stddev1, float stddev2)  
int imProcessLapOfGaussianConvolve (const imImage *src_image,  
    imImage *dst_image, float stddev)  
int imProcessMeanConvolve (const imImage *src_image, imImage  
    *dst_image, int kernel_size)  
int imProcessGaussianConvolveRep (const imImage *src_image,  
    imImage *dst_image, float stddev)  
int imProcessGaussianConvolve (const imImage *src_image,  
    imImage *dst_image, float stddev)  
int imProcessSobelConvolve (const imImage *src_image, imImage  
    *dst_image)  
void imProcessZeroCrossing (const imImage *src_image, imImage  
    *dst_image)  
void imProcessCanny (const imImage *src_image, imImage  
    *dst_image, float stddev)  
int imGaussianStdDev2Repetitions (float stddev)  
int imGaussianStdDev2KernelSize (float stddev)
```

include

im_process_pon.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_process_pon.h](#),v 1.6 2005/12/12 20:29:00 scuri 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_CPXNORM  
}  
  
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 imImage *src_image,
    imImage *dst_image, int op)
void imProcessArithmeticOp (const imImage *src_image1, const
    imImage *src_image2, imImage *dst_image, int op)
void imProcessArithmeticConstOp (const imImage *src_image, float
    src_const, imImage *dst_image, int op)
void imProcessBlend (const imImage *src_image1, imImage
    *src_image2, imImage *dst_image, float alpha)
void imProcessSplitComplex (const imImage *src_image, imImage
    *dst_image1, imImage *dst_image2, int do_polar)
void imProcessMergeComplex (const imImage *src_image1, const
    imImage *src_image2, imImage *dst_image, int polar)
void imProcessMultipleMean (const imImage **src_image_list, int
    src_image_count, imImage *dst_image)
void imProcessMultipleStdDev (const imImage **src_image_list, int
    src_image_count, const imImage *mean_image, imImage
    *dst_image)
    int imProcessAutoCovariance (const imImage *src_image, const
    imImage *mean_image, imImage *dst_image)
void imProcessMultiplyConj (const imImage *src_image1, const
    imImage *src_image2, imImage *dst_image)
void imProcessQuantizeRGBUniform (const imImage *src_image,
    imImage *dst_image, int do_dither)
void imProcessQuantizeGrayUniform (const imImage *src_image,
    imImage *dst_image, int grays)
void imProcessExpandHistogram (const imImage *src_image,
    imImage *dst_image, float percent)
void imProcessEqualizeHistogram (const imImage *src_image,
    imImage *dst_image)
```

```
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_image, imImage *dst_image)
void imProcessSplitComponents (const imImage *src_image,
    imImage **dst_image_list)
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)
void imProcessBitwiseOp (const imImage *src_image1, const
    imImage *src_image2, imImage *dst_image, int op)
void imProcessBitwiseNot (const imImage *src_image, imImage
    *dst_image)
void imProcessBitMask (const imImage *src_image, imImage
    *dst_image, unsigned char mask, int op)
void imProcessBitPlane (const imImage *src_image, imImage
    *dst_image, int plane, int do_reset)
int imProcessRenderOp (imImage *image, imRenderFunc
    render_func, char *render_name, float *param, int plus)
int imProcessRenderCondOp (imImage *image,
    imRenderCondFunc render_cond_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** (iImage *image, float *value)
int **imProcessRenderWheel** (iImage *image, int internal_radius, int external_radius)
int **imProcessRenderCone** (iImage *image, int radius)
int **imProcessRenderTent** (iImage *image, int tent_width, int tent_height)
int **imProcessRenderRamp** (iImage *image, int start, int end, int vert_dir)
int **imProcessRenderBox** (iImage *image, int box_width, int box_height)
int **imProcessRenderSinc** (iImage *image, float x_period, float y_period)
int **imProcessRenderGaussian** (iImage *image, float stddev)
int **imProcessRenderLapOfGaussian** (iImage *image, float stddev)
int **imProcessRenderCosine** (iImage *image, float x_period, float y_period)
int **imProcessRenderGrid** (iImage *image, int x_space, int y_space)
int **imProcessRenderChessboard** (iImage *image, int x_space, int y_space)
void **imProcessToneGamut** (const iImage *src_image, iImage *dst_image, int op, float *param)
void **imProcessUnNormalize** (const iImage *src_image, iImage *dst_image)
void **imProcessDirectConv** (const iImage *src_image, iImage *dst_image)
void **imProcessNegative** (const iImage *src_image, iImage *dst_image)
void **imProcessThreshold** (const iImage *src_image, iImage *dst_image, int level, int value)
void **imProcessThresholdByDiff** (const iImage *src_image1, const iImage *src_image2, iImage *dst_image)
void **imProcessHysteresisThreshold** (const iImage *src_image,

```
    imlImage *dst_image, int low_thres, int high_thres)
void imProcessHysteresisThresEstimate (const imlImage *image, int
    *low_level, int *high_level)

    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 imProcessLocalMaxThresEstimate (const imlImage *image, int
    *level)
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_raw.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_raw.h](#),v 1.2 2005/12/10 00:54:39 scuri 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

[im_util.h](#),v 1.5 2005/12/12 00:18:29 scuri 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 imColorModelsPacked(_cm) (_cm & IM_PACKED)
#define imColorModelsTopDown(_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 **imImageDataSize** (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)
int imCompressDataLZF (const void *src_data, int src_size,
    void *dst_data, int dst_size, int zip_quality)
int imCompressDataUnLZF (const void *src_data, int
    src_size, void *dst_data, int dst_size)
```

include

implua.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[imlua.h](#),v 1.5 2005/12/12 15:42:29 scuri Exp

[Go to the source code of this file.](#)

Functions

```
int implua_open (lua_State *L)
int implua_open_capture (lua_State *L)
int implua_open_process (lua_State *L)
int implua_open_fftw (lua_State *L)
int implua_open_cd (lua_State *L)
```

include

old_im.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[old_im.h](#),v 1.1 2005/04/02 22:07:00 scuri 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](#) | [Enumerator](#) | [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](#)

[All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- a -

- `abs_op()` : [im_math_op.h](#)
- `add_op()` : [im_math_op.h](#)

[All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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](#) | [Enumerator](#) | [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_FILEHANDLE : [im_binfile.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_XYZ : [im.h](#)

- IM_YCBCR : [im.h](#)

[All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)

- 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](#)
- imPutBitmap : [im_image.h](#)

Tecgraf/PUC-Rio Library Download Tips

All the libraries were build using **Tecmake**. Please use it if you intend to recompile the sources. **Tecmake** can be found at <http://www.tecgraf.puc-rio.br/tecmake>.

Executables are linked statically in all platforms to make distribution and execution simpler. Libraries are built with speed optimization. The source code with the "config.mak" files for **Tecmake** are also available for download.

The DLLs are not compatible with Visual Basic or Delphi. They do not use the stdcall calling convention.

By decision, in Visual C++ we use the single thread C Run Time Library for static libraries and the multi thread C RTL for DLLs.

The documentation files does not include the CHM and PDF versions. They are provided only as a separate download.

The **IM** files can be downloaded at

http://luaforge.net/project/showfiles.php?group_id=86.

The **CD** files can be downloaded at

http://luaforge.net/project/showfiles.php?group_id=88.

The **IUP** files can be downloaded at

http://luaforge.net/project/showfiles.php?group_id=89.

The **Lua** files can be downloaded at

http://luaforge.net/project/showfiles.php?group_id=110.

Binaries Description

AIX43 IBM AIX 4.3 (ppc) / gcc 2.95.2 / Motif 2.1.0

AIX43cc IBM AIX 4.3 (ppc) / cc 4.4 / Motif 2.1.0

IRIX65 SGI IRIX 6.5 (mips) / gcc 3.0.4 / Motif 2.1.20

IRIX6465 SGI IRIX 6.5 (mips) (64 bits OS, but libs are still 32 bits) / gcc 3.0.4 / Motif 1.2.4

IRIX6465cc SGI IRIX 6.5 (mips) (") / cc MIPSpro 7.30 / Motif 1.2.4

Linux24 Red Hat 7.3 (x86) / Kernel 2.4 / gcc 2.95.3 / Open Motif 2.1.3

Linux24g3	Red Hat E.L. WS 3 (x86) / Kernel 2.4 / gcc 3.2.3 / Open Motif 2.2.3
Linux24g3_64	Red Hat E.L. WS 3 (x86_64) / Kernel 2.4 / gcc 3.2.3 / Open Motif 2.2.3
Linux26	Fedora Core 3 (x86) / Kernel 2.6 / gcc 3.4.2 / Open Motif 2.2.3
SunOS57	Sun Solaris 7 (sparc) / gcc 2.95.2 / Motif 2.1.0
SunOS57cc	Sun Solaris 7 (sparc) / cc 5.2 (Sun WorkShop 6 update 1) / Motif 2.1.0
SunOS58	Sun Solaris 8 (sparc) / gcc 2.95.3 / Motif 2.1.0
FreBSD54	Free BSD 5.4 (x86) / gcc 3.4.2 / Motif 2.2.3
Darwin78	Mac OS X 10.3.8 (ppc) / Darwin Kernel Version 7.8.0 / gcc 3.4.2
Darwin79	Mac OS X 10.3.9 (ppc) / Darwin Kernel Version 7.9.0 / gcc 3.4.2
Darwin80	Mac OS X 10.4.0 (ppc) / Darwin Kernel Version 8.0.0 / gcc 4.0.1
dll	built using vc6, creates dependency with MSVCRT.DLL (either existing libraries or new applications).
dll7	built using vc7, creates dependency with MSVCR71.DLL (either existing libraries or new applications).
vc6	Microsoft Visual C++ 6 (static RTL/single thread)
vc7	Microsoft Visual C++ 7.1 (.NET 2003) (static RTL/single thread)
owc1	Open Watcom 1.x - http://www.openwatcom.org/
gcc3	Cygwin gcc 3.2 - http://www.cygwin.com/
mingw3	MingW gcc 3.2 - http://www.mingw.org/ Also compatible with Dev-C++ http://www.bloodshed.net/dev_cpp/
bc55	Borland C++ 5.5.1 Free Command Line Compiler - http://www.borland.com/products/downloads/download_cbuild55.exe
bc56	Borland C++ BuilderX 1.0 / Borland C++ 5.6.4 Compiler - http://www.borland.com/products/downloads/download_cbuild56.exe (the C++ BuilderX IDE can also be configured to use mingw32 versions.)
Win32	Executable for Windows NT/2000/XP

_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](#)

NSMutableArray 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 index,  
const char * name,  
int data_type,  
int count,  
const void * data  
) [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(patable, 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(patable, 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: im_attrib.h,v 1.1 2005/04/02 22:07:00 scuri 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 that 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 Praticice 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 v
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 *c
00061     { return imAttribTableGet(pTable, name, data_type, count);
00062
00063 /** For each attribute calls the user callback. If the callb
00064 void ForEach(void* user_data, imAttribTableCallback attrib_f
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: im_binfile.h,v 1.2 2005/07/15 19:58:06 scuri 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 diferent
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, 8.
00063  * \ingroup binfile */
00064 unsigned long imBinFileRead(imBinFile* bfile, void* pValues, unsigned long count);
00065
00066 /** Writes an array of values with sizes: 1, 2, 4, or 8. And i
00067  * ATTENTION: The function will not make a temporary copy.
00068  * So after the call the values will be invalid, if the file is opened in write mode.
00069  * \ingroup binfile */
00070 unsigned long imBinFileWrite(imBinFile* bfile, void* pValues, unsigned long count);
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 beginning 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.\n
00088 * The offset is usually a negative value.
00089 * \ingroup binfile */
00090 void imBinFileSeekFrom(imBinFile* bfile, long pOffset);
00091
00092 /** Returns the current offset position.
00093 * \ingroup binfile */
00094 unsigned long imBinFileTell(imBinFile* bfile);
00095
00096 /** Indicates that the file pointer is at the end of the file.
00097 * \ingroup binfile */
00098 int imBinFileEndOfFile(imBinFile* bfile);
00099
00100 /** Predefined I/O Modules.
00101 * \ingroup binfile */
00102 enum imBinFileModule
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
00108     IM_FILEHANDLE,/**< System dependent file I/O Rotines, but Fi
00109     IM_IOCUSTOM0  /**< Other registered modules starts from here
00110 };
00111
00112 /** Sets the current I/O module.
00113 * \returns the previous function set, or -1 if failed.
00114 * \ingroup binfile */
00115 int imBinFileSetCurrentModule(int pModule);
00116
00117 /** \brief Memory File I/O Filename
00118 *
00119 * \par
00120 * Fake file name for the memory I/O module.
00121 * \ingroup binfile */
00122 typedef struct _imBinMemoryFileName
00123 {
00124     unsigned char *buffer; /**< The memory buffer. If you are re
00125                             * If you are writing the buffer ca
00126     int size;              /**< Size of the buffer. */
00127     float reallocate;     /**< Reallocate factor for the memory
00128 }imBinMemoryFileName;
00129
00130
00131 #if defined(__cplusplus)
00132 }
00133 #endif

```

```

00134
00135
00136 #if defined(__cplusplus)
00137
00138 /** Base class to help the creation of new modules.\n
00139  * It handles the read/write operations with byte order correc
00140  * \ingroup binfile */
00141 class imBinFileBase
00142 {
00143     friend class imBinSubFile;
00144
00145 protected:
00146     int IsNew,
00147         FileByteOrder,
00148         DoByteOrder;    // to speed up byte order checking
00149
00150     // These will actually read/write the data
00151     virtual unsigned long ReadBuf(void* pValues, unsigned long p
00152     virtual unsigned long WriteBuf(void* pValues, unsigned long
00153
00154 public:
00155
00156     int InitByteOrder(int ByteOrder)
00157     {
00158         int old_byte_order = this->FileByteOrder;
00159         this->FileByteOrder = ByteOrder;
00160
00161         if (ByteOrder != imBinCPUByteOrder())
00162             this->DoByteOrder = 1;
00163         else
00164             this->DoByteOrder = 0;
00165         return old_byte_order;
00166     }
00167
00168     // These will take care of byte swap if needed.
00169
00170     unsigned long Read(void* pValues, unsigned long pCount, int
00171     {
00172         unsigned long rSize = ReadBuf(pValues, pCount * pSizeOf);
00173         if (pSizeOf != 1 && DoByteOrder) imBinSwapBytes(pValues, p
00174         return rSize/pSizeOf;
00175     }
00176
00177     unsigned long Write(void* pValues, unsigned long pCount, int
00178     {
00179         if (pSizeOf != 1 && DoByteOrder) imBinSwapBytes(pValues, p
00180         return WriteBuf(pValues, pCount * pSizeOf)/pSizeOf;

```

```
00181 }
00182
00183 virtual void Open(const char* pFileName) = 0;
00184 virtual void New(const char* pFileName) = 0;
00185 virtual void Close() = 0;
00186 virtual unsigned long FileSize() = 0;
00187 virtual int HasError() const = 0;
00188 virtual void SeekTo(unsigned long pOffset) = 0;
00189 virtual void SeekOffset(long pOffset) = 0;
00190 virtual void SeekFrom(long pOffset) = 0;
00191 virtual unsigned long Tell() const = 0;
00192 virtual int EndOfFile() const = 0;
00193 };
00194
00195 /** File I/O module creation callback.
00196  * \ingroup binfile */
00197 typedef imBinFileBase* (*imBinFileNewFunc)();
00198
00199 /** Register a user I/O module.\n
00200  * Returns the new function set id.\n
00201  * Accepts up to 10 modules.
00202  * \ingroup binfile */
00203 int imBinFileRegisterModule(imBinFileNewFunc pNewFunc);
00204
00205 #endif
00206
00207 #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: im_dib.h,v 1.1 2005/04/02 22:07:00 scuri 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 ignore
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->bits
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 th
00063     int              bits_size;      /**< size in bytes of the B
00064     int              line_size;      /**< size in bytes of one l
00065     int              pad_size;       /**< number of bytes remain
00066
00067     int              is_reference;   /**< only a reference, do n
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 purposes. \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: im_file.h,v 1.2 2005/08/10 02:12:16 scuri 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
00036
00037     void* line_buffer;    /**< used for line conversion, contain
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 unpack/pack to/
00044                       When reading converts n packed b
00045                       When writing converts 1 byte to
00046                       If negative will only expand to
00047 int switch_type;      /**< flag to switch the original data
00048
00049 long palette[256];
00050 int palette_count;
00051
00052 int user_color_mode,
00053     user_data_type,
00054     file_color_mode, /* these two must be filled by te dri
00055     file_data_type;
00056
00057 /* these must be filled by the driver when reading,
00058     and given by the user when writing. */
00059
00060 char compression[10];
00061 int image_count,
00062     image_index,
00063     width,
00064     height;
00065 };
00066
00067
00068 /* Internal Use only */
00069
00070 /* Initializes the imFile structure.
00071 * Used by the special format RAW. */
00072 void imFileClear(imFile* ifile);
00073
00074 /* Initializes the line buffer.
00075 * Used by "im_file.cpp" only. */
00076 void imFileLineBufferInit(imFile* ifile);
00077
00078 /* Check if the conversion is valid.
00079 * Used by "im_file.cpp" only. */
00080 int imFileCheckConversion(imFile* ifile);
00081
00082
00083
00084 /* File Format SDK */
00085
00086 /** Number of lines to be accessed.

```

```
00087 * \ingroup filesdk */
00088 int imFileLineBufferCount(imFile* ifile);
00089
00090 /** Increments the row and plane counters.
00091 * \ingroup filesdk */
00092 void imFileLineBufferInc(imFile* ifile, int *row, int *plane);
00093
00094 /** Converts from FILE color mode to USER color mode.
00095 * \ingroup filesdk */
00096 void imFileLineBufferRead(imFile* ifile, void* data, int line,
00097
00098 /** Converts from USER color mode to FILE color mode.
00099 * \ingroup filesdk */
00100 void imFileLineBufferWrite(imFile* ifile, const void* data, in
00101
00102 /** Utility to calculate the line size in byte with a specifie
00103 * "align" can be 1, 2 or 4.
00104 * \ingroup filesdk */
00105 int imFileLineSizeAligned(int width, int bpp, int align);
00106
00107
00108 #if defined(__cplusplus)
00109 }
00110 #endif
00111
00112 #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: im_format.h,v 1.1 2005/04/02 22:07:00 scuri 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: im_complex.h,v 1.1 2005/04/02 22:07:00 scuri 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.imag, C.real) / 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: im_image.h,v 1.7 2005/12/12 13:31:08 scuri 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 imImage
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 NULL before calling imImageDestroy.
00026  * \par
00027  * See \ref im_image.h
00028  * \ingroup imagerep */
00029
00030
00031
00032 /** \brief imImage Structure Definition.
00033  *
00034  * \par
00035  * An image representation than supports all the color spaces,
00036  * but planes are always unpacked and the orientation is always
00037  * \ingroup imgclass */
00038 typedef struct _imImage
00039 {
```

```

00040  /* main parameters */
00041  int width;          /**< Number of columns. image:Width() ->
00042  int height;        /**< Number of lines. image:Height() ->
00043  int color_space;   /**< Color space descriptor. image:Color
00044  int data_type;     /**< Data type descriptor. image:DataTyp
00045  int has_alpha;     /**< Indicates that there is an extra ch
00046                    It will not affect the secondary pa
00047                    It is always 0 unless imImageAddAlp
00048
00049  /* secondary parameters */
00050  int depth;         /**< Number of planes
00051  int line_size;    /**< Number of bytes per line in one pla
00052  int plane_size;   /**< Number of bytes per plane.
00053  int size;         /**< Number of bytes occupied by the ima
00054  int count;        /**< Number of pixels
00055
00056  /* image data */
00057  void** data;      /**< Image data organized as a 2D matrix
00058                    But plane 0 is also a pointer to th
00059                    The remaining planes are: data[i] =
00060                    In Lua, indexing is possible using:
00061
00062  /* image attributes */
00063  long *palette;    /**< Color palette. image:GetPalette() -
00064                    Used when depth=1. Otherwise is NUL
00065  int palette_count; /**< The palette is always 256 colors al
00066
00067  void* attrib_table; /**< in fact is an imAttribTable, but we
00068 } imImage;
00069
00070
00071 /** Creates a new image.
00072 *
00073 * \verbatim im.ImageCreate(width: number, height: number, col
00074 * \ingroup imgclass */
00075 imImage* imImageCreate(int width, int height, int color_space,
00076
00077 /** Initializes the image structure but does not allocates ima
00078 * \ingroup imgclass */
00079 imImage* imImageInit(int width, int height, int color_space, i
00080
00081 /** Creates a new image based on an existing one. \n
00082 * If the adicional parameters are -1, the given image parame
00083 * The image atributes always are copied.
00084 *
00085 * \verbatim im.ImageCreateBased(image: imImage, [width: numbe
00086 * The adicional parameters in Lua should be nil, and they ca

```

```

00087 * \ingroup imgclass */
00088 imImage* imImageCreateBased(imImage* image, int width, int hei
00089
00090 /** Destroys the image and frees the memory used.
00091 * image data is destroyed only if its data[0] is not NULL.
00092 *
00093 * \verbatim im.ImageDestroy(image: imImage) [in Lua 5] \endve
00094 * \verbatim image:Destroy() [in Lua 5] \endverbatim
00095 * \ingroup imgclass */
00096 void imImageDestroy(imImage* image);
00097
00098 /** Adds an alpha channel plane.
00099 *
00100 * \verbatim image:AddAlpha() [in Lua 5] \endverbatim
00101 * \ingroup imgclass */
00102 void imImageAddAlpha(imImage* image);
00103
00104 /** Changes the buffer size. Reallocate internal buffers if th
00105 *
00106 * \verbatim image:Reshape(width: number, height: number) [in
00107 * \ingroup imgclass */
00108 void imImageReshape(imImage* image, int width, int height);
00109
00110 /** Copy image data and attributes from one image to another.
00111 * Images must have the same size and type.
00112 *
00113 * \verbatim image:Copy(dst_image: imImage) [in Lua 5] \endver
00114 * \ingroup imgclass */
00115 void imImageCopy(const imImage* src_image, imImage* dst_image)
00116
00117 /** Copy image data only fom one image to another. \n
00118 * Images must have the same size and type.
00119 *
00120 * \verbatim image:CopyData(dst_image: imImage) [in Lua 5] \en
00121 * \ingroup imgclass */
00122 void imImageCopyData(const imImage* src_image, imImage* dst_ir
00123
00124 /** Creates a copy of the image.
00125 *
00126 * \verbatim image:Duplicate() -> new_image: imImage [in Lua 5
00127 * \ingroup imgclass */
00128 imImage* imImageDuplicate(const imImage* image);
00129
00130 /** Creates a clone of the image. i.e. same attributes but ign
00131 *
00132 * \verbatim image:Clone() -> new_image: imImage [in Lua 5] \e
00133 * \ingroup imgclass */

```

```

00134 imImage* imImageClone(const imImage* image);
00135
00136 /** Changes an extended attribute. \n
00137 * The data will be internally duplicated. \n
00138 * If data is NULL the attribute is removed. \n
00139 * If count is -1 and data_type is IM_BYTE then data is zero t
00140 *
00141 * \verbatim image:SetAttribute(attrib: string, data_type: num
00142 * If data_type is IM_BYTE, as_string can be used as data.
00143 * \ingroup imgclass */
00144 void imImageSetAttribute(imImage* image, const char* attrib, i
00145
00146 /** Returns an extended attribute. \n
00147 * Returns NULL if not found.
00148 *
00149 * \verbatim image:GetAttribute(attrib: string, [as_string: bo
00150 * If data_type is IM_BYTE, as_string can be used to return a
00151 * \ingroup imgclass */
00152 const void* imImageGetAttribute(const imImage* image, const ch
00153
00154 /** Returns a list of the attribute names. \n
00155 * "attrib" must contain room enough for "attrib_count" names.
00156 *
00157 * \verbatim image:GetAttributeList() -> data: table of string
00158 * \ingroup imgclass */
00159 void imImageGetAttributeList(const imImage* image, char** attr
00160
00161 /** Sets all image data to zero.
00162 *
00163 * \verbatim image:Clear() [in Lua 5] \endverbatim
00164 * \ingroup imgclass */
00165 void imImageClear(imImage* image);
00166
00167 /** Indicates that the image can be viewed in common graphic d
00168 * Data type must be IM_BYTE. Color mode can be IM_RGB, IM_MAP
00169 *
00170 * \verbatim image:IsBitmap() -> is_bitmap: boolean [in Lua 5]
00171 * \ingroup imgclass */
00172 int imImageIsBitmap(const imImage* image);
00173
00174 /** Changes the image palette.
00175 * This will destroy the existing palette and replace it with
00176 *
00177 * \verbatim image:SetPalette(palette: imPalette) [in Lua 5] \
00178 * \ingroup imgclass */
00179 void imImageSetPalette(imImage* image, long* palette, int pale
00180

```

```

00181 /** Copies the image attributes from src to dst.
00182  *
00183  * \verbatim image:CopyAttributes(dst_image: imImage) [in Lua
00184  * \ingroup imgclass */
00185 void imImageCopyAttributes(const imImage* src_image, imImage*
00186
00187 /** Returns 1 if the images match width and height. Returns 0
00188  *
00189  * \verbatim image:MatchSize(image2: imImage) -> match: boolea
00190  * \ingroup imgclass */
00191 int imImageMatchSize(const imImage* image1, const imImage* ima
00192
00193 /** Returns 1 if the images match color mode and data type. Re
00194  *
00195  * \verbatim image:MatchColor(image2: imImage) -> match: boole
00196  * \ingroup imgclass */
00197 int imImageMatchColor(const imImage* image1, const imImage* im
00198
00199 /** Returns 1 if the images match width, height and data type.
00200  *
00201  * \verbatim image:MatchDataType(image2: imImage) -> match: bo
00202  * \ingroup imgclass */
00203 int imImageMatchDataType(const imImage* image1, const imImage*
00204
00205 /** Returns 1 if the images match width, height and color spac
00206  *
00207  * \verbatim image:MatchColorSpace(image2: imImage) -> match:
00208  * \ingroup imgclass */
00209 int imImageMatchColorSpace(const imImage* image1, const imImag
00210
00211 /** Returns 1 if the images match in width, height, data type
00212  *
00213  * \verbatim image:Match(image2: imImage) -> match: boolean [i
00214  * \ingroup imgclass */
00215 int imImageMatch(const imImage* image1, const imImage* image2)
00216
00217 /** Changes the image space from gray to binary by just changi
00218  *
00219  * \verbatim image:SetBinary() [in Lua 5] \endverbatim
00220  * \ingroup imgclass */
00221 void imImageSetBinary(imImage* image);
00222
00223 /** Changes a gray data into a binary data, done in-place.
00224  *
00225  * \verbatim image:MakeBinary() [in Lua 5] \endverbatim
00226  * \ingroup imgclass */
00227 void imImageMakeBinary(imImage *image);

```

```

00228
00229
00230
00231 /** \defgroup imgfile imImage Storage
00232 *
00233 * \par
00234 * Functions to simplify the process of reading and writing
00235 * Will also load and save the alpha planes when possible.
00236 * \par
00237 * See \ref im_image.h
00238 * \ingroup file */
00239
00240
00241 /** Loads an image from an already open file. Returns NULL if
00242 * This will call \ref imFileReadImageInfo and \ref imFileRead
00243 * index specifies the image number between 0 and image_count-
00244 * The returned imagem will be of the same color_space and dat
00245 *
00246 * \verbatim ifile:ImageLoad([index: number]) -> image: imImag
00247 * Default index is 0.
00248 * \ingroup imgfile */
00249 imImage* imFileLoadImage(imFile* ifile, int index, int *error)
00250
00251 /** Loads an image from an already open file. Returns NULL if
00252 * This function assumes that the image in the file has the sa
00253 * This will call \ref imFileReadImageInfo and \ref imFileRead
00254 * index specifies the image number between 0 and image_count-
00255 * The returned imagem will be of the same color_space and dat
00256 *
00257 * \verbatim ifile:ImageLoadFrame([index: number], image: imIm
00258 * Default index is 0.
00259 * \ingroup imgfile */
00260 void imFileLoadImageFrame(imFile* ifile, int index, imImage* i
00261
00262 /** Loads an image from an already open file, but forces the i
00263 * The returned imagem will be always a Bitmap image, with col
00264 * index specifies the image number between 0 and image_count-
00265 * Returns NULL if failed.
00266 *
00267 * \verbatim ifile:LoadBitmap([index: number]) -> image: imIma
00268 * Default index is 0.
00269 * \ingroup imgfile */
00270 imImage* imFileLoadBitmap(imFile* ifile, int index, int *error
00271
00272 /** Loads an image from an already open file, but forces the i
00273 * This function assumes that the image in the file has the sa
00274 * The imagem must be a Bitmap image, with color_space RGB, MA

```

```

00275 * index specifies the image number between 0 and image_count-
00276 * Returns NULL if failed.
00277 *
00278 * \verbatim ifile:LoadBitmapFrame([index: number], image: imI
00279 * Default index is 0.
00280 * \ingroup imgfile */
00281 void imFileLoadBitmapFrame(imFile* ifile, int index, imImage*
00282
00283 /** Saves the image to an already open file. \n
00284 * This will call \ref imFileWriteImageInfo and \ref imFilewri
00285 * Returns error code.
00286 *
00287 * \verbatim ifile:SaveImage(image: imImage) -> error: number
00288 * \ingroup imgfile */
00289 int imFileSaveImage(imFile* ifile, const imImage* image);
00290
00291 /** Loads an image from file. Open, loads and closes the file.
00292 * index specifies the image number between 0 and image_count-
00293 * Returns NULL if failed.
00294 *
00295 * \verbatim im.FileImageLoad(file_name: string, [index: numbe
00296 * Default index is 0.
00297 * \ingroup imgfile */
00298 imImage* imFileImageLoad(const char* file_name, int index, int
00299
00300 /** Loads an image from file, but forces the image to be a bit
00301 * index specifies the image number between 0 and image_count-
00302 * Returns NULL if failed.
00303 *
00304 * \verbatim im.FileImageLoadBitmap(file_name: string, [index:
00305 * Default index is 0.
00306 * \ingroup imgfile */
00307 imImage* imFileImageLoadBitmap(const char* file_name, int inde
00308
00309 /** Saves the image to file. Open, saves and closes the file.
00310 * Returns error code.
00311 *
00312 * \verbatim im.FileImageLoadBitmap(file_name: string, format:
00313 * \ingroup imgfile */
00314 int imFileImageSave(const char* file_name, const char* format,
00315
00316
00317
00318 /** Utility macro to draw the image in a CD library canvas.
00319 * Works only for data_type IM_BYTE, and color spaces: IM_RGB,
00320 * \ingroup imgclass */
00321 #define imPutBitmap(_image, _x, _y, _w, _h, _xmin, _xmax, _ymi

```

```
00322 {
00323   if (_image->color_space == IM_RGB)
00324   {
00325     if (image->has_alpha)
00326       cdPutImageRectRGBA(_image->width, _image->height,
00327                          (unsigned char*)_image->data[0],
00328                          (unsigned char*)_image->data[1],
00329                          (unsigned char*)_image->data[2],
00330                          (unsigned char*)_image->data[3],
00331                          _x, _y, _w, _h, _xmin, _xmax, _ymin,
00332     else
00333       cdPutImageRectRGB(_image->width, _image->height,
00334                         (unsigned char*)_image->data[0],
00335                         (unsigned char*)_image->data[1],
00336                         (unsigned char*)_image->data[2],
00337                         _x, _y, _w, _h, _xmin, _xmax, _ymin,
00338     }
00339   else
00340     cdPutImageRectMap(_image->width, _image->height,
00341                      (unsigned char*)_image->data[0], _imag
00342                      _x, _y, _w, _h, _xmin, _xmax, _ymin, _
00343   }
00344
00345
00346 #if defined(__cplusplus)
00347 }
00348 #endif
00349
00350 #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: im_plus.h,v 1.1 2005/04/02 22:07:00 scuri 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 cheked 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 cou
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_cou
00050     { imFileGetInfo(this->ifile, format, compression, image_co
00051
00052 void ReadImageInfo(int index, int *width, int *height, int *
00053     { this->error = imFileReadImageInfo(this->ifile, index, wi
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
00059     { this->error = imFileReadImageData(this->ifile, data, con
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, i
00068     { this->error = imFileWriteImageInfo(this->ifile, width, h
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: im.h,v 1.5 2005/12/12 13:31:08 scuri 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/extra 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 (rgbrgbrgb...)
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,     /**< Insuficient 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  *
00077  * \verbatim im.FileOpen(file_name: string) -> ifile: imFile,
00078  * \ingroup file */
00079  imFile* imFileOpen(const char* file_name, int *error);
00080
00081  /** Creates a new file for writing. If the file exists will be
00082  * It will only initialize the format driver and create the fi
00083  *
00084  * \verbatim im.FileNew(file_name: string, format: string) ->
00085  * \ingroup file */
00086  imFile* imFileNew(const char* file_name, const char* format, i

```

```

00087
00088 /** Closes the file
00089  *
00090  * \verbatim im.FileClose(ifile: imFile) [in Lua 5] \endverbat
00091  * \verbatim ifile:Close() [in Lua 5] \endverbatim
00092  * \ingroup file */
00093 void imFileClose(imFile* ifile);
00094
00095 /** Returns the internal handle. It is file format dependent.
00096  *
00097  * \verbatim ifile:Handle() -> handle: userdata [in Lua 5] \en
00098  * \ingroup file */
00099 void* imFileHandle(imFile* ifile);
00100
00101 /** Returns file information.
00102  * image_count is the number of images in a stack or
00103  * the number of frames in a video/animation or the depth of
00104  * compression and image_count can be NULL.
00105  *
00106  * \verbatim ifile:GetInfo() -> format: string, compression: s
00107  * \ingroup file */
00108 void imFileGetInfo(imFile* ifile, char* format, char* compress
00109
00110 /** Changes the write compression method. \n
00111  * If the compression is not supported will return an error co
00112  * Use NULL to set the default compression. You can use the im
00113  * but only after \ref imFileWriteImageInfo. Only a few format
00114  *
00115  * \verbatim ifile:SetInfo(compression: string) [in Lua 5] \en
00116  * \ingroup file */
00117 void imFileSetInfo(imFile* ifile, const char* compression);
00118
00119 /** Changes an extended attribute. \n
00120  * The data will be internally duplicated. \n
00121  * If data is NULL the attribute is removed.
00122  *
00123  * \verbatim ifile:SetAttribute(attrib: string, data_type: num
00124  * If data_type is IM_BYTE, as_string can be used as data.
00125  * \ingroup file */
00126 void imFileSetAttribute(imFile* ifile, const char* attrib, int
00127
00128 /** Returns an extended attribute. \n
00129  * Returns NULL if not found. data_type and count can be NULL.
00130  *
00131  * \verbatim ifile:GetAttribute(attrib: string, [as_string: bo
00132  * If data_type is IM_BYTE, as_string can be used to return a
00133  * \ingroup file */

```

```

00134 const void* imFileGetAttribute(imFile* ifile, const char* attr
00135
00136 /** Returns a list of the attribute names. \n
00137 * "attrib" must contain room enough for "attrib_count" names.
00138 *
00139 * \verbatim ifile:GetAttributeList() -> data: table of string
00140 * \ingroup file */
00141 void imFileGetAttributeList(imFile* ifile, char** attrib, int
00142
00143 /** Returns the palette if any. \n
00144 * "palette" must be a 256 colors allocated array. \n
00145 * Returns zero in "palette_count" if there is no palette. "pa
00146 *
00147 * \verbatim ifile:GetPalette() -> palette: imPalette [in Lua
00148 * \ingroup file */
00149 void imFileGetPalette(imFile* ifile, long* palette, int *palet
00150
00151 /** Changes the palette. \n
00152 * "palette_count" is >0 and <=256.
00153 *
00154 * \verbatim ifile:SetPalette(palette: imPalette) [in Lua 5] \
00155 * \ingroup file */
00156 void imFileSetPalette(imFile* ifile, long* palette, int palett
00157
00158 /** Reads the image header if any and returns image informatio
00159 * Reads also the extended image attributes, so other image at
00160 * Returns an error code.
00161 * index specifies the image number between 0 and image_count-
00162 * Some drivers reads only in sequence, so "index" can be igno
00163 * Any parameters can be NULL. This function must be called at
00164 *
00165 * \verbatim ifile:ReadImageInfo([index: number]) -> error: nu
00166 * Default index is 0.
00167 * \ingroup file */
00168 int imFileReadImageInfo(imFile* ifile, int index, int *width,
00169
00170 /** Writes the image header. Writes the file header at the fir
00171 * Writes also the extended image attributes. \n
00172 * Must call imFileSetPalette and set other attributes before
00173 * In some formats the color space will be converted to match
00174 * Returns an error code. This function must be called at leas
00175 *
00176 * \verbatim ifile:WriteImageInfo(width: number, height: numbe
00177 * \ingroup file */
00178 int imFileWriteImageInfo(imFile* ifile, int width, int height,
00179
00180 /** Reads the image data with or without conversion. \n

```

```

00181 * The data can be converted to bitmap when reading.
00182 * Data type conversion to byte will always scan for min-max t
00183 * except integer values that min-max are already between 0-25
00184 * Color mode flags contains packed, alpha and top-botttom inf
00185 * If flag is 0 means unpacked, no alpha and bottom up. If fla
00186 * Returns an error code.
00187 *
00188 * \verbatim ifile:ReadImageData(data: userdata, convert2bitma
00189 * \ingroup file */
00190 int imFileReadImageData(imFile* ifile, void* data, int convert
00191
00192 /** Writes the image data. \n
00193 * Returns an error code.
00194 *
00195 * \verbatim ifile:WriteImageData(data: userdata) -> error: nu
00196 * \ingroup file */
00197 int imFileWriteImageData(imFile* ifile, void* data);
00198
00199
00200
00201 /** Returns a list of the registered formats. \n
00202 * format_list is an array of format identifiers.
00203 * Each format identifier is 10 chars max, maximum of 50 forma
00204 * You can use "char* format_list[50]".
00205 *
00206 * \verbatim im.FormatList() -> format_list: table of strings
00207 * \ingroup format */
00208 void imFormatList(char** format_list, int *format_count);
00209
00210 /** Returns the format description. \n
00211 * Format description is 50 chars max. \n
00212 * Extensions are separated like "*.tif;*.tiff;", 50 chars max
00213 * Returns an error code. The parameters can be NULL, except f
00214 *
00215 * \verbatim im.FormatInfo(format: string) -> error: number, d
00216 * \ingroup format */
00217 int imFormatInfo(const char* format, char* desc, char* ext, in
00218
00219 /** Returns the format compressions. \n
00220 * Compressions are 20 chars max each, maximum of 50 compressi
00221 * color_mode and data_type are optional, use -1 to ignore the
00222 * If you use them they will select only the allowed compressi
00223 * Returns an error code.
00224 *
00225 * \verbatim im.FormatCompressions(format: string, [color_mode
00226 * \ingroup format */
00227 int imFormatCompressions(const char* format, char** comp, int

```

```
00228
00229 /** Checks if the format suport the given image class at the g
00230  * Returns an error code.
00231  *
00232  * \verbatim im.FormatCanWriteImage(format: string, compressio
00233  * \ingroup format */
00234 int imFormatCanWriteImage(const char* format, const char* comp
00235
00236
00237 #if defined(__cplusplus)
00238 }
00239 #endif
00240
00241 #include "old_im.h"
00242
00243 #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: im_attrib_flat.h,v 1.1 2005/04/02 22:07:00 scuri 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 ch
00028 void imAttribTableCopyFrom(imAttribTablePrivate* ptable_dst, c
00029 void imAttribTableForEach(const imAttribTablePrivate* ptable, c
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: im_capture.h,v 1.5 2005/12/11 23:41:25 scuri 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  * Functions to capture images from live video devices.
00018  * \par
00019  * See \ref im_capture.h
00020  */
00021
00022 typedef struct _imVideoCapture imVideoCapture;
00023
00024 /** Returns the number of available devices.
00025  *
00026  * \verbatim im.VideoCaptureDeviceCount() -> count: number [in
00027  * \ingroup capture */
00028 int imVideoCaptureDeviceCount(void);
00029
00030 /** Returns the device description. Returns NULL in the last d
00031  *
00032  * \verbatim im.VideoCaptureDeviceDesc(device: number) -> desc
00033  * \ingroup capture */
00034 const char* imVideoCaptureDeviceDesc(int device);
00035
00036 /** Reload the device list. The devices can be dynamically rem
00037  * Returns the number of available devices.
00038  *
00039  * \verbatim im.imVideoCaptureReloadDevices() -> count: number
```

```

00040 * \ingroup capture */
00041 int imVideoCaptureReloadDevices(void);
00042
00043 /** Creates a new imVideoCapture object. \n
00044 * Returns NULL if there is no capture device available. \n
00045 * In Windows returns NULL if DirectX version is older than 8.
00046 *
00047 * \verbatim im.VideoCaptureCreate() -> vc: imVideoCapture [in
00048 * \ingroup capture */
00049 imVideoCapture* imVideoCaptureCreate(void);
00050
00051 /** Destroys a imVideoCapture object.
00052 *
00053 * \verbatim im.VideoCaptureDestroy(vc: imVideoCapture) [in Lu
00054 * \verbatim vc:Destroy() [in Lua 5] \endverbatim
00055 * \ingroup capture */
00056 void imVideoCaptureDestroy(imVideoCapture* vc);
00057
00058 /** Connects to a capture device.
00059 * More than one imVideoCapture object can be created
00060 * but they must be connected to different devices. \n
00061 * If the object is conected it will disconnect first. \n
00062 * Use -1 to return the current connected device,
00063 * in this case returns -1 if not connected. \n
00064 * Returns zero if failed.
00065 *
00066 * \verbatim vc:Connect([device: number]) -> ret: number [in L
00067 * \ingroup capture */
00068 int imVideoCaptureConnect(imVideoCapture* vc, int device);
00069
00070 /** Disconnect from a capture device.
00071 *
00072 * \verbatim vc:Disconnect() [in Lua 5] \endverbatim
00073 * \ingroup capture */
00074 void imVideoCaptureDisconnect(imVideoCapture* vc);
00075
00076 /** Returns the number of available configuration dialogs.
00077 *
00078 * \verbatim vc:DialogCount() -> count: number [in Lua 5] \end
00079 * \ingroup capture */
00080 int imVideoCaptureDialogCount(imVideoCapture* vc);
00081
00082 /** Displays a configuration modal dialog of the connected dev
00083 * In Windows, the capturing will be stopped in some cases. \n
00084 * In Windows parent is a HWND of a parent window, it can be N
00085 * dialog can be from 0 to \ref imVideoCaptureDialogCount. \n
00086 * Returns zero if failed.

```

```

00087 *
00088 * \verbatim vc:ShowDialog(dialog: number, parent: userdata) -
00089 * \ingroup capture */
00090 int imVideoCaptureShowDialog(imVideoCapture* vc, int dialog, v
00091
00092 /** Returns the description of a configuration dialog.
00093 * dialog can be from 0 to \ref imVideoCaptureDialogCount. \n
00094 *
00095 * \verbatim vc:DialogDesc(dialog: number) -> desc: string [in
00096 * \ingroup capture */
00097 const char* imVideoCaptureDialogDesc(imVideoCapture* vc, int d
00098
00099 /** Returns the number of available video formats. \n
00100 * Returns zero if failed.
00101 *
00102 * \verbatim vc:FormatCount() -> error: number [in Lua 5] \end
00103 * \ingroup capture */
00104 int imVideoCaptureFormatCount(imVideoCapture* vc);
00105
00106 /** Returns information about the video format. \n
00107 * format can be from 0 to \ref imVideoCaptureFormatCount. \n
00108 * desc should be of size 10. \n
00109 * The image size is usually the maximum size for that format.
00110 * Other sizes can be available using \ref imVideoCaptureSetIm
00111 * Returns zero if failed.
00112 *
00113 * \verbatim vc:GetFormat(format: number) -> error: number, wi
00114 * \ingroup capture */
00115 int imVideoCaptureGetFormat(imVideoCapture* vc, int format, in
00116
00117 /** Changes the video format of the connected device. \n
00118 * Should NOT work for DV devices. Use \ref imVideoCaptureSetI
00119 * Use -1 to return the current format, in this case returns -
00120 * When the format is changed in the dialog, for some formats
00121 * the returned format is the preferred format, not the curren
00122 * This will not affect color_mode of the capture image. \n
00123 * Returns zero if failed.
00124 *
00125 * \verbatim vc:SetFormat(format: number) -> error: number [in
00126 * \ingroup capture */
00127 int imVideoCaptureSetFormat(imVideoCapture* vc, int format);
00128
00129 /** Returns the current image size of the connected device. \n
00130 * width and height returns 0 if not connected.
00131 *
00132 * \verbatim vc:GetImageSize() -> width: number, height: numbe
00133 * \ingroup capture */

```

```

00134 void imVideoCaptureGetImageSize(imVideoCapture* vc, int *width
00135
00136 /** Changes the image size of the connected device. \n
00137 * Similar to \ref imVideoCaptureSetFormat, but changes only t
00138 * Valid sizes can be obtained with \ref imVideoCaptureGetFor
00139 * Returns zero if failed.
00140 *
00141 * \verbatim vc:SetImageSize(width: number, height: number) ->
00142 * \ingroup capture */
00143 int imVideoCaptureSetImageSize(imVideoCapture* vc, int width,
00144
00145 /** Returns a new captured frame. Use -1 for infinite timeout.
00146 * Color space can be IM_RGB or IM_GRAY, and mode can be packe
00147 * It can not have an alpha channel and orientation is always
00148 * Returns zero if failed or timeout expired, the buffer is no
00149 *
00150 * \verbatim vc:Frame(image: imImage, timeout: number) -> erro
00151 * \ingroup capture */
00152 int imVideoCaptureFrame(imVideoCapture* vc, unsigned char* dat
00153
00154 /** Start capturing, returns the new captured frame and stop c
00155 * This is more usefull if you are switching between devices.
00156 * Data format is the same as imVideoCaptureFrame. \n
00157 * Returns zero if failed.
00158 *
00159 * \verbatim vc:OneFrame(image: imImage) -> error: number [in
00160 * \ingroup capture */
00161 int imVideoCaptureOneFrame(imVideoCapture* vc, unsigned char*
00162
00163 /** Start capturing. \n
00164 * Use -1 to return the current state. \n
00165 * Returns zero if failed.
00166 *
00167 * \verbatim vc:Live(live: number) -> error: number [in Lua 5]
00168 * \ingroup capture */
00169 int imVideoCaptureLive(imVideoCapture* vc, int live);
00170
00171 /** Resets a camera or video attribute to the default value or
00172 * to the automatic setting. \n
00173 * Not all attributes support automatic modes. \n
00174 * Returns zero if failed.
00175 *
00176 * \verbatim vc:ResetAttribute(attrib: string, fauto: number)
00177 * \ingroup capture */
00178 int imVideoCaptureResetAttribute(imVideoCapture* vc, const cha
00179
00180 /** Returns a camera or video attribute in percentage of the v

```

```

00181 * Returns zero if failed.
00182 *
00183 * \verbatim vc:GetAttribute(attrib: string) -> error: number,
00184 * \ingroup capture */
00185 int imVideoCaptureGetAttribute(imVideoCapture* vc, const char*
00186
00187 /** Changes a camera or video attribute in percentage of the v
00188 * Returns zero if failed.
00189 *
00190 * \verbatim vc:SetAttribute(attrib: string, percent: number)
00191 * \ingroup capture */
00192 int imVideoCaptureSetAttribute(imVideoCapture* vc, const char*
00193
00194 /** Returns a list of the description of the valid attributes.
00195 *
00196 * \verbatim vc:GetAttributeList() -> attrib_list: table of st
00197 * \ingroup capture */
00198 const char** imVideoCaptureGetAttributeList(imVideoCapture* vc
00199
00200
00201 /** \defgroup winattrib Windows Attributes Names
00202 \verbatim
00203 VideoBrightness - Specifies the brightness, also called the
00204 VideoContrast - Specifies the contrast, expressed as gain fa
00205 VideoHue - Specifies the hue angle.
00206 VideoSaturation - Specifies the saturation.
00207 VideoSharpness - Specifies the sharpness.
00208 VideoGamma - Specifies the gamma.
00209 VideoColorEnable - Specifies the color enable setting. (0/100)
00210 VideoWhiteBalance - Specifies the white balance, as a color
00211 VideoBacklightCompensation - Specifies the backlight compens
00212 VideoGain - Specifies the gain adjustment.
00213 CameraPanAngle - Specifies the camera's pan angle. To 100 ro
00214 CameraTiltAngle - Specifies the camera's tilt angle. To 100
00215 CameraRollAngle - Specifies the camera's roll angle. To 100
00216 CameraLensZoom - Specifies the camera's zoom setting.
00217 CameraExposure - Specifies the exposure setting.
00218 CameraIris - Specifies the camera's iris setting.
00219 CameraFocus - Specifies the camera's focus setting, as the d
00220 FlipHorizontal - Specifies the video will be flipped in the
00221 FlipVertical - Specifies the video will be flipped in the ve
00222 AnalogFormat - Specifies the video format standard NTSC, PAL
00223     NTSC_M     = 0
00224     NTSC_M_J   = 1
00225     NTSC_433   = 2
00226     PAL_B      = 3
00227     PAL_D      = 4

```

```

00228     PAL_H       = 5
00229     PAL_I       = 6
00230     PAL_M       = 7
00231     PAL_N       = 8
00232     PAL_60     = 9
00233     SECAM_B    = 10
00234     SECAM_D    = 11
00235     SECAM_G    = 12
00236     SECAM_H    = 13
00237     SECAM_K    = 14
00238     SECAM_K1   = 15
00239     SECAM_L    = 16
00240     SECAM_L1   = 17
00241     PAL_N_COMBO = 18
00242 \endverbatim
00243 * \ingroup capture */
00244
00245
00246 #if defined(__cplusplus)
00247 }
00248
00249 /** A C++ Wrapper for the imVideoCapture structure functions.
00250 * \ingroup capture */
00251 class imCapture
00252 {
00253 public:
00254     imCapture()
00255     { vc = imVideoCaptureCreate(); }
00256
00257     ~imCapture()
00258     { if (vc) imVideoCaptureDestroy(vc); }
00259
00260     int Failed()
00261     { if (!vc) return 0; else return 1; }
00262
00263     int Connect(int device)
00264     { return imVideoCaptureConnect(vc, device); }
00265
00266     void Disconnect()
00267     { imVideoCaptureDisconnect(vc); }
00268
00269     int DialogCount()
00270     { return imVideoCaptureDialogCount(vc); }
00271
00272     int ShowDialog(int dialog, void* parent)
00273     { return imVideoCaptureShowDialog(vc, dialog, parent); }
00274

```

```
00275 const char* DialogDescription(int dialog)
00276     { return imVideoCaptureDialogDesc(vc, dialog); }
00277
00278 int FormatCount()
00279     { return imVideoCaptureFormatCount(vc); }
00280
00281 int GetFormat(int format, int *width, int *height, char* des
00282     { return imVideoCaptureGetFormat(vc, format, width, height
00283
00284 int SetFormat(int format)
00285     { return imVideoCaptureSetFormat(vc, format); }
00286
00287 void GetImageSize(int *width, int *height)
00288     { imVideoCaptureGetImageSize(vc, width, height); }
00289
00290 int SetImageSize(int width, int height)
00291     { return imVideoCaptureSetImageSize(vc, width, height); }
00292
00293 int GetFrame(unsigned char* data, int color_mode, int timeou
00294     { return imVideoCaptureFrame(vc, data, color_mode, timeout
00295
00296 int GetOneFrame(unsigned char* data, int color_mode)
00297     { return imVideoCaptureOneFrame(vc, data, color_mode); }
00298
00299 int Live(int live)
00300     { return imVideoCaptureLive(vc, live); }
00301
00302 int ResetAttribute(const char* attrib, int fauto)
00303     { return imVideoCaptureResetAttribute(vc, attrib, fauto); }
00304
00305 int GetAttribute(const char* attrib, float *percent)
00306     { return imVideoCaptureGetAttribute(vc, attrib, percent); }
00307
00308 int SetAttribute(const char* attrib, float percent)
00309     { return imVideoCaptureSetAttribute(vc, attrib, percent); }
00310
00311 const char** GetAttributeList(int *num_attrib)
00312     { return imVideoCaptureGetAttributeList(vc, num_attrib); }
00313
00314 protected:
00315     imVideoCapture* vc;
00316 };
00317
00318 #endif
00319
00320 #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: im_color.h,v 1.1 2005/04/02 22:07:00 scuri 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 gammut 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 aproximates 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 aproximates lightnes
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 distante 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 saturat
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 Vid
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 (use
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 s
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 uns
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_FWLAB(_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). \n
00241  * The white point is D65. \n
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_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 }
00273
00274 #define IM_GWLAB(_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     float fY = (L + 0.16f) / 1.16f;
00349     Y = IM_GWLAB(fY) * 1.0f;
00350
00351     float u1 = 0.1978f, v1 = 0.4683f;
00352     if (L != 0)
00353     {
00354         u1 = u / (6.5f * L) + 0.1978f;
00355         v1 = v / (6.5f * L) + 0.4683f;
00356     }
00357
00358     X = ((9 * u1) / (4 * v1)) * Y;
00359     Z = ((12 - 3 * u1 - 20 * v1) / (4 * v1)) * Y;
00360 }
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 floa
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_colorhsi.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: im_colorhsi.h,v 1.2 2005/07/15 19:59:21 scuri 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 coord
00020  * \par
00021  * "I" is defined along the cube diagonal. It ranges from 0 (b
00022  * HS are the polar coordinates of a plane normal to "I". \n
00023  * "S" is the normal distance from the diagonal of the RGB cub
00024  * "H" is the angle starting from the red vector, given in deg
00025  * \par
00026  * This is not a new color space, this is exactly the same gam
00027  * Since it is still a cube, Smax depends on H.
00028  * \par
00029  * See \ref im_colorhsi.h
00030  * \ingroup color */
00031
00032
00033 /** Returns the maximum S for H (here in radians) and I.
00034  * \ingroup hsi */
00035 float imColorHSI_Smax(float h, double cosh, double sinh, float
00036
00037 /** Returns I where S is maximum given H (here in radians).
00038  * \ingroup hsi */
00039 float imColorHSI_ImaxS(float h, double cosh, double sinh);
```

```
00040
00041 /** Converts from RGB to HSI.
00042  * \ingroup hsi */
00043 void imColorRGB2HSI(float r, float g, float b, float *h, float
00044
00045 /** Converts from RGB (byte) to HSI.
00046  * \ingroup hsi */
00047 void imColorRGB2HSIbyte(unsigned char r, unsigned char g, unsi
00048
00049 /** Converts from HSI to RGB.
00050  * \ingroup hsi */
00051 void imColorHSI2RGB(float h, float s, float i, float *r, float
00052
00053 /** Converts from HSI to RGB (byte).
00054  * \ingroup hsi */
00055 void imColorHSI2RGBbyte(float h, float s, float i, unsigned ch
00056
00057
00058 #if defined(__cplusplus)
00059 }
00060 #endif
00061
00062 #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: im_convert.h,v 1.3 2005/12/10 00:54:39 scuri 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 imgclass */
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  *
00065  * \verbatim im.ConvertDataType(src_image: imImage, dst_image:
00066  * \ingroup convert */
00067 int imConvertDataType(const imImage* src_image, imImage* dst_i
00068
00069 /** Converts one color space to another. Images must be of the
00070  * CMYK can be converted to RGB only, and it is a very simple
00071  * All colors can be converted to Binary, the non zero gray va
00072  * RGB to Map uses the median cut implementation from the free
00073  * All other color space conversions assume sRGB and CIE defin
00074  * Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.
00075  *
00076  * \verbatim im.ConvertColorSpace(src_image: imImage, dst_imag
00077  * \ingroup convert */
00078 int imConvertColorSpace(const imImage* src_image, imImage* dst
00079
00080 /** Converts the image to its bitmap equivalent,
00081  * uses \ref imConvertColorSpace and \ref imConvertDataType. \
00082  * Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER.
00083  *
00084  * \verbatim im.ConvertToBitmap(src_image: imImage, dst_image:
00085  * \ingroup convert */
00086 int imConvertToBitmap(const imImage* src_image, imImage* dst_i

```

```

00087
00088
00089
00090 /** \defgroup cnvutil Raw Data Conversion Utilities
00091 * \par
00092 * Utilities for raw data buffers.
00093 * \par
00094 * See \ref im_convert.h
00095 * \ingroup imagerep */
00096
00097
00098 /** Changes the packing of the data buffer.
00099 * \ingroup cnvutil */
00100 void imConvertPacking(const void* src_data, void* dst_data, in
00101
00102 /** Changes in-place a MAP data into a RGB data. The data must
00103 * depth can be 3 or 4. count=width*height. \n
00104 * Very usefull for OpenGL applications.
00105 * \ingroup cnvutil */
00106 void imConvertMapToRGB(unsigned char* data, int count, int dep
00107
00108
00109
00110 /* Converts a RGB bitmap into a map bitmap using the median cu
00111 * Used only "im_convertcolor.cpp" implemented in "im_rgb2map.
00112 * Internal function kept here because of the compatibility mo
00113 * Will not be at the documentation. */
00114 int imConvertRGB2Map(int width, int height,
00115                     unsigned char *red, unsigned char *green, unsign
00116                     unsigned char *map, long *palette, int *palette_
00117
00118
00119 #if defined(__cplusplus)
00120 }
00121 #endif
00122
00123 #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: im_counter.h,v 1.1 2005/04/02 22:07:00 scuri 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 diferrent counters. \n
00027  * Progress in a count reports a value from 0 to 1000. If -1 i
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, imCoun
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 th
```

```
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: im_format_all.h,v 1.8 2005/12/07 20:28:27 scuri 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 kept 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 3.7.4 \
00029  * http://www.remotesensing.org/libtiff/ \
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 YBCBR]
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, ush
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, LZW compression is enabled. LZ
00095         libGeoTIFF can be used without XTIFF initialization. Use
00096
00097     Changes:
00098         "tiff_jpeg.c" - commented "downsampled_output = TRUE" in
00099         "tiff_ojpeg.c" for boolean type compilation.
00100         New file "tif_config.h" to match our needs.
00101         New file "tiff_binfile.c" that implement I/O routines usi
00102 \endverbatim
00103 * \ingroup format */
00104 void imFormatRegisterTIFF(void);
00105
00106 /** \defgroup jpeg JPEG - JPEG File Interchange Format
00107 * \section Description
00108 *
00109 * \par
00110 * ISO/IEC 10918 (1994, 1995, 1997, 1999)\n
00111 * http://www.jpeg.org/
00112 * \par
00113 * Access to the JPEG file format uses libJPEG version 6b. \n
00114 * http://www.ijg.org \n
00115 * Copyright (C) 1991-1998, Thomas G. Lane \n
00116 * from the Independent JPEG Group.
00117 * \par
00118 * Access to the EXIF attributes uses libEXIF version 0.6.12.
00119 * http://sourceforge.net/projects/libexif
00120 * Copyright (C) 2001-2003, Lutz Müller
00121 *
00122 * \section Features
00123 *
00124 \verbatim
00125     Data Types: Byte
00126     Color Spaces: Gray, RGB, CMYK and YCbCr (Binary Saved as G
00127     Compressions:
00128         JPEG - ISO JPEG [default]
00129     Only one image.
00130     No alpha channel.
00131     Internally the components are always packed.
00132     Internally the lines are arranged from top down to bottom.
00133     Handle() returns jpeg_decompress_struct* when reading, and

```

```

00134         jpeg_compress_struct* when writing.
00135
00136     Attributes:
00137         AutoYCbCr IM_INT (1) (controls YCbCr auto conversion) de
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         jdohuf.c - added support for OJPEG_SUPPORT in libTIFF.
00150         new file created: jconfig.h
00151
00152     Changes to libEXIF:
00153         new file config.h
00154         changed "exif-tag.c" to add new function
00155         changed "exif-entry.c" to improve exif_entry_initialize
00156         fixed small bug in "mnote-pentax-tag.h".
00157
00158     Comments:
00159         Other APPx markers are ignored.
00160         No thumbnail support.
00161         RGB images are automatically converted to YCbCr when sav
00162         Also YcbCr are converted to RGB when loaded. Use AutoYCb
00163 \endverbatim
00164 * \ingroup format */
00165 void imFormatRegisterJPEG(void);
00166
00167 /** \defgroup png PNG - Portable Network Graphic Format
00168 * \section Description
00169 *
00170 * \par
00171 * Access to the PNG file format uses libPNG version 1.2.8. \n
00172 * http://www.libpng.org \n
00173 * Copyright (C) 1998-2004 Glenn Randers-Pehrson
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 colori
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) an
00220     Changed pngconf.h to use int instead of long in png_uint
00221 \endverbatim
00222 * \ingroup format */
00223 void imFormatRegisterPNG(void);
00224
00225 /** \defgroup gif GIF - Graphics Interchange Format
00226 * \section Description
00227 *

```

```

00228 * \par
00229 * Copyright (c) 1987,1988,1989,1990 CompuServe Incorporated.
00230 * GIF is a Service Mark property of CompuServe Incorporated.
00231 * Graphics Interchange Format Programming Reference, 1990. \n
00232 * LZW Copyright Unisys.
00233 * \par
00234 * Patial Internal Implementation. \n
00235 * Decoding and encoding code were extracted from GIFLib 1.0.
00236 * Copyright (c) 1989 Gershon Elber.
00237 *
00238 * \section Features
00239 *
00240 \verbatim
00241     Data Types: Byte
00242     Color Spaces: MAP only, (Gray and Binary saved as MAP)
00243     Compressions:
00244         LZW - Lempel-Ziv & Welch          [default]
00245     Can have more than one image.
00246     No alpha channel.
00247     Internally the lines are arranged from top down to bottom.
00248     Handle() returns a imBinFile* pointer.
00249
00250     Attributes:
00251         ScreenHeight, ScreenWidth IM_USHORT (1) screen size [def
00252         Interlaced IM_INT (1 | 0) default 0
00253         Description (string)
00254         TransparencyIndex IM_BYTE (1)
00255         XScreen, YScreen IM_USHORT (1) screen position
00256         UserInput IM_BYTE (1) [1, 0]
00257         Disposal (string) [UNDEF, LEAVE, RBACK, RPREV]
00258         Delay IM_USHORT (1)
00259         Iterations IM_USHORT (1) (NETSCAPE2.0 Application Extens
00260
00261     Comments:
00262         Attributes after the last image are ignored.
00263         Reads GIF87 and GIF89, but writes GIF89 always.
00264         Ignored attributes: Background Color Index, Pixel Aspect
00265                             Plain Text Extensions, Application E
00266 \endverbatim
00267 * \ingroup format */
00268 void imFormatRegisterGIF(void);
00269
00270 /** \defgroup bmp BMP - Windows Device Independent Bitmap
00271 * \section Description
00272 *
00273 * \par
00274 * Windows Copyright Microsoft Corporation.

```

```

00275 * \par
00276 * Internal Implementation.
00277 *
00278 * \section Features
00279 *
00280 \verbatim
00281     Data Types: Byte
00282     Color Spaces: RGB, MAP and Binary (Gray saved as MAP)
00283     Compressions:
00284         NONE - no compression [default]
00285         RLE - Run Length Encoding (only for MAP and Gray)
00286     Only one image.
00287     Can have an alpha channel (only for RGB)
00288     Internally the components are always packed.
00289     Lines arranged from top down to bottom or bottom up to top
00290     Handle() returns imBinFile* pointer.
00291
00292     Attributes:
00293         ResolutionUnit (string) ["DPC", "DPI"]
00294         XResolution, YResolution IM_FLOAT (1)
00295
00296     Comments:
00297         Reads OS2 1.x and Windows 3, but writes Windows 3 always
00298         Version 4 and 5 BMPs are not supported.
00299 \endverbatim
00300 * \ingroup format */
00301 void imFormatRegisterBMP(void);
00302
00303 /** \defgroup ras RAS - Sun Raster File
00304 * \section Description
00305 *
00306 * \par
00307 * Copyright Sun Corporation.
00308 * \par
00309 * Internal Implementation.
00310 *
00311 * \section Features
00312 *
00313 \verbatim
00314     Data Types: Byte
00315     Color Spaces: Gray, RGB, MAP and Binary
00316     Compressions:
00317         NONE - no compression [default]
00318         RLE - Run Length Encoding
00319     Only one image.
00320     Can have an alpha channel (only for IM_RGB)
00321     Internally the components are always packed.

```

```
00322     Internally the lines are arranged from top down to bottom.
00323     Handle() returns imBinFile* pointer.
00324
00325     Attributes:
00326         none
00327 \endverbatim
00328 * \ingroup format */
00329 void imFormatRegisterRAS(void);
00330
00331 /** \defgroup led LED - IUP image in LED
00332 * \section Description
00333 *
00334 * \par
00335 * Copyright Tecgraf/PUC-Rio and PETROBRAS/CENPES.
00336 * \par
00337 * Internal Implementation.
00338 *
00339 * \section Features
00340 *
00341 \verbatim
00342     Data Types: Byte
00343     Color Spaces: MAP only (Gray and Binary saved as MAP)
00344     Compressions:
00345         NONE - no compression [default]
00346     Only one image.
00347     No alpha channel.
00348     Internally the lines are arranged from top down to bottom.
00349     Handle() returns imBinFile* pointer.
00350
00351     Attributes:
00352         none
00353
00354     Comments:
00355         LED file must start with "LEDImage = IMAGE[".
00356 \endverbatim
00357 * \ingroup format */
00358 void imFormatRegisterLED(void);
00359
00360 /** \defgroup sgi SGI - Silicon Graphics Image File Format
00361 * \section Description
00362 *
00363 * \par
00364 * SGI is a trademark of Silicon Graphics, Inc.
00365 * \par
00366 * Internal Implementation.
00367 *
00368 * \section Features
```

```
00369  *
00370  \verbatim
00371      Data Types: Byte and UShort
00372      Color Spaces: Gray and RGB (Binary saved as Gray, MAP with
00373      Compressions:
00374          NONE - no compression [default]
00375          RLE - Run Lenght Encoding
00376      Only one image.
00377      Can have an alpha channel (only for IM_RGB)
00378      Internally the components are always packed.
00379      Internally the lines are arranged from bottom up to top.
00380      Handle() returns imBinFile* pointer.
00381
00382      Attributes:
00383          Description (string)
00384  \endverbatim
00385  * \ingroup format */
00386 void imFormatRegisterSGI(void);
00387
00388 /** \defgroup pcx PCX - ZSoft Picture
00389  * \section Description
00390  *
00391  * \par
00392  * Copyright ZSoft Corporation. \n
00393  * ZSoft (1988) PCX Technical Reference Manual.
00394  * \par
00395  * Internal Implementation.
00396  *
00397  * \section Features
00398  *
00399  \verbatim
00400      Data Types: Byte
00401      Color Spaces: RGB, MAP and Binary (Gray saved as MAP)
00402      Compressions:
00403          NONE - no compression
00404          RLE - Run Lenght Encoding [default - since uncompressed
00405      Only one image.
00406      No alpha channel.
00407      Internally the components are always packed.
00408      Internally the lines are arranged from top down to bottom.
00409      Handle() returns imBinFile* pointer.
00410
00411      Attributes:
00412          ResolutionUnit (string) ["DPC", "DPI"]
00413          XResolution, YResolution IM_FLOAT (1)
00414          XScreen, YScreen IM_USHORT (1) screen position
00415
```

```
00416     Comments:
00417         Reads Versions 0-5, but writes Version 5 always.
00418 \endverbatim
00419 * \ingroup format */
00420 void imFormatRegisterPCX(void);
00421
00422 /** \defgroup tga TGA - Truevision Graphics Adapter File
00423 * \section Description
00424 *
00425 * \par
00426 * Truevision TGA File Format Specification Version 2.0 \n
00427 * Technical Manual Version 2.2 January, 1991          \n
00428 * Copyright 1989, 1990, 1991 Truevision, Inc.
00429 * \par
00430 * Internal Implementation.
00431 *
00432 * \section Features
00433 *
00434 \verbatim
00435     Supports 8 bits per component only. Data type is always By
00436     Color Spaces: Gray, RGB and MAP (Binary saved as Gray)
00437     Compressions:
00438         NONE - no compression [default]
00439         RLE - Run Length Encoding
00440     Only one image.
00441     No alpha channel.
00442     Internally the components are always packed.
00443     Internally the lines are arranged from bottom up to top or
00444     Handle() returns imBinFile* pointer.
00445
00446     Attributes:
00447         XScreen, YScreen IM_USHORT (1) screen position
00448         Title, Author, Description, JobName, Software (string)
00449         SoftwareVersion (read only) (string)
00450         DateTimeModified (string) [when writing uses the current
00451         Gamma IM_FLOAT (1)
00452 \endverbatim
00453 * \ingroup format */
00454 void imFormatRegisterTGA(void);
00455
00456 /** \defgroup pnm PNM - Netpbm Portable Image Map
00457 * \section Description
00458 *
00459 * \par
00460 * PNM formats Copyright Jef Poskanzer
00461 * \par
00462 * Internal Implementation.
```

```
00463 *
00464 * \section Features
00465 *
00466 \verbatim
00467     Data Types: Byte and UShort
00468     Color Spaces: Gray, RGB and Binary
00469     Compressions:
00470         NONE - no compression [default]
00471         ASCII (textual data)
00472     Can have more than one image, but sequential access only.
00473     No alpha channel.
00474     Internally the components are always packed.
00475     Internally the lines are arranged from top down to bottom.
00476     Handle() returns imBinFile* pointer.
00477
00478     Attributes:
00479         Description (string)
00480
00481     Comments:
00482         In fact ASCII is an expansion...
00483 \endverbatim
00484 * \ingroup format */
00485 void imFormatRegisterPNM(void);
00486
00487 /** \defgroup ico ICO - Windows Icon
00488 * \section Description
00489 *
00490 * \par
00491 * Windows Copyright Microsoft Corporation.
00492 * \par
00493 * Internal Implementation.
00494 *
00495 * \section Features
00496 *
00497 \verbatim
00498     Data Types: Byte
00499     Color Spaces: RGB, MAP and Binary (Gray saved as MAP)
00500     Compressions:
00501         NONE - no compression [default]
00502     Can have more than one image. But writing is limited to 5
00503         and all images must have different sizes and bpp.
00504     No alpha channel.
00505     Internally the components are always packed.
00506     Internally the lines are arranged from bottom up to top.
00507     Handle() returns imBinFile* pointer.
00508
00509     Attributes:
```

```

00510     TransparencyIndex IM_BYTE (1 or N)
00511
00512     Comments:
00513         If the user specifies an alpha channel, the AND mask is
00514         but the file color mode will not contain the IM_ALPHA
00515         For IM_MAP imagens, if the user does not specifies an al
00516         the TransparencyIndex is used to initialize the AND ma
00517         and the most repeated index with transparency will be
00518         Although any size and bpp can be used is recomended to u
00519         16x16, 32x32, 48x48, 64x64 or 96x96
00520         2 colors, 16 colors or 256 colors
00521 \endverbatim
00522 * \ingroup format */
00523 void imFormatRegisterICO(void);
00524
00525 /** \defgroup krn KRN - IM Kernel File Format
00526 * \section Description
00527 *
00528 * \par
00529 * Textual format to provied a simple way to create kernel con
00530 * \par
00531 * Internal Implementation.
00532 *
00533 * \section Features
00534 *
00535 \verbatim
00536     Data Types: Byte, Int
00537     Color Spaces: Gray
00538     Compressions:
00539         NONE - no compression [default]
00540     Only one image.
00541     No alpha channel.
00542     Internally the lines are arranged from top down to bottom.
00543     Handle() returns imBinFile* pointer.
00544
00545     Attributes:
00546         Description (string)
00547
00548     Comments:
00549         The format is very simple, inspired by PNM.
00550         It was developed because PNM does not have support for I
00551         Remeber that usually convolution operations use kernel s
00552
00553     Format Model:
00554         IMKERNEL
00555         Description up to 512 characters
00556         width height

```

```
00557     type (0 - IM_INT, 1 - IM_FLOAT)
00558     data...
00559
00560     Example:
00561     IMKERNEL
00562     Gradian
00563     3 3
00564     0
00565     0 -1 0
00566     0 1 0
00567     0 0 0
00568 \endverbatim
00569 * \ingroup format */
00570 void imFormatRegisterKRN(void);
00571
00572
00573 #if defined(__cplusplus)
00574 }
00575 #endif
00576
00577 #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: im_format_avi.h,v 1.4 2005/12/06 00:58:09 scuri 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  * Access to the AVI format uses Windows AVIFile library. Avail
00022  * When writing a new file you must use an ".avi" extension, o
00023  * You must link the application with "im_avi.lib"
00024  * and you must call the function \ref imFormatRegisterAVI onc
00025  * to register the format into the IM core library. \n
00026  * Depends also on the VFW library (vfw32.lib).
00027  * When using the "im_avi.dll" this extra library is not neces
00028  * If using Cygwin or MingW must link with "-lvfw32".
00029  * Old versions of Cygwin and MingW use the "-lvfw_ms32" and "
00030  * \par
00031  * See \ref im_format_avi.h
00032  *
00033  * \section Features
00034  *
00035  \verbatim
00036      Data Types: Byte
00037      Color Spaces: RGB, MAP and Binary (Gray saved as MAP)
00038      Compressions (installed in Windows XP by default):
00039          NONE      - no compression [default]
```

```

00040     RLE      - Microsoft RLE (8bpp only)
00041     CINEPACK - Cinepak Codec by Radius
00042     MSVC     - Microsoft Video 1 (old)
00043     M261     - Microsoft H.261 Video Codec
00044     M263     - Microsoft H.263 Video Codec
00045     I420     - Intel 4:2:0 Video Codec (same as M263)
00046     IV32     - Intel Indeo Video Codec 3.2 (old)
00047     IV41     - Intel Indeo Video Codec 4.5 (old)
00048     IV50     - Intel Indeo Video 5.1
00049     IYUV     - Intel IYUV Codec
00050     MPG4     - Microsoft MPEG-4 Video Codec V1 (not MPEG-4 c
00051     MP42     - Microsoft MPEG-4 Video Codec V2 (not MPEG-4 c
00052     CUSTOM  - (show compression dialog)
00053     DIVX     - DivX 5.0.4 Codec (DivX must be installed)
00054     (others, must be the 4 charaters of the fourfcc code)
00055     Can have more than one image.
00056     Can have an alpha channel (only for RGB)
00057     Internally the components are always packed.
00058     Lines arranged from top down to bottom or bottom up to top
00059     Handle() returns PAVIFILE.
00060
00061     Attributes:
00062         FPS IM_FLOAT (1) (should set when writing, default 15)
00063         AVIQuality IM_INT (1) [1-10000, default -1] (write only)
00064         KeyFrameRate IM_INT (1) (write only) [key frame frequenc
00065         DataRate IM_INT (1) (write only) [kilobits/second, defau
00066
00067     Comments:
00068         Reads only the first video stream. Other streams are ign
00069         All the images have the same size, you must call imFileR
00070         at least once.
00071         For codecs comparsion and download go to:
00072         http://graphics.lcs.mit.edu/~tbuehler/video/codecs/
00073         http://www.fourcc.org
00074 \endverbatim
00075 * \ingroup format */
00076
00077 /** Register the AVI Format
00078 * \ingroup avi */
00079 void imFormatRegisterAVI(void);
00080
00081 #if defined(__cplusplus)
00082 }
00083 #endif
00084
00085 #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: im_format_jp2.h,v 1.2 2005/04/14 19:35:39 scuri 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 we write always
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 jas_stream_create and jas
00057         Changed jp2_dec.c and jpc_cs.c to remove "uint" and "ulo
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: im_format_raw.h,v 1.1 2005/04/02 22:07:00 scuri 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: im_format_wmv.h,v 1.2 2005/08/10 02:12:16 scuri 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  * Access to the WMV format uses Windows Media SDK. Available
00023  * You must link the application with "im_wmv.lib"
00024  * and you must call the function \ref imFormatRegisterWMV onc
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 neces
00028  * \par
00029  * The application users should have the WMV codec 9 installed
00030  * http://www.microsoft.com/windows/windowsmedia/format/codecd
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.a
00037  * http://msdn.microsoft.com/library/en-us/wmform/htm/introduc
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         Sequential 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: im_lib.h,v 1.6 2005/12/13 18:58:44 scuri 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  * \ingroup util
00018  * \par
00019  * Usefull definitions for about dialogs and
00020  * for checking the compiled version with the linked version f
00021  * \par
00022  * See \ref im_lib.h
00023  * @{
00024  */
00025 #define IM_AUTHOR "Antonio Scuri"
00026 #define IM_COPYRIGHT "Copyright (C) 1994-2005 Tecgraf/PUC-Rio
00027 #define IM_VERSION "3.1.0"
00028 #define IM_VERSION_DATE "2005/12/12"
00029 /** @} */
00030
00031 /** Library release number used in the compilation time. \n
00032  * You can compare this with the value returned by \ref imVers
00033  * \ingroup lib */
00034 #define IM_VERSION_NUMBER 301000
00035
00036 /** Returns the library current version.
00037  *
00038  * \verbatim im.Version() -> version: string [in Lua 5] \endve
00039  * \ingroup lib */
```

```

00040 const char* imVersion(void);
00041
00042 /** Returns the library current version release date.
00043  *
00044  * \verbatim im.VersionDate() -> date: string [in Lua 5] \endv
00045  * \ingroup lib */
00046 const char* imVersionDate(void);
00047
00048 /** Returns the library current version number.
00049  *
00050  * \verbatim im.VersionNumber() -> version: number [in Lua 5]
00051  * \ingroup lib */
00052 int imVersionNumber(void);
00053
00054
00055 #if defined(__cplusplus)
00056 }
00057 #endif
00058
00059
00060 /*! \mainpage IM
00061  * <CENTER>
00062  * <H3> Image Representation, Storage, Capture and Processing
00063  * Tecgraf: Computer Graphics Technology Group, PUC-Rio, Brazi
00064  * http://www.tecgraf.puc-rio.br/im \n
00065  * mailto:im@tecgraf.puc-rio.br
00066  * </CENTER>
00067  *
00068  * \section over Overview
00069  * \par
00070  * IM is a toolkit for Digital Imaging.
00071  * \par
00072  * It provides support for image capture, several image file f
00073  * \par
00074  * Image representation includes scientific data types (like I
00075  * and attributes (or metadata like GeoTIFF and Exif tags).
00076  * Animation, video and volumes are supported as image sequenc
00077  * but there is no digital audio support.
00078  * \par
00079  * The main goal of the library is to provide a simple API and
00080  * of images for scientific applications.
00081  * \par
00082  * The toolkit API is written in C.
00083  * The core library source code is implemented in C++ and it i
00084  * it can be compiled in Windows and UNIX with no modification
00085  * New image processing operations can be implemented in C or
00086  * \par

```

```
00087 * IM is free software, can be used for public and commercial
00088 * \par
00089 * This work was developed at Tecgraf/PUC-Rio
00090 * by means of the partnership with PETROBRAS/CENPES.
00091 *
00092 * \section author Author
00093 * \par
00094 * Basic Software Group @ Tecgraf/PUC-Rio
00095 * - Antonio Scuri scuri@tecgraf.puc-rio.br
00096 *
00097 * \section copyright Copyright Notice
00098 \verbatim
00099
00100 *****
00101 * Copyright (C) 1994-2004 Tecgraf/PUC-Rio and PETROBRAS S/A.
00102 * All rights reserved.
00103 *
00104 * Permission is hereby granted, free of charge, to any person
00105 * a copy of this software and associated documentation files (
00106 * "Software"), to deal in the Software without restriction, in
00107 * without limitation the rights to use, copy, modify, merge, p
00108 * distribute, sublicense, and/or sell copies of the Software,
00109 * permit persons to whom the Software is furnished to do so, s
00110 * the following conditions:
00111 *
00112 * The above copyright notice and this permission notice shall
00113 * included in all copies or substantial portions of the Softwa
00114 *
00115 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KI
00116 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANT
00117 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINF
00118 * IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
00119 * CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF C
00120 * TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WIT
00121 * SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00122 *****
00123 \endverbatim
00124 */
00125
00126
00127 /** \defgroup imagerep Image Representation
00128 * \par
00129 * See \ref im.h
00130 */
00131
00132
00133 /** \defgroup file Image Storage
```

```

00134 * \par
00135 * See \ref im.h
00136 */
00137
00138
00139 /** \defgroup format File Formats
00140 * \par
00141 * See \ref im.h
00142 *
00143 * Internal Predefined File Formats:
00144 * \li "BMP" - Windows Device Independent Bitmap
00145 * \li "PCX" - ZSoft Picture
00146 * \li "GIF" - Graphics Interchange Format
00147 * \li "TIFF" - Tagged Image File Format
00148 * \li "RAS" - Sun Raster File
00149 * \li "SGI" - Silicon Graphics Image File Format
00150 * \li "JPEG" - JPEG File Interchange Format
00151 * \li "LED" - IUP image in LED
00152 * \li "TGA" - Truevision Targa
00153 * \li "RAW" - RAW File
00154 * \li "PNM" - Netpbm Portable Image Map
00155 * \li "ICO" - Windows Icon
00156 * \li "PNG" - Portable Network Graphic Format
00157 *
00158 * Other Supported File Formats:
00159 * \li "JP2" - JPEG-2000 JP2 File Format
00160 * \li "AVI" - Windows Audio-Video Interleaved RIFF
00161 * \li "WMV" - Windows Media Video Format
00162 *
00163 * Some Known Compressions:
00164 * \li "NONE" - No Compression.
00165 * \li "RLE" - Run Length Encoding.
00166 * \li "LZW" - Lempel, Ziff and Welsh.
00167 * \li "JPEG" - Join Photographics Experts Group.
00168 * \li "DEFLATE" - LZ77 variation (ZIP)
00169 *
00170 * \ingroup file */
00171
00172
00173 /** Library Names Convention
00174 *
00175 * Global Functions and Types - "im[Object][Action]" using
00176 * Local Functions and Types - "i[Object][Action]" using
00177 * Local Static Variables - same as local functions and type
00178 * Local Static Tables - same as local functions and types w
00179 * Variables and Members - no prefix, all lower case (width)
00180 * Defines and Enumerations - all capitals (IM_ERR_NONE)

```

```
00181  *
00182  */
00183
00184
00185 #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: im_math.h,v 1.2 2005/05/19 19:34:35 uid20021 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(y1 - box_height/2.0 - 0.5) + 1;
00052     x1 = (int)floor(xl + box_width/2.0 - 0.5);
00053     y1 = (int)floor(y1 + 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(y1 - box_height/2.0 - 0.5) + 1;
00096     x1 = (int)floor(xl + box_width/2.0 - 0.5);
00097     y1 = (int)floor(y1 + 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 }
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)(x1-0.5f);
00149     int y0 = (int)(y1-0.5f);
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)(x1-0.5f);
00161     int y0 = (int)(y1-0.5f);
00162     int x1 = x0+1;
00163     int y1 = y0+1;
00164
00165     float t = x1 - (x0+0.5f);
00166     float u = y1 - (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 fh1 = map[y0*width + x1];
00175     T flh = map[y1*width + x0];
00176     T fhh = map[y1*width + x1];
00177
00178     return (T)((fhh - flh - fh1 + fll) * u * t +
00179                (fh1 - 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)(x1-0.5f);
00192         int y0 = (int)(y1-0.5f);
00193         int x1 = x0-1;
00194         int x2 = x0+2;
00195         int y1 = y0-1;
00196         int y2 = y0+2;
00197
00198         float t = x1 - (x0+0.5f);
00199         float u = y1 - (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<=(TU)0? (TU)0: Value<=(TU)255? Value: (TU)
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: im_math_op.h,v 1.2 2005/12/05 20:44:36 scuri 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 inline imbyte pow_op(const imbyte& v1, const imbyte& v2)
00088 {
00089     return (imbyte)pow((float)v1, v2);
00090 }
00091
00092 inline imushort pow_op(const imushort& v1, const imushort& v2)
00093 {
00094     return (imushort)pow((float)v1, v2);
00095 }
00096
00097 inline int pow_op(const int& v1, const int& v2)
00098 {
00099     return (int)pow((float)v1, v2);
00100 }
00101
00102 /// Generic Power with 2 template types
00103 template <class T1, class T2>
00104 inline T1 pow_op(const T1& v1, const T2& v2)
00105 {
00106     return (T1)pow(v1, v2);
00107 }
00108
00109 /// Generic Absolute
00110 template <class T>
00111 inline T abs_op(const T& v)
00112 {
00113     if (v <= 0)
00114         return -1*v;
00115     return v;
00116 }
00117
00118 /// Generic Less
00119 template <class T>
00120 inline T less_op(const T& v)
00121 {
00122     return -1*v;
00123 }
00124
00125 /// Generic Square
00126 template <class T>
00127 inline T sqr_op(const T& v)
00128 {
00129     return v*v;
00130 }
00131
00132 inline int sqrt(const int& C)
00133 {

```

```
00134     return (int)sqrt(float(C));
00135 }
00136
00137 /// Generic Square Root
00138 template <class T>
00139 inline T sqrt_op(const T& v)
00140 {
00141     return (T)sqrt(v);
00142 }
00143
00144 inline int exp(const int& v)
00145 {
00146     return (int)exp((float)v);
00147 }
00148
00149 /// Generic Exponential
00150 template <class T>
00151 inline T exp_op(const T& v)
00152 {
00153     return (T)exp(v);
00154 }
00155
00156 inline int log(const int& v)
00157 {
00158     return (int)log((float)v);
00159 }
00160
00161 /// Generic Logarithm
00162 template <class T>
00163 inline T log_op(const T& v)
00164 {
00165     // if (v <= 0) return (T)IM_NEARINF;
00166     return (T)log(v);
00167 }
00168
00169 // Dummy sin
00170 inline imcfloat sin(const imcfloat& v)
00171 {
00172     return (v);
00173 }
00174
00175 inline int sin(const int& v)
00176 {
00177     return (int)sin((float)v);
00178 }
00179
00180 /// Generic Sine
```

```
00181 template <class T>
00182 inline T sin_op(const T& v)
00183 {
00184     return (T)sin(v);
00185 }
00186
00187 inline int cos(const int& v)
00188 {
00189     return (int)cos((float)v);
00190 }
00191
00192 // Dummy cos
00193 inline imcfloat cos(const imcfloat& v)
00194 {
00195     return (v);
00196 }
00197
00198 /// Generic Cosine
00199 template <class T>
00200 inline T cos_op(const T& v)
00201 {
00202     return (T)cos(v);
00203 }
00204
00205 /// Sets a bit in an array
00206 inline void imDataBitSet(imbyte* data, int index, int bit)
00207 {
00208     if (bit)
00209         data[index / 8] |= (0x01 << (7 - (index % 8)));
00210     else
00211         data[index / 8] &= ~(0x01 << (7 - (index % 8)));
00212 }
00213
00214 /// Gets a bit from an array
00215 inline int imDataBitGet(imbyte* data, int index)
00216 {
00217     return (data[index / 8] >> (7 - (index % 8))) & 0x01;
00218 }
00219
00220 #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: im_palette.h,v 1.4 2005/12/12 13:31:08 scuri 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  * In Lua, to create a palette you can call im.PaletteCreate.
00021  * \verbatim im.PaletteCreate([count: number]) -> pal: imPalet
00022  * Default count is 256.
00023  * IMLua and CDLua palettes are 100% compatible.
00024  * \par
00025  * See \ref im_palette.h
00026  * \ingroup util */
00027
00028
00029 /** Searches for the nearest color on the table and returns th
00030  * It looks in all palette entries and finds the minimum eucli
00031  * If the color matches the given color it returns immediately
00032  *
00033  * \verbatim im.PaletteFindNearest(pal: imPalette, color: ligh
00034  * \ingroup palette */
00035 int imPaletteFindNearest(const long *palette, int palette_coun
00036
00037 /** Searches for the color on the table and returns the color
00038  * If the tolerance is 0 search for the exact match in the pal
00039  * first color that fits in the tolerance range.
```

```

00040 *
00041 * \verbatim im.PaletteFindColor(pal: imPalette, color: lightu
00042 * \ingroup palette */
00043 int imPaletteFindColor(const long *palette, int palette_count,
00044
00045 /** Creates a palette of gray scale values.
00046 * The colors are arranged from black to white.
00047 *
00048 * \verbatim im.PaletteGray() -> pal: imPalette [in Lua 5] \en
00049 * \ingroup palette */
00050 long* imPaletteGray(void);
00051
00052 /** Creates a palette of a gradient of red colors.
00053 * The colors are arranged from black to pure red.
00054 *
00055 * \verbatim im.PaletteRed() -> pal: imPalette [in Lua 5] \end
00056 * \ingroup palette */
00057 long* imPaletteRed(void);
00058
00059 /** Creates a palette of a gradient of green colors.
00060 * The colors are arranged from black to pure green.
00061 *
00062 * \verbatim im.PaletteGreen() -> pal: imPalette [in Lua 5] \e
00063 * \ingroup palette */
00064 long* imPaletteGreen(void);
00065
00066 /** Creates a palette of a gradient of blue colors.
00067 * The colors are arranged from black to pure blue.
00068 *
00069 * \verbatim im.PaletteBlue() -> pal: imPalette [in Lua 5] \en
00070 * \ingroup palette */
00071 long* imPaletteBlue(void);
00072
00073 /** Creates a palette of a gradient of yellow colors.
00074 * The colors are arranged from black to pure yellow.
00075 *
00076 * \verbatim im.PaletteYellow() -> pal: imPalette [in Lua 5] \
00077 * \ingroup palette */
00078 long* imPaletteYellow(void);
00079
00080 /** Creates a palette of a gradient of magenta colors.
00081 * The colors are arranged from black to pure magenta.
00082 *
00083 * \verbatim im.PaletteMagenta() -> pal: imPalette [in Lua 5]
00084 * \ingroup palette */
00085 long* imPaletteMagenta(void);
00086

```

```

00087 /** Creates a palette of a gradient of cian colors.
00088 * The colors are arranged from black to pure cian.
00089 *
00090 * \verbatim im.PaletteCian() -> pal: imPalette [in Lua 5] \en
00091 * \ingroup palette */
00092 long* imPaletteCian(void);
00093
00094 /** Creates a palette of rainbow colors.
00095 * The colors are arranged in the light wave length spectrum o
00096 *
00097 * \verbatim im.PaletteRainbow() -> pal: imPalette [in Lua 5]
00098 * \ingroup palette */
00099 long* imPaletteRainbow(void);
00100
00101 /** Creates a palette of hues with maximum saturation.
00102 *
00103 * \verbatim im.PaletteHues() -> pal: imPalette [in Lua 5] \en
00104 * \ingroup palette */
00105 long* imPaletteHues(void);
00106
00107 /** Creates a palette of a gradient of blue colors.
00108 * The colors are arranged from pure blue to white.
00109 *
00110 * \verbatim im.PaletteBlueIce() -> pal: imPalette [in Lua 5]
00111 * \ingroup palette */
00112 long* imPaletteBlueIce(void);
00113
00114 /** Creates a palette of a gradient from black to white passin
00115 *
00116 * \verbatim im.PaletteHotIron() -> pal: imPalette [in Lua 5]
00117 * \ingroup palette */
00118 long* imPaletteHotIron(void);
00119
00120 /** Creates a palette of a gradient from black to white passin
00121 *
00122 * \verbatim im.PaletteBlackBody() -> pal: imPalette [in Lua 5]
00123 * \ingroup palette */
00124 long* imPaletteBlackBody(void);
00125
00126 /** Creates a palette with high contrast colors.
00127 *
00128 * \verbatim im.PaletteHighContrast() -> pal: imPalette [in Lu
00129 * \ingroup palette */
00130 long* imPaletteHighContrast(void);
00131
00132 /** Creates a palette of an uniform range of colors from black
00133 * This is a  $2^{(2.6)}$  bits per pixel palette.

```

```
00134  *
00135  * \verbatim im.PaletteUniform() -> pal: imPalette [in Lua 5]
00136  * \ingroup palette */
00137 long* imPaletteUniform(void);
00138
00139 /** Returns the index of the correspondent RGB color of an uni
00140  *
00141  * \verbatim im.PaletteUniformIndex(color: lightuserdata) -> i
00142  * \ingroup palette */
00143 int imPaletteUniformIndex(long color);
00144
00145 /** Returns the index of the correspondent RGB color of an uni
00146  * Uses an 8x8 ordered dither to lookup the index in a halftone
00147  * The spatial position used by the halftone method.
00148  *
00149  * \verbatim im.PaletteUniformIndexHalftoned(color: lightuserdata)
00150  * \ingroup palette */
00151 int imPaletteUniformIndexHalftoned(long color, int x, int y);
00152
00153
00154 #if defined(__cplusplus)
00155 }
00156 #endif
00157
00158 #endif
00159
```

include

im_process.h File Reference

Detailed Description

See Copyright Notice in [im_lib.h](#)

Id

[im_process.h](#),v 1.1 2005/04/02 22:07:00 scuri Exp

[Go to the source code of this file.](#)

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: im_process.h,v 1.1 2005/04/02 22:07:00 scuri 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  */
00030
00031
00032 #if defined(__cplusplus)
00033 }
00034 #endif
00035
00036 #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: im_process_ana.h,v 1.6 2005/12/11 23:41:25 scuri 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 two images (Root Mean Squ
00027  *
00028  * \verbatim im.CalcRMSError(image1: imImage, image2: imImage)
00029  * \ingroup stats */
00030 float imCalcRMSError(const imImage* image1, const imImage* ima
00031
00032 /** Calculates the SNR of an image and its noise (Signal Noise
00033  *
00034  * \verbatim im.CalcSNR(src_image: imImage, noise_image: imIma
00035  * \ingroup stats */
00036 float imCalcSNR(const imImage* src_image, const imImage* noise
00037
00038 /** Count the number of different colors in an image. \n
00039  * Image must be IM_BYTE, but all color spaces except IM_CMYK.
```

```

00040 *
00041 * \verbatim im.CalcCountColors(image: imImage) -> count: numb
00042 * \ingroup stats */
00043 unsigned long imCalcCountColors(const imImage* image);
00044
00045 /** Calculates the histogram of a IM_BYTE data. \n
00046 * Histogram is always 256 positions long. \n
00047 * When cumulative is different from zero it calculates the cu
00048 *
00049 * \verbatim im.CalcHistogram(image: imImage, plane: number, c
00050 * Where plane is the depth plane to calculate the histogram.
00051 * The returned table is zero indexed. image can be IM_USHORT
00052 * \ingroup stats */
00053 void imCalcHistogram(const unsigned char* data, int count, uns
00054
00055 /** Calculates the histogram of a IM_USHORT data. \n
00056 * Histogram is always 65535 positions long. \n
00057 * When cumulative is different from zero it calculates the cu
00058 * Use \ref imCalcHistogram in Lua.
00059 * \ingroup stats */
00060 void imCalcUShortHistogram(const unsigned short* data, int cou
00061
00062 /** Calculates the gray histogram of an image. \n
00063 * Image must be IM_BYTE/(IM_RGB, IM_GRAY, IM_BINARY or IM_MAP
00064 * If the image is IM_RGB then the histogram of the luma compo
00065 * Histogram is always 256 positions long. \n
00066 * When cumulative is different from zero it calculates the cu
00067 *
00068 * \verbatim im.CalcGrayHistogram(image: imImage, cumulative:
00069 * \ingroup stats */
00070 void imCalcGrayHistogram(const imImage* image, unsigned long*
00071
00072 /** Numerical Statistics Structure
00073 * \ingroup stats */
00074 typedef struct _imStats
00075 {
00076     float max;                /**< Maximum value          */
00077     float min;                /**< Minimum value          */
00078     unsigned long positive;   /**< Number of Positive Values */
00079     unsigned long negative;   /**< Number of Negative Values */
00080     unsigned long zeros;      /**< Number of Zeros         */
00081     float mean;               /**< Mean                    */
00082     float stddev;            /**< Standard Deviation      */
00083 } imStats;
00084
00085 /** Calculates the statistics about the image data. \n
00086 * There is one stats for each depth plane. For ex: stats[0]=r

```

```

00087 * Supports all data types except IM_CFLOAT. \n
00088 *
00089 * \verbatim im.CalcImageStatistics(image: imImage) -> stats:
00090 * Table contains the following fields: max, min, positive, ne
00091 * The same as the \ref imStats structure.
00092 * \ingroup stats */
00093 void imCalcImageStatistics(const imImage* image, imStats* stat
00094
00095 /** Calculates the statistics about the image histogram data.\
00096 * There is one stats for each depth plane. For ex: stats[0]=r
00097 * Only IM_BYTE images are supported.
00098 *
00099 * \verbatim im.CalcHistogramStatistics(image: imImage) -> sta
00100 * \ingroup stats */
00101 void imCalcHistogramStatistics(const imImage* image, imStats*
00102
00103 /** Calculates some extra statistics about the image histogram
00104 * There is one stats for each depth plane. \n
00105 * Only IM_BYTE images are supported. \n
00106 * mode will be -1 if more than one max is found.
00107 *
00108 * \verbatim im.CalcHistoImageStatistics(image: imImage) -> me
00109 * \ingroup stats */
00110 void imCalcHistoImageStatistics(const imImage* image, int* med
00111
00112
00113
00114 /** \defgroup analyze Image Analysis
00115 * \par
00116 * See \ref im_process_ana.h
00117 * \ingroup process */
00118
00119 /** Find white regions in binary image. \n
00120 * Result is IM_GRAY/IM_USHORT type. Regions can be 4 connecte
00121 * Returns the number of regions found. Background is marked a
00122 * Regions touching the border are considered only if touch_bo
00123 *
00124 * \verbatim im.AnalyzeFindRegions(src_image: imImage, dst_ima
00125 * \verbatim im.AnalyzeFindRegionsNew(image: imImage, connect:
00126 * \ingroup analyze */
00127 int imAnalyzeFindRegions(const imImage* src_image, imImage* ds
00128
00129 /** Measure the actual area of all regions. Holes are not incl
00130 * This is the number of pixels of each region. \n
00131 * Source image is IM_GRAY/IM_USHORT type (the result of \ref
00132 * area has size the number of regions.
00133 *

```

```

00134 * \verbatim im.AnalyzeMeasureArea(image: imImage, [region_cou
00135 * The returned table is zero indexed.
00136 * \ingroup analyze */
00137 void imAnalyzeMeasureArea(const imImage* image, int* area);
00138
00139 /** Measure the polygonal area limited by the perimeter line o
00140 * Notice that some regions may have polygonal area zero. \n
00141 * Source image is IM_GRAY/IM_USHORT type (the result of \ref
00142 * perimarea has size the number of regions.
00143 *
00144 * \verbatim im.AnalyzeMeasurePerimArea(image: imImage, [regio
00145 * The returned table is zero indexed.
00146 * \ingroup analyze */
00147 void imAnalyzeMeasurePerimArea(const imImage* image, float* pe
00148
00149 /** Calculate the centroid position of all regions. Holes are
00150 * Source image is IM_GRAY/IM_USHORT type (the result of \ref
00151 * area, cx and cy have size the number of regions. If area is
00152 *
00153 * \verbatim im.AnalyzeMeasureCentroid(image: imImage, [area:
00154 * The returned tables are zero indexed.
00155 * \ingroup analyze */
00156 void imAnalyzeMeasureCentroid(const imImage* image, const int*
00157
00158 /** Calculate the principal major axis slope of all regions. \
00159 * Source image is IM_GRAY/IM_USHORT type (the result of \ref
00160 * data has size the number of regions. If area or centroid ar
00161 * Principal (major and minor) axes are defined to be those ax
00162 * centroid, about which the moment of inertia of the region i
00163 *
00164 * \verbatim im.AnalyzeMeasurePrincipalAxis(image: imImage, [a
00165 * -> major_slope: table of numbers
00166 * The returned tables are zero indexed.
00167 * \ingroup analyze */
00168 void imAnalyzeMeasurePrincipalAxis(const imImage* image, const
00169 * const int region_count, flo
00170 * flo
00171
00172 /** Measure the number and area of holes of all regions. \n
00173 * Source image is IM_USHORT type (the result of \ref imAnalyze
00174 * area and perim has size the number of regions, if some is N
00175 *
00176 * \verbatim im.AnalyzeMeasureHoles(image: imImage, connect: n
00177 * The returned tables are zero indexed.
00178 * \ingroup analyze */
00179 void imAnalyzeMeasureHoles(const imImage* image, int connect,
00180

```

```

00181 /** Measure the total perimeter of all regions (external and i
00182  * Source image is IM_GRAY/IM_USHORT type (the result of imAna
00183  * It uses a half-pixel inter distance for 8 neighbors in a p
00184  * This function can also be used to measure line lenght. \n
00185  * perim has size the number of regions.
00186  *
00187  * \verbatim im.AnalyzeMeasurePerimeter(image: imImage) -> per
00188  * \ingroup analyze */
00189 void imAnalyzeMeasurePerimeter(const imImage* image, float* pe
00190
00191 /** Isolates the perimeter line of gray integer images. Backgr
00192  * It just checks if at least one of the 4 connected neighbor
00193  *
00194  * \verbatim im.ProcessPerimeterLine(src_image: imImage, dst_i
00195  * \verbatim im.ProcessPerimeterLineNew(image: imImage) -> new
00196  * \ingroup analyze */
00197 void imProcessPerimeterLine(const imImage* src_image, imImage*
00198
00199 /** Eliminates regions that have size outside the given interv
00200  * Source and destiny are a binary images. Regions can be 4 co
00201  * Can be done in-place. end_size can be zero to ignore big ob
00202  *
00203  * \verbatim im.ProcessPrune(src_image: imImage, dst_image: im
00204  * \verbatim im.ProcessPruneNew(image: imImage, connect: numbe
00205  * \ingroup analyze */
00206 void imProcessPrune(const imImage* src_image, imImage* dst_ima
00207
00208 /** Fill holes inside white regions. \n
00209  * Source and destiny are a binary images. Regions can be 4 co
00210  * Can be done in-place.
00211  *
00212  * \verbatim im.ProcessFillHoles(src_image: imImage, dst_image
00213  * \verbatim im.ProcessFillHolesNew(image: imImage, connect: n
00214  * \ingroup analyze */
00215 void imProcessFillHoles(const imImage* src_image, imImage* dst
00216
00217
00218 #if defined(__cplusplus)
00219 }
00220 #endif
00221
00222 #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: im_process_glo.h,v 1.4 2005/12/12 20:29:00 scuri 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 Other Domain Transform Operations
00020  * \par
00021  * Hough, Distance.
00022  *
00023  * See \ref im_process_glo.h
00024  * \ingroup process */
00025
00026 /** Hough Lines Transform. \n
00027  * It will detect white lines in a black background. So the so
00028  * with the white lines of interest enhanced. The better the t
00029  * the line detection. \n
00030  * The destiny image must have IM_GRAY, IM_INT, width=180, hei
00031  * The houfh transform defines "cos(theta) * X + sin(theta)
00032  * theta = "0 .. 179", rho = "-height/2 .. height/2" .\n
00033  * Returns zero if the counter aborted. \n
00034  * Inspired from ideas in XITE, Copyright 1991, Blab, UiO \n
00035  * http://www.ifi.uio.no/~blab/Software/Xite/
00036  *
00037  * \verbatim im.ProcessHoughLines(src_image: imImage, dst_imag
00038  * \verbatim im.ProcessHoughLinesNew(image: imImage) -> counte
00039  * \ingroup transform */
```

```

00040 int imProcessHoughLines(const imImage* src_image, imImage* dst
00041
00042 /** Draw detected hough lines. \n
00043 * The source image must be IM_GRAY and IM_BYTE. The destiny i
00044 * it can be the source image for in place processing. \n
00045 * The hough points image is a hough transform image that was
00046 * usually using a Local Max threshold operation. Again the be
00047 * The destiny image will be set to IM_MAP, and the detected l
00048 * Returns the number of detected lines.
00049 *
00050 * \verbatim im.ProcessHoughLinesDraw(src_image: imImage, houg
00051 * \verbatim im.ProcessHoughLinesDrawNew(image: imImage, hough
00052 * \ingroup transform */
00053 int imProcessHoughLinesDraw(const imImage* src_image, const im
00054
00055 /** Calculates the Cross Correlation in the frequency domain.
00056 *  $CrossCorr(a,b) = IFFT(Conj(FFT(a))*FFT(b))$  \n
00057 * Images must be of the same size and only destiny image must
00058 *
00059 * \verbatim im.ProcessCrossCorrelation(src_image1: imImage, s
00060 * \verbatim im.ProcessCrossCorrelationNew(image1: imImage, im
00061 * \ingroup transform */
00062 void imProcessCrossCorrelation(const imImage* src_image1, cons
00063
00064 /** Calculates the Auto Correlation in the frequency domain. \
00065 * Uses the cross correlation.
00066 * Images must be of the same size and only destiny image must
00067 *
00068 * \verbatim im.ProcessAutoCorrelation(src_image: imImage, dst
00069 * \verbatim im.ProcessAutoCorrelationNew(image: imImage) -> n
00070 * \ingroup transform */
00071 void imProcessAutoCorrelation(const imImage* src_image, imImag
00072
00073 /** Calculates the Distance Transform of a binary image
00074 * using an aproximation of the euclidian distance.\n
00075 * Each white pixel in the binary image is
00076 * assigned a value equal to its distance from the nearest
00077 * black pixel. \n
00078 * Uses a two-pass algorithm incrementally calculating the dis
00079 * Source image must be IM_BINARY, destiny must be IM_FLOAT.
00080 *
00081 * \verbatim im.ProcessDistanceTransform(src_image: imImage, d
00082 * \verbatim im.ProcessDistanceTransformNew(image: imImage) ->
00083 * \ingroup transform */
00084 void imProcessDistanceTransform(const imImage* src_image, imIm
00085
00086 /** Marks all the regional maximum of the distance transform.

```

```

00087 * source is IMGRAY/IM_FLOAT destiny in IM_BINARY. \n
00088 * We consider maximum all connected pixel values that have sm
00089 *
00090 * \verbatim im.ProcessRegionalMaximum(src_image: imImage, dst
00091 * \verbatim im.ProcessRegionalMaximumNew(image: imImage) -> n
00092 * \ingroup transform */
00093 void imProcessRegionalMaximum(const imImage* src_image, imImag
00094
00095
00096
00097 /** \defgroup fourier Fourier Transform Operations
00098 * \par
00099 * All Fourier transforms use FFTW library version 2.1.5. \n
00100 * Although there are newer versions, we build binaries only t
00101 * because it is small and as fast as newer versions.
00102 * Source code to use FFTW version 3 is available.
00103 * \par
00104 * FFTW Copyright Matteo Frigo, Steven G. Johnson and the MIT.
00105 * http://www.fftw.org
00106 * See "fftw.h"
00107 * \par
00108 * Must link with "im_fftw" library. \n
00109 * \par
00110 * The FFTW lib has a GPL license. The license of the "im_fftw
00111 * So you cannot use it for commercial applications without co
00112 * \par
00113 * See \ref im_process_glo.h
00114 * \ingroup process */
00115
00116 /** Forward FFT. \n
00117 * The result has its lowest frequency at the center of the im
00118 * This is an unnormalized fft. \n
00119 * Images must be of the same size. Destiny image must be of t
00120 *
00121 * \verbatim im.ProcessFFT(src_image: imImage, dst_image: imIm
00122 * \verbatim im.ProcessFFTNew(image: imImage) -> new_image: im
00123 * \ingroup fourier */
00124 void imProcessFFT(const imImage* src_image, imImage* dst_image
00125
00126 /** Inverse FFT. \n
00127 * The image has its lowest frequency restored to the origin b
00128 * The result is normalized by (width*height). \n
00129 * Images must be of the same size and both must be of type co
00130 *
00131 * \verbatim im.ProcessIFFT(src_image: imImage, dst_image: imI
00132 * \verbatim im.ProcessIFFTNew(image: imImage) -> new_image: i
00133 * \ingroup fourier */

```

```

00134 void imProcessIFFT(const imImage* src_image, imImage* dst_imag
00135
00136 /** Raw in-place FFT (forward or inverse). \n
00137 * The lowest frequency can be centered after forward, or
00138 * can be restored to the origin before inverse. \n
00139 * The result can be normalized after the transform by sqrt(w*
00140 * or left unnormalized [0]. \n
00141 * Images must be of the same size and both must be of type co
00142 *
00143 * \verbatim im.ProcessFFTraw(image: imImage, inverse: number,
00144 * \ingroup fourier */
00145 void imProcessFFTraw(imImage* image, int inverse, int center,
00146
00147 /** Auxiliary function for the raw FFT. \n
00148 * This is the function used internally to change the lowest f
00149 * If the image size has even dimensions the flag "center2orig
00150 * you must specify if its from center to origin (usually used
00151 * from origin to center (usually used after forward). \n
00152 * Notice that this function is used for images in the the fre
00153 * Image type must be complex.
00154 *
00155 * \verbatim im.ProcessSwapQuadrants(image: imImage, center2or
00156 * \ingroup fourier */
00157 void imProcessSwapQuadrants(imImage* image, int center2origin)
00158
00159
00160 #if defined(__cplusplus)
00161 }
00162 #endif
00163
00164 #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: im_process_loc.h,v 1.6 2005/12/13 18:35:02 scuri 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 reduce the image size using the given decimation order
00027  * Supported decimation 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  *
00033  * \verbatim im.ProcessReduce(src_image: imImage, dst_image: i
00034  * \verbatim im.ProcessReduceNew(image: imImage, order: number
00035  * \ingroup resize */
00036 int imProcessReduce(const imImage* src_image, imImage* dst_ima
00037
00038 /** Change the image size using the given interpolation order.
00039  * Supported interpolation orders:
```

```

00040 * \li 0 - zero order (near neighborhood)
00041 * \li 1 - first order (bilinear interpolation)
00042 * \li 3 - third order (bicubic interpolation)
00043 * Images must be of the same type. \n
00044 * Returns zero if the counter aborted.
00045 *
00046 * \verbatim im.ProcessResize(src_image: imImage, dst_image: i
00047 * \verbatim im.ProcessResizeNew(image: imImage, order: number
00048 * \ingroup resize */
00049 int imProcessResize(const imImage* src_image, imImage* dst_ima
00050
00051 /** Reduce the image area by 4 (w/2,h/2). \n
00052 * Images must be of the same type. Destiny image size must be
00053 *
00054 * \verbatim im.ProcessReduceBy4(src_image: imImage, dst_image
00055 * \verbatim im.ProcessReduceBy4New(image: imImage) -> new_ima
00056 * \ingroup resize */
00057 void imProcessReduceBy4(const imImage* src_image, imImage* dst
00058
00059 /** Extract a rectangular region from an image. \n
00060 * Images must be of the same type. Destiny image size must be
00061 * ymin and xmin must be >0 and <size.
00062 *
00063 * \verbatim im.ProcessCrop(src_image: imImage, dst_image: imI
00064 * \verbatim im.ProcessCropNew(image: imImage, xmin: number, x
00065 * \ingroup resize */
00066 void imProcessCrop(const imImage* src_image, imImage* dst_imag
00067
00068 /** Insert a rectangular region in an image. \n
00069 * Images must be of the same type. Region image size can be 1
00070 * ymin and xmin must be >0 and <size. \n
00071 * Source and destiny must be of the same size. Can be done in
00072 *
00073 * \verbatim im.ProcessInsert(src_image: imImage, region_image
00074 * \verbatim im.ProcessInsertNew(image: imImage, region_image:
00075 * \ingroup resize */
00076 void imProcessInsert(const imImage* src_image, const imImage*
00077
00078 /** Increase the image size by adding pixels with zero value.
00079 * Images must be of the same type. Destiny image size must be
00080 *
00081 * \verbatim im.ProcessAddMargins(src_image: imImage, dst_imag
00082 * \verbatim im.ProcessAddMarginsNew(image: imImage, xmin: num
00083 * \ingroup resize */
00084 void imProcessAddMargins(const imImage* src_image, imImage* ds
00085
00086

```

```

00087
00088 /** \defgroup geom Geometric Operations
00089 * \par
00090 * Operations to change the shape of the image.
00091 * \par
00092 * See \ref im_process_loc.h
00093 * \ingroup process */
00094
00095 /** Calculates the size of the new image after rotation.
00096 *
00097 * \verbatim im.ProcessCalcRotateSize(width: number, height: n
00098 * \ingroup geom */
00099 void imProcessCalcRotateSize(int width, int height, int *new_w
00100
00101 /** Rotates the image using the given interpolation order (see
00102 * Images must be of the same type. The destiny size can be ca
00103 * Returns zero if the counter aborted.
00104 *
00105 * \verbatim im.ProcessRotate(src_image: imImage, dst_image: i
00106 * \verbatim im.ProcessRotateNew(image: imImage, cos0: number,
00107 * \ingroup geom */
00108 int imProcessRotate(const imImage* src_image, imImage* dst_ima
00109
00110 /** Rotate the image in 90 degrees counterclockwise or clockwi
00111 * Images must be of the same type. Destiny width and height m
00112 * Direction can be clockwise (1) or counter clockwise (-1).
00113 *
00114 * \verbatim im.ProcessRotate90(src_image: imImage, dst_image:
00115 * \verbatim im.ProcessRotate90New(image: imImage, dir_clockwi
00116 * \ingroup geom */
00117 void imProcessRotate90(const imImage* src_image, imImage* dst_
00118
00119 /** Rotate the image in 180 degrees. Swap columns and swap lin
00120 * Images must be of the same type and size.
00121 *
00122 * \verbatim im.ProcessRotate180(src_image: imImage, dst_image
00123 * \verbatim im.ProcessRotate180New(image: imImage) -> new_ima
00124 * \ingroup geom */
00125 void imProcessRotate180(const imImage* src_image, imImage* dst
00126
00127 /** Mirrors the image in a horizontal flip. Swap columns. \n
00128 * Images must be of the same type and size.
00129 *
00130 * \verbatim im.ProcessMirror(src_image: imImage, dst_image: i
00131 * \verbatim im.ProcessMirrorNew(image: imImage) -> new_image:
00132 * \ingroup geom */
00133 void imProcessMirror(const imImage* src_image, imImage* dst_ir

```

```

00134
00135 /** Apply a vertical flip. Swap lines. \n
00136 * Images must be of the same type and size.
00137 *
00138 * \verbatim im.ProcessFlip(src_image: imImage, dst_image: imI
00139 * \verbatim im.ProcessFlipNew(image: imImage) -> new_image: i
00140 * \ingroup geom */
00141 void imProcessFlip(const imImage* src_image, imImage* dst_imag
00142
00143 /** Apply a radial distortion using the given interpolation or
00144 * Images must be of the same type and size. Returns zero if t
00145 *
00146 * \verbatim im.ProcessRadial(src_image: imImage, dst_image: i
00147 * \verbatim im.ProcessRadialNew(image: imImage, k1: number, o
00148 * \ingroup geom */
00149 int imProcessRadial(const imImage* src_image, imImage* dst_ima
00150
00151
00152
00153 /** \defgroup morphgray Morphology Operations for Gray Images
00154 * \par
00155 * See \ref im_process_loc.h
00156 * \ingroup process */
00157
00158 /** Base gray morphology convolution. \n
00159 * Supports all data types except IM_CFLOAT. Can be applied on
00160 * Kernel is always IM_INT. Use kernel size odd for better res
00161 * You can use the maximum value or else the minimum value. \n
00162 * No border extensions are used.
00163 * All the gray morphology operations use this function. \n
00164 * If the kernel image attribute "Description" exists it is us
00165 *
00166 * \verbatim im.ProcessGrayMorphConvolve(src_image: imImage, d
00167 * \verbatim im.ProcessGrayMorphConvolveNew(image: imImage, ke
00168 * \ingroup morphgray */
00169 int imProcessGrayMorphConvolve(const imImage* src_image, imIma
00170
00171 /** Gray morphology convolution with a kernel full of "0"s and
00172 *
00173 * \verbatim im.ProcessGrayMorphErode(src_image: imImage, dst_
00174 * \verbatim im.ProcessGrayMorphErodeNew(image: imImage, kerne
00175 * \ingroup morphgray */
00176 int imProcessGrayMorphErode(const imImage* src_image, imImage*
00177
00178 /** Gray morphology convolution with a kernel full of "0"s and
00179 *
00180 * \verbatim im.ProcessGrayMorphDilate(src_image: imImage, dst

```

```

00181 * \verbatim im.ProcessGrayMorphDilateNew(image: imImage, kern
00182 * \ingroup morphgray */
00183 int imProcessGrayMorphDilate(const imImage* src_image, imImage
00184
00185 /** Erode+Dilate.
00186 *
00187 * \verbatim im.ProcessGrayMorphOpen(src_image: imImage, dst_i
00188 * \verbatim im.ProcessGrayMorphOpenNew(image: imImage, kernel
00189 * \ingroup morphgray */
00190 int imProcessGrayMorphOpen(const imImage* src_image, imImage*
00191
00192 /** Dilate+Erode.
00193 *
00194 * \verbatim im.ProcessGrayMorphClose(src_image: imImage, dst_
00195 * \verbatim im.ProcessGrayMorphCloseNew(image: imImage, kerne
00196 * \ingroup morphgray */
00197 int imProcessGrayMorphClose(const imImage* src_image, imImage*
00198
00199 /** Open+Difference.
00200 *
00201 * \verbatim im.ProcessGrayMorphTopHat(src_image: imImage, dst
00202 * \verbatim im.ProcessGrayMorphTopHatNew(image: imImage, kern
00203 * \ingroup morphgray */
00204 int imProcessGrayMorphTopHat(const imImage* src_image, imImage
00205
00206 /** Close+Difference.
00207 *
00208 * \verbatim im.ProcessGrayMorphWell(src_image: imImage, dst_i
00209 * \verbatim im.ProcessGrayMorphWellNew(image: imImage, kernel
00210 * \ingroup morphgray */
00211 int imProcessGrayMorphWell(const imImage* src_image, imImage*
00212
00213 /** Difference(Erode, Dilate).
00214 *
00215 * \verbatim im.ProcessGrayMorphGradient(src_image: imImage, d
00216 * \verbatim im.ProcessGrayMorphGradientNew(image: imImage, ke
00217 * \ingroup morphgray */
00218 int imProcessGrayMorphGradient(const imImage* src_image, imIma
00219
00220
00221
00222 /** \defgroup morphbin Morphology Operations for Binary Images
00223 * \par
00224 * See \ref im_process_loc.h
00225 * \ingroup process */
00226
00227 /** Base binary morphology convolution. \n

```

```

00228 * Images are all IM_BINARY. Kernel is IM_INT. Use kernel size
00229 * Hit white means hit=1 and miss=0, or else hit=0 and miss=1.
00230 * Use -1 for don't care positions in kernel. \n
00231 * The operation can be repeated by a number of iterations.
00232 * The border is zero extended. \n
00233 * Almost all the binary morphology operations use this functi
00234 * If the kernel image attribute "Description" exists it is us
00235 *
00236 * \verbatim im.ProcessBinMorphConvolve(src_image: imImage, ds
00237 * \verbatim im.ProcessBinMorphConvolveNew(image: imImage, ker
00238 * \ingroup morphbin */
00239 int imProcessBinMorphConvolve(const imImage* src_image, imImag
00240
00241 /** Binary morphology convolution with a kernel full of "1"s a
00242 *
00243 * \verbatim im.ProcessBinMorphErode(src_image: imImage, dst_i
00244 * \verbatim im.ProcessBinMorphErodeNew(image: imImage, kernel
00245 * \ingroup morphbin */
00246 int imProcessBinMorphErode(const imImage* src_image, imImage*
00247
00248 /** Binary morphology convolution with a kernel full of "0"s a
00249 *
00250 * \verbatim im.ProcessBinMorphDilate(src_image: imImage, dst_
00251 * \verbatim im.ProcessBinMorphDilateNew(image: imImage, kerne
00252 * \ingroup morphbin */
00253 int imProcessBinMorphDilate(const imImage* src_image, imImage*
00254
00255 /** Erode+Dilate.
00256 * When iteration is more than one it means Erode+Erode+Erode+
00257 *
00258 * \verbatim im.ProcessBinMorphOpen(src_image: imImage, dst_im
00259 * \verbatim im.ProcessBinMorphOpenNew(image: imImage, kernel_
00260 * \ingroup morphbin */
00261 int imProcessBinMorphOpen(const imImage* src_image, imImage* d
00262
00263 /** Dilate+Erode.
00264 *
00265 * \verbatim im.ProcessBinMorphClose(src_image: imImage, dst_i
00266 * \verbatim im.ProcessBinMorphCloseNew(image: imImage, kernel
00267 * \ingroup morphbin */
00268 int imProcessBinMorphClose(const imImage* src_image, imImage*
00269
00270 /** Erode+Difference. \n
00271 * The difference from the source image is applied only once.
00272 *
00273 * \verbatim im.ProcessBinMorphOutline(src_image: imImage, dst
00274 * \verbatim im.ProcessBinMorphOutlineNew(image: imImage, kern

```

```

00275 * \ingroup morphbin */
00276 int imProcessBinMorphOutline(const imImage* src_image, imImage
00277
00278 /** This is the supplied binary image using Rosenfeld's parallel
00279 * Reference: \n
00280 * "Efficient Binary Image Thinning using Neighborhood Maps" \
00281 * by Joseph M. Cychosz, 3ksnn64@ecn.purdue.edu \
00282 * in "Graphics Gems IV", Academic Press, 1994
00283 *
00284 * \verbatim im.ProcessBinMorphThin(src_image: imImage, dst_im
00285 * \verbatim im.ProcessBinMorphThinNew(image: imImage) -> new_
00286 * \ingroup morphbin */
00287 void imProcessBinMorphThin(const imImage* src_image, imImage*
00288
00289
00290
00291 /** \defgroup rank Rank Convolution Operations
00292 * \par
00293 * All the rank convolution use the same base function. Near t
00294 * includes only the real image pixels in the rank. No border
00295 * \par
00296 * See \ref im_process_loc.h
00297 * \ingroup process */
00298
00299 /** Rank convolution using the median value. \n
00300 * Returns zero if the counter aborted. \n
00301 * Supports all data types except IM_CFLOAT. Can be applied on
00302 *
00303 * \verbatim im.ProcessMedianConvolve(src_image: imImage, dst_
00304 * \verbatim im.ProcessMedianConvolveNew(image: imImage, kerne
00305 * \ingroup rank */
00306 int imProcessMedianConvolve(const imImage* src_image, imImage*
00307
00308 /** Rank convolution using (maximum-minimum) value. \n
00309 * Returns zero if the counter aborted. \n
00310 * Supports all data types except IM_CFLOAT. Can be applied on
00311 *
00312 * \verbatim im.ProcessRangeConvolve(src_image: imImage, dst_i
00313 * \verbatim im.ProcessRangeConvolveNew(image: imImage, kernel
00314 * \ingroup rank */
00315 int imProcessRangeConvolve(const imImage* src_image, imImage*
00316
00317 /** Rank convolution using the closest maximum or minimum valu
00318 * Returns zero if the counter aborted. \n
00319 * Supports all data types except IM_CFLOAT. Can be applied on
00320 *
00321 * \verbatim im.ProcessRankClosestConvolve(src_image: imImage,

```

```

00322 * \verbatim im.ProcessRankClosestConvolveNew(image: imImage,
00323 * \ingroup rank */
00324 int imProcessRankClosestConvolve(const imImage* src_image, imI
00325
00326 /** Rank convolution using the maximum value. \n
00327 * Returns zero if the counter aborted. \n
00328 * Supports all data types except IM_CFLOAT. Can be applied on
00329 *
00330 * \verbatim im.ProcessRankMaxConvolve(src_image: imImage, dst
00331 * \verbatim im.ProcessRankMaxConvolveNew(image: imImage, kern
00332 * \ingroup rank */
00333 int imProcessRankMaxConvolve(const imImage* src_image, imImage
00334
00335 /** Rank convolution using the minimum value. \n
00336 * Returns zero if the counter aborted. \n
00337 * Supports all data types except IM_CFLOAT. Can be applied on
00338 *
00339 * \verbatim im.ProcessRankMinConvolve(src_image: imImage, dst
00340 * \verbatim im.ProcessRankMinConvolveNew(image: imImage, kern
00341 * \ingroup rank */
00342 int imProcessRankMinConvolve(const imImage* src_image, imImage
00343
00344 /** Threshold using a rank convolution with a range contrast f
00345 * Supports all integer IM_GRAY images as source, and IM_BINAR
00346 * Local variable threshold by the method of Bernsen. \n
00347 * Extracted from XITE, Copyright 1991, Blab, UiO \n
00348 * http://www.ifi.uio.no/~blab/Software/Xite/
00349 \verbatim
00350 Reference:
00351 Bernsen, J: "Dynamic thresholding of grey-level images"
00352 Proc. of the 8th ICPR, Paris, Oct 1986, 1251-1255.
00353 Author: Oivind Due Trier
00354 \endverbatim
00355 * Returns zero if the counter aborted.
00356 *
00357 * \verbatim im.ProcessRangeContrastThreshold(src_image: imIma
00358 * \verbatim im.ProcessRangeContrastThresholdNew(image: imImag
00359 * \ingroup threshold */
00360 int imProcessRangeContrastThreshold(const imImage* src_image,
00361
00362 /** Threshold using a rank convolution with a local max functi
00363 * Returns zero if the counter aborted. \n
00364 * Supports all integer IM_GRAY images as source, and IM_BINAR
00365 *
00366 * \verbatim im.ProcessLocalMaxThreshold(src_image: imImage, d
00367 * \verbatim im.ProcessLocalMaxThresholdNew(image: imImage, ke
00368 * \ingroup threshold */

```

```

00369 int imProcessLocalMaxThreshold(const imImage* src_image, imIma
00370
00371
00372
00373 /** \defgroup convolve Convolution Operations
00374 * \par
00375 * See \ref im_process_loc.h
00376 * \ingroup process */
00377
00378 /** Base Convolution with a kernel. \n
00379 * Kernel can be IM_INT or IM_FLOAT, but always IM_GRAY. Use k
00380 * Supports all data types. The border is mirrored. \n
00381 * Returns zero if the counter aborted. Most of the convolutio
00382 * If the kernel image attribute "Description" exists it is us
00383 *
00384 * \verbatim im.ProcessConvolve(src_image: imImage, dst_image:
00385 * \verbatim im.ProcessConvolveNew(image: imImage, kernel: imI
00386 * \ingroup convolve */
00387 int imProcessConvolve(const imImage* src_image, imImage* dst_i
00388
00389 /** Repeats the convolution a number of times. \n
00390 * Returns zero if the counter aborted.\n
00391 * If the kernel image attribute "Description" exists it is us
00392 *
00393 * \verbatim im.ProcessConvolveRep(src_image: imImage, dst_ima
00394 * \verbatim im.ProcessConvolveRepNew(image: imImage, kernel:
00395 * \ingroup convolve */
00396 int imProcessConvolveRep(const imImage* src_image, imImage* ds
00397
00398 /** Convolve with a kernel rotating it 8 times and getting the
00399 * Kernel must be square. \n
00400 * The rotation is implemented only for kernel sizes 3x3, 5x5
00401 * Supports all data types except IM_CFLOAT.
00402 * Returns zero if the counter aborted.\n
00403 * If the kernel image attribute "Description" exists it is us
00404 *
00405 * \verbatim im.ProcessCompassConvolve(src_image: imImage, dst
00406 * \verbatim im.ProcessCompassConvolveNew(image: imImage, kern
00407 * \ingroup convolve */
00408 int imProcessCompassConvolve(const imImage* src_image, imImage
00409
00410 /** Utility function to rotate a kernel one time.
00411 *
00412 * \verbatim im.ProcessRotateKernel(kernel: imImage) [in Lua 5
00413 * \ingroup convolve */
00414 void imProcessRotateKernel(imImage* kernel);
00415

```

```

00416 /** Difference(Gaussian1, Gaussian2). \n
00417 * Supports all data types,
00418 * but if source is IM_BYTE or IM_USHORT destiny image must be
00419 *
00420 * \verbatim im.ProcessDiffOfGaussianConvolve(src_image: imIma
00421 * \verbatim im.ProcessDiffOfGaussianConvolveNew(image: imImag
00422 * \ingroup convolve */
00423 int imProcessDiffOfGaussianConvolve(const imImage* src_image,
00424
00425 /** Difference(Gaussian1, Gaussian2) using gaussian repetition
00426 * Supports all data types,
00427 * but if source is IM_BYTE or IM_USHORT destiny image must be
00428 *
00429 * \verbatim im.ProcessDiffOfGaussianConvolveRep(src_image: im
00430 * \verbatim im.ProcessDiffOfGaussianConvolveRepNew(image: imI
00431 * \ingroup convolve */
00432 int imProcessDiffOfGaussianConvolveRep(const imImage* src_imag
00433
00434 /** Convolution with a laplacian of a gaussian kernel. \n
00435 * Supports all data types,
00436 * but if source is IM_BYTE or IM_USHORT destiny image must be
00437 *
00438 * \verbatim im.ProcessLapOfGaussianConvolve(src_image: imImag
00439 * \verbatim im.ProcessLapOfGaussianConvolveNew(image: imImage
00440 * \ingroup convolve */
00441 int imProcessLapOfGaussianConvolve(const imImage* src_image, i
00442
00443 /** Convolution with a kernel full of "1"s inside a circle. \n
00444 * Supports all data types.
00445 *
00446 * \verbatim im.ProcessMeanConvolve(src_image: imImage, dst_im
00447 * \verbatim im.ProcessMeanConvolveNew(image: imImage, kernel_
00448 * \ingroup convolve */
00449 int imProcessMeanConvolve(const imImage* src_image, imImage* d
00450
00451 /** Convolution with a gaussian kernel. The gaussian in obtain
00452 * Supports all data types.
00453 *
00454 * \verbatim im.ProcessGaussianConvolveRep(src_image: imImage,
00455 * \verbatim im.ProcessGaussianConvolveRepNew(image: imImage,
00456 * \ingroup convolve */
00457 int imProcessGaussianConvolveRep(const imImage* src_image, imI
00458
00459 /** Convolution with a float gaussian kernel. \n
00460 * Supports all data types.
00461 *
00462 * \verbatim im.ProcessGaussianConvolve(src_image: imImage, ds

```

```

00463 * \verbatim im.ProcessGaussianConvolveNew(image: imImage, std
00464 * \ingroup convolve */
00465 int imProcessGaussianConvolve(const imImage* src_image, imImage*
00466
00467 /** Magnitude of the sobel convolution. \n
00468 * Supports all data types except IM_CFLOAT.
00469 *
00470 * \verbatim im.ProcessSobelConvolve(src_image: imImage, dst_i
00471 * \verbatim im.ProcessSobelConvolveNew(image: imImage) -> cou
00472 * \ingroup convolve */
00473 int imProcessSobelConvolve(const imImage* src_image, imImage*
00474
00475 /** Finds the zero crossings of IM_INT and IM_FLOAT images. Cr
00476 * indicating the intensity of the edge. It is usually used af
00477 * Extracted from XITE, Copyright 1991, Blab, UiO \n
00478 * http://www.ifi.uio.no/~blab/Software/Xite/
00479 *
00480 * \verbatim im.ProcessZeroCrossing(src_image: imImage, dst_im
00481 * \verbatim im.ProcessZeroCrossingNew(image: imImage) -> new_
00482 * \ingroup convolve */
00483 void imProcessZeroCrossing(const imImage* src_image, imImage*
00484
00485 /** First part of the Canny edge detector. Includes the gaussi
00486 * After using this you could apply a Hysteresis Threshold, se
00487 * Image must be IM_BYTE/IM_GRAY. \n
00488 * Implementation from the book:
00489 \verbatim
00490     J. R. Parker
00491     "Algoritms for Image Processing and Computer Vision"
00492     WILEY
00493 \endverbatim
00494 *
00495 * \verbatim im.ProcessCanny(src_image: imImage, dst_image: im
00496 * \verbatim im.ProcessCannyNew(image: imImage, stddev: number
00497 * \ingroup convolve */
00498 void imProcessCanny(const imImage* src_image, imImage* dst_ima
00499
00500 /** Calculates the number of 3x3 gaussian repetitions given th
00501 *
00502 * \verbatim im.GaussianStdDev2Repetitions(stddev: number) ->
00503 * \ingroup convolve */
00504 int imGaussianStdDev2Repetitions(float stddev);
00505
00506 /** Calculates the kernel size given the standard deviation.
00507 *
00508 * \verbatim im.GaussianStdDev2KernelSize(stddev: number) -> k
00509 * \ingroup convolve */

```

```
00510 int imGaussianStdDev2KernelSize(float stddev);
00511
00512
00513 #if defined(__cplusplus)
00514 }
00515 #endif
00516
00517 #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: im_process_pon.h,v 1.6 2005/12/12 20:29:00 scuri 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_EQL,      /**< 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_CPXNORM /**< 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 *
00048 * \verbatim im.ProcessUnArithmeticOp(src_image: imImage, dst_
00049 * \verbatim im.ProcessUnArithmeticOpNew(image: imImage, op: n
00050 * \ingroup arithm */
00051 void imProcessUnArithmeticOp(const imImage* src_image, imImage
00052
00053 /** Binary Arithmetic Operations.
00054 * Inverse and log may lead to math exceptions.
00055 * \ingroup arithm */
00056 enum imBinaryOp {
00057 IM_BIN_ADD,      /**< add          = a+b          */
00058 IM_BIN_SUB,      /**< subtract       = a-b          */
00059 IM_BIN_MUL,      /**< multiply        = a*b          */
00060 IM_BIN_DIV,      /**< divide         = a/b          (#) */
00061 IM_BIN_DIFF,     /**< difference     = |a-b|         */
00062 IM_BIN_POW,      /**< power          = a^b          */
00063 IM_BIN_MIN,      /**< minimum        = (a < b)? a: b */
00064 IM_BIN_MAX,      /**< maximum        = (a > b)? a: b */
00065 };
00066
00067 /** Apply a binary arithmetic operation. \n
00068 * Can be done in place, images must match size. \n
00069 * Source images must match type, destiny image can be several
00070 * \li byte -> byte, ushort, int, float
00071 * \li ushort -> ushort, int, float
00072 * \li int -> int, float
00073 * \li float -> float
00074 * \li complex -> complex
00075 * One exception is that you can combine complex with float re
00076 *
00077 * \verbatim im.ProcessArithmeticOp(src_image1: imImage, src_i
00078 * \verbatim im.ProcessArithmeticOpNew(image1: imImage, image2
00079 * The New function will create a new image of the same type o
00080 * \ingroup arithm */
00081 void imProcessArithmeticOp(const imImage* src_image1, const im
00082
00083 /** Apply a binary arithmetic operation with a constant value.
00084 * Can be done in place, images must match size. \n
00085 * Destiny image can be several types depending on source: \n
00086 * \li byte -> byte, ushort, int, float

```

```

00087 * \li ushort -> byte, ushort, int, float
00088 * \li int -> byte, ushort, int, float
00089 * \li float -> float
00090 * \li complex -> complex
00091 * The constant value is type casted to an appropriate type bef
00092 *
00093 * \verbatim im.ProcessArithmeticConstOp(src_image: imImage, s
00094 * \verbatim im.ProcessArithmeticConstOpNew(image: imImage, sr
00095 * \ingroup arithm */
00096 void imProcessArithmeticConstOp(const imImage* src_image, floa
00097
00098 /** Blend two images using an alpha value = [a * alpha + b * (
00099 * Can be done in place, images must match size and type.
00100 *
00101 * \verbatim im.ProcessBlend(src_image1: imImage, src_image2:
00102 * \verbatim im.ProcessBlendNew(image1: imImage, image2: imIma
00103 * \ingroup arithm */
00104 void imProcessBlend(const imImage* src_image1, imImage* src_ir
00105
00106 /** Split a complex image into two images with real and imagin
00107 * or magnitude and phase parts (polar). \n
00108 * Source image must be IM_CFLOAT, destiny images must be IM_F
00109 *
00110 * \verbatim im.ProcessSplitComplex(src_image: imImage, dst_ir
00111 * \verbatim im.ProcessSplitComplexNew(image: imImage, do_pola
00112 * \ingroup arithm */
00113 void imProcessSplitComplex(const imImage* src_image, imImage*
00114
00115 /** Merges two images as the real and imaginary parts of a com
00116 * or as magnitude and phase parts (polar = 1). \n
00117 * Source images must be IM_FLOAT, destiny image must be IM_CF
00118 *
00119 * \verbatim im.ProcessMergeComplex(src_image1: imImage, src_i
00120 * \verbatim im.ProcessMergeComplexNew(image1: imImage, image2
00121 * \ingroup arithm */
00122 void imProcessMergeComplex(const imImage* src_image1, const im
00123
00124 /** Calculates the mean of multiple images. \n
00125 * Images must match size and type.
00126 *
00127 * \verbatim im.ProcessMultipleMean(src_image_list: table of i
00128 * \verbatim im.ProcessMultipleMeanNew(src_image_list: table o
00129 * \ingroup arithm */
00130 void imProcessMultipleMean(const imImage** src_image_list, int
00131
00132 /** Calculates the standard deviation of multiple images. \n
00133 * Images must match size and type. Use \ref imProcessMultiple

```

```

00134 *
00135 * \verbatim im.ProcessMultipleStdDev(src_image_list: table of
00136 * \verbatim im.ProcessMultipleStdDevNew(src_image_list: table
00137 * \ingroup arithm */
00138 void imProcessMultipleStdDev(const imImage** src_image_list, i
00139
00140 /** Calculates the auto-covariance of an image with the mean o
00141 * Images must match size and type. Returns zero if the counte
00142 *
00143 * \verbatim im.ProcessAutoCovariance(src_image: imImage, mean
00144 * \verbatim im.ProcessAutoCovarianceNew(src_image: imImage, m
00145 * \ingroup arithm */
00146 int imProcessAutoCovariance(const imImage* src_image, const im
00147
00148 /** Multiplies the conjugate of one complex image with another
00149 * Images must match size. Conj(img1) * img2 \n
00150 * Can be done in-place.
00151 *
00152 * \verbatim im.ProcessMultiplyConj(src_image1: imImage, src_i
00153 * \verbatim im.ProcessMultiplyConjNew(src_image1: imImage, sr
00154 * \ingroup arithm */
00155 void imProcessMultiplyConj(const imImage* src_image1, const im
00156
00157
00158
00159 /** \defgroup quantize Additional Image Quantization Operation
00160 * \par
00161 * Additionally operations to the \ref imConvertColorSpace fun
00162 * \par
00163 * See \ref im_process_pon.h
00164 * \ingroup process */
00165
00166 /** Converts a RGB image to a MAP image using uniform quantiza
00167 * with an optional 8x8 ordered dither. The RGB image must hav
00168 *
00169 * \verbatim im.ProcessQuantizeRGBUniform(src_image: imImage,
00170 * \verbatim im.ProcessQuantizeRGBUniformNew(src_image: imImag
00171 * \ingroup quantize */
00172 void imProcessQuantizeRGBUniform(const imImage* src_image, imI
00173
00174 /** Quantizes a gray scale image in less that 256 grays using
00175 * Both images must be IM_BYTE/IM_GRAY. Can be done in place.
00176 *
00177 * \verbatim im.ProcessQuantizeGrayUniform(src_image: imImage,
00178 * \verbatim im.ProcessQuantizeGrayUniformNew(src_image: imIma
00179 * \ingroup quantize */
00180 void imProcessQuantizeGrayUniform(const imImage* src_image, im

```

```

00181
00182
00183
00184 /** \defgroup histo Histogram Based Operations
00185 * \par
00186 * See \ref im_process_pon.h
00187 * \ingroup process */
00188
00189 /** Performs an histogram expansion. \n
00190 * Percentage defines an amount of pixels to include at start
00191 * If its is zero only empty counts of the histogram will be c
00192 * Images must be IM_BYTE/(IM_RGB or IM_GRAY). Can be done in
00193 *
00194 * \verbatim im.ProcessExpandHistogram(src_image: imImage, dst
00195 * \verbatim im.ProcessExpandHistogramNew(src_image: imImage,
00196 * \ingroup histo */
00197 void imProcessExpandHistogram(const imImage* src_image, imImag
00198
00199 /** Performs an histogram equalization. \n
00200 * Images must be IM_BYTE/(IM_RGB or IM_GRAY). Can be done in
00201 *
00202 * \verbatim im.ProcessEqualizeHistogram(src_image: imImage, d
00203 * \verbatim im.ProcessEqualizeHistogramNew(src_image: imImage
00204 * \ingroup histo */
00205 void imProcessEqualizeHistogram(const imImage* src_image, imIm
00206
00207
00208
00209 /** \defgroup colorproc Color Processing Operations
00210 * \par
00211 * Operations to change the color components configuration.
00212 * \par
00213 * See \ref im_process_pon.h
00214 * \ingroup process */
00215
00216 /** Split a RGB image into luma and chroma. \n
00217 * Chroma is calculated as R-Y,G-Y,B-Y. Source image must be I
00218 * luma image is IM_GRAY/IM_BYTE and chroma is IM_RGB/IM_BYTE.
00219 * Source and destiny must have the same size.
00220 *
00221 * \verbatim im.ProcessSplitYChroma(src_image: imImage, y_imag
00222 * \verbatim im.ProcessSplitYChromaNew(src_image: imImage) ->
00223 * \ingroup colorproc */
00224 void imProcessSplitYChroma(const imImage* src_image, imImage*
00225
00226 /** Split a RGB image into HSI planes. \n
00227 * Source image must be IM_RGB/IM_BYTE,IM_FLOAT. Destiny image

```

```

00228 * Source images must normalized to 0-1 if type is IM_FLOAT (\
00229 * Source and destiny must have the same size.
00230 *
00231 * \verbatim im.ProcessSplitHSI(src_image: imImage, h_image: i
00232 * \verbatim im.ProcessSplitHSINew(src_image: imImage) -> h_im
00233 * \ingroup colorproc */
00234 void imProcessSplitHSI(const imImage* src_image, imImage* h_im
00235
00236 /** Merge HSI planes into a RGB image. \n
00237 * Source images must be IM_GRAY/IM_FLOAT. Destiny image can b
00238 * Source and destiny must have the same size. See \ref hsi fo
00239 *
00240 * \verbatim im.ProcessMergeHSI(h_image: imImage, s_image: imI
00241 * \verbatim im.ProcessMergeHSINew(h_image: imImage, s_image:
00242 * \ingroup colorproc */
00243 void imProcessMergeHSI(const imImage* h_image, const imImage*
00244
00245 /** Split a multicomponent image into separate components.\n
00246 * Destiny images must be IM_GRAY. Size and data types must be
00247 * The number of destiny images must match the depth of the so
00248 *
00249 * \verbatim im.ProcessSplitComponents(src_image: imImage, dst
00250 * \verbatim im.ProcessSplitComponentsNew(src_image: imImage)
00251 * \ingroup colorproc */
00252 void imProcessSplitComponents(const imImage* src_image, imImag
00253
00254 /** Merges separate components into a multicomponent image.\n
00255 * Source images must be IM_GRAY. Size and data types must be
00256 * The number of source images must match the depth of the des
00257 *
00258 * \verbatim im.ProcessMergeComponents(src_image_list: table o
00259 * \verbatim im.ProcessMergeComponentsNew(src_image_list: tabl
00260 * \ingroup colorproc */
00261 void imProcessMergeComponents(const imImage** src_image_list,
00262
00263 /** Normalize the color components by their sum. Example: c1 =
00264 * Destiny image must be IM_FLOAT.
00265 *
00266 * \verbatim im.ProcessNormalizeComponents(src_image: imImage,
00267 * \verbatim im.ProcessNormalizeComponentsNew(src_image: imIma
00268 * \ingroup colorproc */
00269 void imProcessNormalizeComponents(const imImage* src_image, im
00270
00271 /** Replaces the source color by the destiny color. \n
00272 * The color will be type casted to the image data type. \n
00273 * The colors must have the same number of components of the i
00274 * Supports all color spaces and all data types except IM_CFLC

```

```

00275 *
00276 * \verbatim im.ProcessReplaceColor(src_image: imImage, dst_im
00277 * \verbatim im.ProcessReplaceColorNew(src_image: imImage, src
00278 * \ingroup colorproc */
00279 void imProcessReplaceColor(const imImage* src_image, imImage*
00280
00281
00282
00283 /** \defgroup logic Logical Arithmetic Operations
00284 * \par
00285 * Logical binary math operations for images.
00286 * \par
00287 * See \ref im_process_pon.h
00288 * \ingroup process */
00289
00290 /** Logical Operations.
00291 * \ingroup logic */
00292 enum imLogicOp {
00293     IM_BIT_AND,    /**< and = a & b */
00294     IM_BIT_OR,    /**< or = a | b */
00295     IM_BIT_XOR    /**< xor = ~(a | b) */
00296 };
00297
00298 /** Apply a logical operation.\n
00299 * Images must have data type IM_BYTE, IM_USHORT or IM_INT. Ca
00300 *
00301 * \verbatim im.ProcessBitwiseOp(src_image1: imImage, src_imag
00302 * \verbatim im.ProcessBitwiseOpNew(src_image1: imImage, src_i
00303 * \ingroup logic */
00304 void imProcessBitwiseOp(const imImage* src_image1, const imIma
00305
00306 /** Apply a logical NOT operation.\n
00307 * Images must have data type IM_BYTE, IM_USHORT or IM_INT. Ca
00308 *
00309 * \verbatim im.ProcessBitwiseNot(src_image: imImage, dst_imag
00310 * \verbatim im.ProcessBitwiseNotNew(src_image: imImage) -> ne
00311 * \ingroup logic */
00312 void imProcessBitwiseNot(const imImage* src_image, imImage* ds
00313
00314 /** Apply a bit mask. \n
00315 * The same as imProcessBitwiseOp but the second image is repl
00316 * Images must have data type IM_BYTE. It is valid only for AN
00317 *
00318 * \verbatim im.ProcessBitMask(src_image: imImage, dst_image:
00319 * \verbatim im.ProcessBitMaskNew(src_image: imImage, mask: st
00320 * In Lua, mask is a string with 0s and 1s, for example: "1100
00321 * \ingroup logic */

```

```

00322 void imProcessBitMask(const imImage* src_image, imImage* dst_i
00323
00324 /** Extract or Reset a bit plane. For ex: 000X0000 or XXX0XXXX
00325 * Images must have data type IM_BYTE. Can be done in place.
00326 *
00327 * \verbatim im.ProcessBitPlane(src_image: imImage, dst_image:
00328 * \verbatim im.ProcessBitPlaneNew(src_image: imImage, plane:
00329 * \ingroup logic */
00330 void imProcessBitPlane(const imImage* src_image, imImage* dst_
00331
00332
00333
00334 /** \defgroup render Synthetic Image Render
00335 * \par
00336 * Renders some 2D mathematical functions as images. All the f
00337 * and supports all data types except IM_CFLOAT.
00338 * \par
00339 * See \ref im_process_pon.h
00340 * \ingroup process */
00341
00342 /** Render Funtion.
00343 * \verbatim render_func(x: number, y: number, d: number, para
00344 * \ingroup render */
00345 typedef float (*imRenderFunc)(int x, int y, int d, float* para
00346
00347 /** Render Conditional Funtion.
00348 * \verbatim render_cond_func(x: number, y: number, d: number,
00349 * \ingroup render */
00350 typedef float (*imRenderCondFunc)(int x, int y, int d, int *co
00351
00352 /** Render a synthetic image using a render function. \n
00353 * plus will make the render be added to the current image dat
00354 * or else all data will be replaced. All the render functions
00355 * Returns zero if the counter aborted.
00356 *
00357 * \verbatim im.ProcessRenderOp(image: imImage, render_func: f
00358 * \ingroup render */
00359 int imProcessRenderOp(imImage* image, imRenderFunc render_func
00360
00361 /** Render a synthetic image using a conditional render functi
00362 * Data will be rendered only if the condional param is true.
00363 * Returns zero if the counter aborted.
00364 *
00365 * \verbatim im.ProcessRenderCondOp(image: imImage, render_con
00366 * \ingroup render */
00367 int imProcessRenderCondOp(imImage* image, imRenderCondFunc ren
00368

```

```

00369 /** Render speckle noise on existing data. Can be done in plac
00370 *
00371 * \verbatim im.ProcessRenderAddSpeckleNoise(src_image: imImag
00372 * \verbatim im.ProcessRenderAddSpeckleNoiseNew(src_image: imI
00373 * \ingroup render */
00374 int imProcessRenderAddSpeckleNoise(const imImage* src_image, i
00375
00376 /** Render gaussian noise on existing data. Can be done in pla
00377 *
00378 * \verbatim im.ProcessRenderAddGaussianNoise(src_image: imIma
00379 * \verbatim im.ProcessRenderAddGaussianNoiseNew(src_image: im
00380 * \ingroup render */
00381 int imProcessRenderAddGaussianNoise(const imImage* src_image,
00382
00383 /** Render uniform noise on existing data. Can be done in plac
00384 *
00385 * \verbatim im.ProcessRenderAddUniformNoise(src_image: imImag
00386 * \verbatim im.ProcessRenderAddUniformNoiseNew(src_image: imI
00387 * \ingroup render */
00388 int imProcessRenderAddUniformNoise(const imImage* src_image, i
00389
00390 /** Render random noise.
00391 *
00392 * \verbatim im.ProcessRenderRandomNoise(image: imImage) -> co
00393 * \ingroup render */
00394 int imProcessRenderRandomNoise(imImage* image);
00395
00396 /** Render a constant. The number of values must match the dep
00397 *
00398 * \verbatim im.ProcessRenderConstant(image: imImage, value: t
00399 * \ingroup render */
00400 int imProcessRenderConstant(imImage* image, float* value);
00401
00402 /** Render a centered wheel.
00403 *
00404 * \verbatim im.ProcessRenderWheel(image: imImage, internal_ra
00405 * \ingroup render */
00406 int imProcessRenderWheel(imImage* image, int internal_radius,
00407
00408 /** Render a centered cone.
00409 *
00410 * \verbatim im.ProcessRenderCone(image: imImage, radius: numb
00411 * \ingroup render */
00412 int imProcessRenderCone(imImage* image, int radius);
00413
00414 /** Render a centered tent.
00415 *

```

```

00416 * \verbatim im.ProcessRenderTent(image: imImage, tent_width:
00417 * \ingroup render */
00418 int imProcessRenderTent(imImage* image, int tent_width, int te
00419
00420 /** Render a ramp. Direction can be vertical (1) or horizontal
00421 *
00422 * \verbatim im.ProcessRenderRamp(image: imImage, start: numbe
00423 * \ingroup render */
00424 int imProcessRenderRamp(imImage* image, int start, int end, in
00425
00426 /** Render a centered box.
00427 *
00428 * \verbatim im.ProcessRenderBox(image: imImage, box_width: nu
00429 * \ingroup render */
00430 int imProcessRenderBox(imImage* image, int box_width, int box_
00431
00432 /** Render a centered sinc.
00433 *
00434 * \verbatim im.ProcessRenderSinc(image: imImage, x_period: nu
00435 * \ingroup render */
00436 int imProcessRenderSinc(imImage* image, float x_period, float
00437
00438 /** Render a centered gaussian.
00439 *
00440 * \verbatim im.ProcessRenderGaussian(image: imImage, stddev:
00441 * \ingroup render */
00442 int imProcessRenderGaussian(imImage* image, float stddev);
00443
00444 /** Render the laplacian of a centered gaussian.
00445 *
00446 * \verbatim im.ProcessRenderLapOfGaussian(image: imImage, std
00447 * \ingroup render */
00448 int imProcessRenderLapOfGaussian(imImage* image, float stddev)
00449
00450 /** Render a centered cosine.
00451 *
00452 * \verbatim im.ProcessRenderCosine(image: imImage, x_period:
00453 * \ingroup render */
00454 int imProcessRenderCosine(imImage* image, float x_period, floa
00455
00456 /** Render a centered grid.
00457 *
00458 * \verbatim im.ProcessRenderGrid(image: imImage, x_space: num
00459 * \ingroup render */
00460 int imProcessRenderGrid(imImage* image, int x_space, int y_spa
00461
00462 /** Render a centered chessboard.

```

```

00463 *
00464 * \verbatim im.ProcessRenderChessboard(image: imImage, x_space
00465 * \ingroup render */
00466 int imProcessRenderChessboard(imImage* image, int x_space, int
00467
00468
00469
00470 /** \defgroup tonegamut Tone Gamut Operations
00471 * \par
00472 * Operations that try to preserve the min-max interval in the
00473 * \par
00474 * See \ref im_process_pon.h
00475 * \ingroup process */
00476
00477
00478 /** Tone Gamut Operations.
00479 * \ingroup tonegamut */
00480 enum imToneGamut {
00481     IM_GAMUT_NORMALIZE, /**< normalize = (a-min) / (max-min)
00482     IM_GAMUT_POW,      /**< pow      = ((a-min) / (max-min))^g
00483                        param[0]=gamma
00484     IM_GAMUT_LOG,     /**< log      = log(K * (a-min) / (max-
00485                        param[0]=K      (K>0)
00486     IM_GAMUT_EXP,     /**< exp      = (exp(K * (a-min) / (max
00487                        param[0]=K
00488     IM_GAMUT_INVERT,  /**< invert   = max - (a-min)
00489     IM_GAMUT_ZEROSTART, /**< zerostart = a - min
00490     IM_GAMUT_SOLARIZE, /**< solarize = a < level ? a: (level
00491                        param[0]=level percenta
00492                        photography solarizatio
00493     IM_GAMUT_SLICE,   /**< slice    = start < a || a > end ?
00494                        param[0]=start, param[
00495     IM_GAMUT_EXPAND,  /**< expand    = a < start ? min: a > e
00496                        param[0]=start, param[
00497     IM_GAMUT_CROP,    /**< crop     = a < start ? start: a >
00498                        param[0]=start, param[
00499     IM_GAMUT_BRIGHTCONT /**< brightcont = a < min ? min: a > m
00500                        param[0]=bright_shift
00501                        change brightness and
00502 };
00503
00504 /** Apply a gamut operation with arguments. \n
00505 * Supports all data types except IM_CFLOAT. \n
00506 * The linear operation do a special conversion when min > 0 a
00507 * IM_BYTE images have min=0 and max=255 always. \n
00508 * Can be done in place. When there is no extra params use NUL
00509 *

```

```

00510 * \verbatim im.ProcessToneGamut(src_image: imImage, dst_image
00511 * \verbatim im.ProcessToneGamutNew(src_image: imImage, op: nu
00512 * \ingroup tonegamut */
00513 void imProcessToneGamut(const imImage* src_image, imImage* dst
00514
00515 /** Converts from (0-1) to (0-255), crop out of bounds values.
00516 * Source image must be IM_FLOAT, and destiny image must be IM
00517 *
00518 * \verbatim im.ProcessUnNormalize(src_image: imImage, dst_ima
00519 * \verbatim im.ProcessUnNormalizeNew(src_image: imImage) -> n
00520 * \ingroup tonegamut */
00521 void imProcessUnNormalize(const imImage* src_image, imImage* d
00522
00523 /** Directly converts IM_USHORT, IM_INT and IM_FLOAT into IM_B
00524 * This can also be done using \ref imConvertDataType with IM_
00525 *
00526 * \verbatim im.ProcessDirectConv(src_image: imImage, dst_imag
00527 * \verbatim im.ProcessDirectConvNew(src_image: imImage) -> ne
00528 * \ingroup tonegamut */
00529 void imProcessDirectConv(const imImage* src_image, imImage* ds
00530
00531 /** A negative effect. Uses \ref imProcessToneGamut with IM_GA
00532 * Supports all color spaces and all data types except IM_CFLC
00533 *
00534 * \verbatim im.ProcessNegative(src_image: imImage, dst_image:
00535 * \verbatim im.ProcessNegativeNew(src_image: imImage) -> new_
00536 * \ingroup tonegamut */
00537 void imProcessNegative(const imImage* src_image, imImage* dst_
00538
00539
00540
00541 /** \defgroup threshold Threshold Operations
00542 * \par
00543 * Operations that converts a usually IM_GRAY/IM_BYTE image in
00544 * \par
00545 * See \ref im_process_pon.h
00546 * \ingroup process */
00547
00548 /** Apply a manual threshold. \n
00549 * threshold = a <= level ? 0: value \n
00550 * Normal value is 1 but another common value is 255. Can be d
00551 * Supports all integer IM_GRAY images as source, and IM_BINAR
00552 *
00553 * \verbatim im.ProcessThreshold(src_image: imImage, dst_image
00554 * \verbatim im.ProcessThresholdNew(src_image: imImage, level:
00555 * \ingroup threshold */
00556 void imProcessThreshold(const imImage* src_image, imImage* dst

```

```

00557
00558 /** Apply a threshold by the difference of two images. \n
00559 * threshold = a1 <= a2 ? 0: 1  \n
00560 * Can be done in place.
00561 *
00562 * \verbatim im.ProcessThresholdByDiff(src_image1: imImage, sr
00563 * \verbatim im.ProcessThresholdByDiffNew(src_image1: imImage,
00564 * \ingroup threshold */
00565 void imProcessThresholdByDiff(const imImage* src_image1, const
00566
00567 /** Apply a threshold by the Hysteresis method. \n
00568 * Hysteresis thersholding of edge pixels. Starting at pixels
00569 * value greater than the HIGH threshold, trace a connected se
00570 * of pixels that have a value greater than the LOW threhsold.
00571 * Note: could not find the original source code author name.
00572 *
00573 * \verbatim im.ProcessHysteresisThreshold(src_image: imImage,
00574 * \verbatim im.ProcessHysteresisThresholdNew(src_image: imIma
00575 * \ingroup threshold */
00576 void imProcessHysteresisThreshold(const imImage* src_image, im
00577
00578 /** Estimates hysteresis low and high threshold levels. \n
00579 * Usefull for \ref imProcessHysteresisThreshold.
00580 *
00581 * \verbatim im.ProcessHysteresisThresEstimate(image: imImage)
00582 * \ingroup threshold */
00583 void imProcessHysteresisThresEstimate(const imImage* image, in
00584
00585 /** Calculates the threshold level for manual threshold using
00586 * Extracted from XITE, Copyright 1991, Blab, UiO \n
00587 * http://www.ifi.uio.no/~blab/Software/Xite/
00588 \verbatim
00589     Reference:
00590         S. M. Dunn & D. Harwood & L. S. Davis:
00591         "Local Estimation of the Uniform Error Threshold"
00592         IEEE Trans. on PAMI, Vol PAMI-6, No 6, Nov 1984.
00593     Comments: It only works well on images whith large objects.
00594     Author: Olav Borgli, BLAB, ifi, UiO
00595     Image processing lab, Department of Informatics, University
00596 \endverbatim
00597 * Returns the used level.
00598 *
00599 * \verbatim im.ProcessUniformErrThreshold(src_image: imImage,
00600 * \verbatim im.ProcessUniformErrThresholdNew(src_image: imIma
00601 * \ingroup threshold */
00602 int imProcessUniformErrThreshold(const imImage* src_image, imI
00603

```

```

00604 /** Apply a dithering on each image channel by using a difusio
00605  * It can be applied on any IM_BYTE images. It will "threshold
00606  * source and destiny must be of the same depth.
00607  *
00608  * \verbatim im.ProcessDifusionErrThreshold(src_image: imImage
00609  * \verbatim im.ProcessDifusionErrThresholdNew(src_image: imIm
00610  * \ingroup threshold */
00611 void imProcessDifusionErrThreshold(const imImage* src_image, i
00612
00613 /** Calculates the threshold level for manual threshold using
00614  * that should stay bellow the threshold. \n
00615  * Returns the used level.
00616  *
00617  * \verbatim im.ProcessPercentThreshold(src_image: imImage, ds
00618  * \verbatim im.ProcessPercentThresholdNew(src_image: imImage,
00619  * \ingroup threshold */
00620 int imProcessPercentThreshold(const imImage* src_image, imImag
00621
00622 /** Calculates the threshold level for manual threshold using
00623  * Returns the used level. \n
00624  * Original implementation by Flavio Szenberg.
00625  *
00626  * \verbatim im.ProcessOtsuThreshold(src_image: imImage, dst_i
00627  * \verbatim im.ProcessOtsuThresholdNew(src_image: imImage) ->
00628  * \ingroup threshold */
00629 int imProcessOtsuThreshold(const imImage* src_image, imImage*
00630
00631 /** Calculates the threshold level for manual threshold using
00632  * Returns the used level. \n
00633  * Supports all integer IM_GRAY images as source, and IM_BINAR
00634  *
00635  * \verbatim im.ProcessMinMaxThreshold(src_image: imImage, dst
00636  * \verbatim im.ProcessMinMaxThresholdNew(src_image: imImage)
00637  * \ingroup threshold */
00638 int imProcessMinMaxThreshold(const imImage* src_image, imImage
00639
00640 /** Estimates Local Max threshold level for IM_BYTE images.
00641  *
00642  * \verbatim im.ProcessLocalMaxThresEstimate(image: imImage) -
00643  * \ingroup threshold */
00644 void imProcessLocalMaxThresEstimate(const imImage* image, int
00645
00646 /** Apply a manual threshold using an interval. \n
00647  * threshold = start_level <= a <= end_level ? 1: 0 \n
00648  * Normal value is 1 but another common value is 255. Can be d
00649  * Supports all integer IM_GRAY images as source, and IM_BINAR
00650  *

```

```

00651 * \verbatim im.ProcessSliceThreshold(src_image: imImage, dst_
00652 * \verbatim im.ProcessSliceThresholdNew(src_image: imImage, s
00653 * \ingroup threshold */
00654 void imProcessSliceThreshold(const imImage* src_image, imImage
00655
00656
00657 /** \defgroup effects Special Effects
00658 * \par
00659 * Operations to change image appearance.
00660 * \par
00661 * See \ref im_process_pon.h
00662 * \ingroup process */
00663
00664
00665 /** Generates a zoom in effect averaging colors inside a squar
00666 * Operates only on IM_BYTE images.
00667 *
00668 * \verbatim im.ProcessPixelate(src_image: imImage, dst_image:
00669 * \verbatim im.ProcessPixelateNew(src_image: imImage, box_siz
00670 * \ingroup effects */
00671 void imProcessPixelate(const imImage* src_image, imImage* dst_
00672
00673 /** A simple Posterize effect. It reduces the number of colors
00674 * less significant bit planes. Can have 1 to 7 levels. See \r
00675 * Image data type must be integer.
00676 *
00677 * \verbatim im.ProcessPosterize(src_image: imImage, dst_image
00678 * \verbatim im.ProcessPosterizeNew(src_image: imImage, level:
00679 * \ingroup effects */
00680 void imProcessPosterize(const imImage* src_image, imImage* dst
00681
00682
00683
00684 #if defined(__cplusplus)
00685 }
00686 #endif
00687
00688 #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: im_raw.h,v 1.2 2005/12/10 00:54:39 scuri 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  *
00018  * \verbatim im.FileOpenRaw(file_name: string) -> ifile: imFil
00019  * \ingroup raw */
00020 imFile* imFileOpenRaw(const char* file_name, int *error);
00021
00022 /** Creates a RAW image file.
00023  *
00024  * \verbatim im.FileNewRaw(file_name: string) -> ifile: imFile
00025  * \ingroup raw */
00026 imFile* imFileNewRaw(const char* file_name, int *error);
00027
00028
00029 #if defined(__cplusplus)
00030 }
00031 #endif
00032
00033 #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: im_util.h,v 1.5 2005/12/12 00:18:29 scuri 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 Raw Data Utilities
00048  * \par
00049  * See \ref im_util.h
00050  * \ingroup imagerep */
00051
00052 /** Returns the size of the data buffer.
00053  *
00054  * \verbatim im.ImageDataSize(width: number, height: number, c
00055  * \ingroup imageutil */
00056 int imImageDataSize(int width, int height, int color_mode, int
00057
00058 /** Returns the size of one line of the data buffer. \n
00059  * This depends if the components are packed. If packed includ
00060  *
00061  * \verbatim im.ImageLineSize(width: number, color_mode: numbe
00062  * \ingroup imageutil */
00063 int imImageLineSize(int width, int color_mode, int data_type);
00064
00065 /** Returns the number of elements of one line of the data buf
00066  * This depends if the components are packed. If packed includ
00067  *
00068  * \verbatim im.ImageLineCount(width: number, color_mode: numb
00069  * \ingroup imageutil */
00070 int imImageLineCount(int width, int color_mode);
00071
00072 /** Check if the combination color_mode+data_type is valid.
00073  *
00074  * \verbatim im.ImageCheckFormat(color_mode: number, data_type
00075  * \ingroup imageutil */
00076 int imImageCheckFormat(int color_mode, int data_type);
00077
00078
00079
00080 /** \defgroup colorutil Color Utilities
00081  * \par
00082  * See \ref im_util.h
00083  * \ingroup util */
00084
00085 /** Encode RGB components in a long for palette usage. \n
00086  * "long" definition is compatible with the CD library definit

```

```

00087 *
00088 * \verbatim im.ColorEncode(red: number, green: number, blue:
00089 * \ingroup colorutil */
00090 long imColorEncode(unsigned char red, unsigned char green, uns
00091
00092 /** Decode RGB components from a long for palette usage. \n
00093 * "long" definition is compatible with the CD library definit
00094 *
00095 * \verbatim im.ColorDecode(color: lightuserdata) -> red: numb
00096 * \ingroup colorutil */
00097 void imColorDecode(unsigned char *red, unsigned char *green, u
00098
00099
00100
00101 /** \defgroup colormodeutil Color Mode Utilities
00102 * \par
00103 * See \ref im_util.h
00104 * \ingroup imagerep */
00105
00106 /** Returns the color mode name.
00107 *
00108 * \verbatim im.ColorModeSpaceName(color_mode: number) -> name
00109 * \ingroup colormodeutil */
00110 const char* imColorModeSpaceName(int color_mode);
00111
00112 /** Returns the number of components of the color space includ
00113 *
00114 * \verbatim im.ColorModeDepth(color_mode: number) -> depth: n
00115 * \ingroup colormodeutil */
00116 int imColorModeDepth(int color_mode);
00117
00118 /** Returns the color space of the color mode.
00119 *
00120 * \verbatim im.ColorModeSpace(color_mode: number) -> color_sp
00121 * \ingroup colormodeutil */
00122 #define imColorModeSpace(_cm) (_cm & 0xFF)
00123
00124 /** Check if the two color modes match. Only the color space i
00125 *
00126 * \verbatim im.ColorModeMatch(color_mode1: number, color_mode
00127 * \ingroup colormodeutil */
00128 #define imColorModeMatch(_cm1, _cm2) (imColorModeSpace(_cm1) =
00129
00130 /** Check if the color mode has an alpha channel.
00131 *
00132 * \verbatim im.ColorModeHasAlpha(color_mode: number) -> has_a
00133 * \ingroup colormodeutil */

```

```

00134 #define imColorModeHasAlpha(_cm) (_cm & IM_ALPHA)
00135
00136 /** Check if the color mode components are packed in one plane
00137  *
00138  * \verbatim im.ColorModeIsPacked(color_mode: number) -> is_pa
00139  * \ingroup colormodeutil */
00140 #define imColorModeIsPacked(_cm) (_cm & IM_PACKED)
00141
00142 /** Check if the color mode orients the image from top down to
00143  *
00144  * \verbatim im.ColorModeIsTopDown(color_mode: number) -> is_t
00145  * \ingroup colormodeutil */
00146 #define imColorModeIsTopDown(_cm) (_cm & IM_TOPDOWN)
00147
00148 /** Returns the color space of the equivalent display bitmap i
00149  * Original packing and alpha are ignored. Returns IM_RGB, IM_
00150  *
00151  * \verbatim im.ColorModeToBitmap(color_mode: number) -> color
00152  * \ingroup colormodeutil */
00153 int imColorModeToBitmap(int color_mode);
00154
00155 /** Check if the color mode and data_type defines a display bi
00156  *
00157  * \verbatim im.ColorModeIsBitmap(color_mode: number, data_typ
00158  * \ingroup colormodeutil */
00159 int imColorModeIsBitmap(int color_mode, int data_type);
00160
00161
00162
00163 /** \defgroup datatypeutil Data Type Utilities
00164  * \par
00165  * See \ref im_util.h
00166  * \ingroup util
00167  * @{
00168  */
00169
00170 typedef unsigned char imbyte;
00171 typedef unsigned short imushort;
00172
00173 #define IM_BYTECROP(_v) (_v < 0? 0: _v > 255? 255: _v)
00174 #define IM_CROPMAX(_v, _max) (_v < 0? 0: _v > _max? _max: _v)
00175
00176 /** @} */
00177
00178 /** Returns the size in bytes of a specified numeric data type
00179  *
00180  * \verbatim im.DataTypeSize(data_type: number) -> size: numbe

```

```

00181 * \ingroup datatypeutil */
00182 int imDataTypeSize(int data_type);
00183
00184 /** Returns the numeric data type name given its identifier.
00185 *
00186 * \verbatim im.DataTypeName(data_type: number) -> name: string
00187 * \ingroup datatypeutil */
00188 const char* imDataTypeName(int data_type);
00189
00190 /** Returns the maximum value of an integer data type. For floating point
00191 *
00192 * \verbatim im.DataTypeIntMax(data_type: number) -> int_max: int
00193 * \ingroup datatypeutil */
00194 unsigned long imDataTypeIntMax(int data_type);
00195
00196 /** Returns the minimum value of an integer data type. For floating point
00197 *
00198 * \verbatim im.DataTypeIntMin(data_type: number) -> int_min: int
00199 * \ingroup datatypeutil */
00200 long imDataTypeIntMin(int data_type);
00201
00202
00203
00204 /** \defgroup bin Binary Data Utilities
00205 * \par
00206 * See \ref im_util.h
00207 * \ingroup util */
00208
00209 /** CPU Byte Orders.
00210 * \ingroup bin */
00211 enum imByteOrder
00212 {
00213     IM_LITTLEENDIAN, /**< Little Endian - The most significant byte is at the lowest address
00214     IM_BIGENDIAN     /**< Big Endian - The most significant byte is at the highest address
00215 };
00216
00217 /** Returns the current CPU byte order.
00218 * \ingroup bin */
00219 int imBinCPUByteOrder(void);
00220
00221 /** Changes the byte order of an array of 2, 4 or 8 byte values.
00222 * \ingroup bin */
00223 void imBinSwapBytes(void *data, int count, int size);
00224
00225 /** Changes the byte order of an array of 2 byte values.
00226 * \ingroup bin */
00227 void imBinSwapBytes2(void *data, int count);

```

```

00228
00229 /** Inverts the byte order of the 4 byte values
00230 * \ingroup bin */
00231 void imBinSwapBytes4(void *data, int count);
00232
00233 /** Inverts the byte order of the 8 byte values
00234 * \ingroup bin */
00235 void imBinSwapBytes8(void *data, int count);
00236
00237
00238
00239 /** \defgroup compress Data Compression Utilities
00240 * \par
00241 * Deflate compression support uses zlib version 1.2.3.      \n
00242 * http://www.zlib.org/                                       \
00243 * Copyright (C) 1995-2004 Jean-loup Gailly and Mark Adler
00244 * \par
00245 * LZF compression support uses libLZF version 1.51.        \n
00246 * http://liblzf.plan9.de/                                    \n
00247 * Copyright (C) 2000-2005 Marc Alexander Lehmann
00248 * See \ref im_util.h
00249 * \ingroup util */
00250
00251 /** Compresses the data using the ZLIB Deflate compression. \n
00252 * The destination buffer must be at least 0.1% larger than so
00253 * It compresses raw byte data. zip_quality can be 1 to 9. \n
00254 * Returns the size of the compressed buffer or zero if failed
00255 * \ingroup compress */
00256 int imCompressDataZ(const void* src_data, int src_size, void*
00257
00258 /** Uncompresses the data compressed with the ZLIB Deflate com
00259 * Returns zero if failed.
00260 * \ingroup compress */
00261 int imCompressDataUnZ(const void* src_data, int src_size, void
00262
00263 /** Compresses the data using the libLZF compression. \n
00264 * Returns the size of the compressed buffer or zero if failed
00265 * \ingroup compress */
00266 int imCompressDataLZF(const void* src_data, int src_size, void
00267
00268 /** Uncompresses the data compressed with the libLZF compressi
00269 * Returns zero if failed.
00270 * \ingroup compress */
00271 int imCompressDataUnLZF(const void* src_data, int src_size, vo
00272
00273
00274 #if defined(__cplusplus)

```

```
00275 }  
00276 #endif  
00277  
00278 #endif
```

include

imlua.h

[Go to the documentation of this file.](#)

```
00001 /** \file
00002  * \brief IM Lua 5 Binding
00003  *
00004  * See Copyright Notice in im_lib.h
00005  * $Id: imlua.h,v 1.5 2005/12/12 15:42:29 scuri Exp $
00006  */
00007
00008 #ifndef __IMLUA_H
00009 #define __IMLUA_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 /** \defgroup imlua IM Lua 5 Binding
00016  * \par
00017  * Binding for the Lua 5 scripting language. Works with versio
00018  * Lua 5.0.2 Copyright (C) 1994-2004 Tecgraf, PUC-Rio
00019  * Lua 5.1 Copyright (C) 1994-2005 Lua.org, PUC-Rio
00020  * R. Ierusalimschy, L. H. de Figueiredo & W. Celes
00021  * http://www.lua.org
00022  * \par
00023  * The name of the functions were changed because of the names
00024  * As a general rule use:
00025  \verbatim
00026     imXxx  -> im.Xxx
00027     IM_XXX -> im.XXX
00028     imFileXXX(ifile,... -> ifile:XXX(...
00029     imImageXXX(image,... -> image:XXX(...
00030  \endverbatim
00031  * All the objects are garbage collected by the Lua garbage co
00032  * \par
00033  * See \ref imlua.h
00034  * \ingroup util */
00035
00036 #ifdef LUA_NOOBJECT /* Lua 3 */
00037 void imlua_open(void);
00038 #endif
00039
```

```

00040 #ifndef LUA_TNONE /* Lua 5 */
00041
00042 /** Initializes the Lua binding of the main IM library. \n
00043 * You must link the application with the "imlua5" library.
00044 * \ingroup imlua */
00045 int imlua_open(lua_State *L);
00046
00047 /** Initializes the Lua binding of the capture library. \n
00048 * You must link the application with the "imlua_capture5" lib
00049 * \ingroup imlua */
00050 int imlua_open_capture(lua_State *L);
00051
00052 /** Initializes the Lua binding of the process library. \n
00053 * You must link the application with the "imlua_process5" lib
00054 * \ingroup imlua */
00055 int imlua_open_process(lua_State *L);
00056
00057 /** Initializes the Lua binding of the fourier transform libra
00058 * You must link the application with the "imlua_fftw5" librar
00059 * \ingroup imlua */
00060 int imlua_open_fftw(lua_State *L);
00061
00062 /** Initializes the Lua binding of additional functions for th
00063 * CD library is a 2D graphics library. \n
00064 * http://www.tecgraf.puc-rio.br/cd \n
00065 * You must link the application with the "imlua_cd5" library.
00066 *
00067 * The following functions are defined:
00068 * \verbatim bitmap:imImageCreate() -> image: imImage [in Lua
00069     Creates an imImage from a cdBitmap. \endverbatim
00070 *
00071 * \verbatim image:cdCreateBitmap() -> bitmap: cdBitmap [in Lu
00072     Creates a cdBitmap from an imImage.
00073     The imImage must be a bitmap image, see imImageIsBitmap.
00074 *
00075 * \verbatim image:cdInitBitmap() -> bitmap: cdBitmap [in Lua
00076     Creates a cdBitmap from an imImage, but reuses image data
00077     When the cdBitmap is destroyed the data is preserved. \en
00078 *
00079 * \verbatim image:cdPutImageRect(x: number, y: number, w: num
00080     Draws the imImage into the active cdCanvas.
00081     The imImage must be a bitmap image, see imImageIsBitmap.
00082 *
00083 * \verbatim image:wdPutImageRect(x: number, y: number, w: num
00084     Draws the imImage into the active cdCanvas using world co
00085     The imImage must be a bitmap image, see imImageIsBitmap.
00086 *

```

```
00087 * \verbatim image:cdGetImage(x: number, y: number) [in Lua 5]
00088     Retrieve the imImage data from the active cdCanvas.
00089     The imImage must be a IM_RGB/IM_BYTE image. \endverbatim
00090 *
00091 * \verbatim image:cdCreateCanvas([res: number]) -> canvas: cd
00092     Creates a cdCanvas using the CD_IMAGERGB driver. Resoluti
00093     The imImage must be a IM_RGB/IM_BYTE image. \endverbatim
00094 *
00095 * \ingroup imlua */
00096 int imlua\_open\_cd(lua_State *L);
00097
00098
00099 #endif
00100
00101 #ifdef __cplusplus
00102 }
00103 #endif
00104
00105 #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: old_im.h,v 1.1 2005/04/02 22:07:00 scuri 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 cha
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 \[code\]](#)
file [im_attrib.h \[code\]](#)
file [im_attrib_flat.h \[code\]](#)
file [im_binfile.h \[code\]](#)
file [im_capture.h \[code\]](#)
file [im_color.h \[code\]](#)
file [im_colorhsi.h \[code\]](#)
file [im_complex.h \[code\]](#)
file [im_convert.h \[code\]](#)
file [im_counter.h \[code\]](#)
file [im_dib.h \[code\]](#)
file [im_file.h \[code\]](#)
file [im_format.h \[code\]](#)
file [im_format_all.h \[code\]](#)
file [im_format_avi.h \[code\]](#)
file [im_format_jp2.h \[code\]](#)
file [im_format_raw.h \[code\]](#)
file [im_format_wmv.h \[code\]](#)
file [im_image.h \[code\]](#)
file [im_lib.h \[code\]](#)
file [im_math.h \[code\]](#)
file [im_math_op.h \[code\]](#)
file [im_palette.h \[code\]](#)
file [im_plus.h \[code\]](#)
file [im_process.h \[code\]](#)
file [im_process_ana.h \[code\]](#)
file [im_process_glo.h \[code\]](#)

file [im_process_loc.h](#) [code]
file [im_process_pon.h](#) [code]
file [im_raw.h](#) [code]
file [im_util.h](#) [code]
file [imlua.h](#) [code]
file [old_im.h](#) [code]

| [All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- C -

- `cos_op()` : [im_math_op.h](#)
- `crop_byte()` : [im_math_op.h](#)

| [All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- d -

- [diff_op\(\)](#) : [im_math_op.h](#)
- [div_op\(\)](#) : [im_math_op.h](#)

| [All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- e -

- `exp_op()` : [im_math_op.h](#)

[| All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- 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_FILEHANDLE : [im_binfile.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](#)
- `imCompressDataLZF()` : [im_util.h](#)
- `imCompressDataUnLZF()` : [im_util.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](#)
- `imFileImageLoad()` : [im_image.h](#)
- `imFileImageLoadBitmap()` : [im_image.h](#)
- `imFileImageSave()` : [im_image.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](#)
- `imFileLoadBitmapFrame()` : [im_image.h](#)
- `imFileLoadImage()` : [im_image.h](#)
- `imFileLoadImageFrame()` : [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](#)
- `imImageAddAlpha()` : [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](#)
- `imImageCreateBased()` : [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](#)
- `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](#)
- `implua_open()` : [implua.h](#)
- `implua_open_capture()` : [implua.h](#)
- `implua_open_cd()` : [implua.h](#)
- `implua_open_fftw()` : [implua.h](#)
- `implua_open_process()` : [implua.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**
- imProcessInsert() : **im_process_loc.h**
- imProcessLapOfGaussianConvolve() : **im_process_loc.h**
- imProcessLocalMaxThresEstimate() : **im_process_pon.h**
- imProcessLocalMaxThreshold() : **im_process_loc.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](#)
- `imPutBitmap` : [im_image.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](#)
- `imVideoCaptureFormatCount()` : [im_capture.h](#)
- `imVideoCaptureFrame()` : [im_capture.h](#)
- `imVideoCaptureGetAttribute()` : [im_capture.h](#)
- `imVideoCaptureGetAttributeList()` : [im_capture.h](#)
- `imVideoCaptureGetFormat()` : [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](#)
- `imVideoCaptureSetFormat()` : [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](#)

| [All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- | -

- `less_op()` : [im_math_op.h](#)
- `log_op()` : [im_math_op.h](#)

| [All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- m -

- `max_op()` : [im_math_op.h](#)
- `min_op()` : [im_math_op.h](#)
- `mul_op()` : [im_math_op.h](#)

| [All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- p -

- `pow_op()` : [im_math_op.h](#)

| [All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [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:

- 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](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- C -

- `cos_op()` : [im_math_op.h](#)
- `crop_byte()` : [im_math_op.h](#)

[| All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- d -

- `diff_op()` : [im_math_op.h](#)
- `div_op()` : [im_math_op.h](#)

[| All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- 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](#)

- `imCompressDataLZF()` : [im_util.h](#)
- `imCompressDataUnLZF()` : [im_util.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](#)
- `imFileImageLoad()` : [im_image.h](#)
- `imFileImageLoadBitmap()` : [im_image.h](#)
- `imFileImageSave()` : [im_image.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](#)
- `imFileLoadBitmapFrame()` : [im_image.h](#)
- `imFileLoadImage()` : [im_image.h](#)
- `imFileLoadImageFrame()` : [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](#)
- `imImageAddAlpha()` : [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](#)
- `imImageCreateBased()` : [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](#)
- `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](#)
- `implua_open()` : [implua.h](#)
- `implua_open_capture()` : [implua.h](#)
- `implua_open_cd()` : [implua.h](#)
- `implua_open_fftw()` : [implua.h](#)
- `implua_open_process()` : [implua.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](#)
- imProcessInsert() : [im_process_loc.h](#)
- imProcessLapOfGaussianConvolve() : [im_process_loc.h](#)
- imProcessLocalMaxThresEstimate() : [im_process_pon.h](#)
- imProcessLocalMaxThreshold() : [im_process_loc.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](#)
- `imVideoCaptureFormatCount()` : [im_capture.h](#)
- `imVideoCaptureFrame()` : [im_capture.h](#)
- `imVideoCaptureGetAttribute()` : [im_capture.h](#)
- `imVideoCaptureGetAttributeList()` : [im_capture.h](#)

- `imVideoCaptureGetFormat()` : [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](#)
- `imVideoCaptureSetFormat()` : [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](#)

[| All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- | -

- `less_op()` : [im_math_op.h](#)
- `log_op()` : [im_math_op.h](#)

[| All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- m -

- `max_op()` : [im_math_op.h](#)
- `min_op()` : [im_math_op.h](#)
- `mul_op()` : [im_math_op.h](#)

[| All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- p -

- `pow_op()` : [im_math_op.h](#)

[| All](#) | [Functions](#) | [Typedefs](#) | [Enumerations](#) | [Enumerator](#) | [Defines](#)
[a](#) | [c](#) | [d](#) | [e](#) | [i](#) | [l](#) | [m](#) | [p](#) | [s](#)

- 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](#)