

IM
Image Representation, Storage, Capture and Processing
Version 3.5

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.

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@tecgraf.puc-rio.br. 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 the **IM** support features at **SourceForge**: <http://sourceforge.net/projects/imtoolkit/>. It provides us Mailing List, CVS Repository and Downloads.

The discussion list is available at: <http://lists.sourceforge.net/lists/listinfo/imtoolkit-users>.

Source code, pre-compiled binaries and documentation can be downloaded at: http://sourceforge.net/project/showfiles.php?group_id=241318.

The CVS can be browsed at: <http://imtoolkit.cvs.sourceforge.net/imtoolkit/>.

If you want us to develop a specific feature for the toolkit, Tecgraf is available for partnerships and cooperation. Please contact tcg@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.

Library Author:

- Antonio Scuri

Thanks to the people that worked and contributed to the library:

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

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 M ller ([libExif](#))
- Glenn Randers-Pehrson ([libPNG](#))
- Jean-loup Gailly and Mark Adler ([zlib](#))
- Gershon Elber (GIFLib)
- Michael Adams ([libJasper](#))
- Svein B e, Tor L nnestad 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](#), [libexif.txt](#).

Thanks for the [SourceForge](#) for hosting the support features. Thanks for the [LuaForge](#) team for previously hosting the support features for many years.

IM is registered at the National Institute of Intellectual Property in Brazil (INPI) under the number 07570-6, and so it is protected against illegal use. See the [Tecgraf Library License](#) for further usage information and Copyright.

Documentation

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

The full documentation can be downloaded from the [Download Files](#). The documentation is also available in Adobe Acrobat and Windows HTML Help 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/>).

Publications

- Scuri, A. "IM - Imaging Toolkit". Software Developer's Journal. Jan/2006. [<http://en.sdjournal.org/products/articleInfo/25>]
- Scuri, A., "IM – An Imaging Tool", Poster, SIBGRAPI 2004 [[poster.pdf](#), [poster_text.pdf](#)]

Tecgraf Library License

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

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 paperwork, no royalties, no GNU-like "copyleft" restrictions, either. Just download and use it. 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, [PUC-Rio](#) keeps 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 and a link to our site in a web page for your product.

The libraries are designed, implemented and maintained 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.

Copyright © 1994-2009 [Tecgraf](#), [PUC-Rio](#).

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.

Download

The download site for pre-compiled binaries, documentation and sources is at **SourceForge**:

<http://sourceforge.net/projects/imtoolkit/files/>

Use this link for the latest version: <http://sourceforge.net/projects/imtoolkit/files/3.4.2/>

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/>

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>.

The **IM** files can be downloaded at http://sourceforge.net/project/showfiles.php?group_id=241318.

The **CD** files can be downloaded at http://sourceforge.net/project/showfiles.php?group_id=241317.

The **IUP** files can be downloaded at http://sourceforge.net/project/showfiles.php?group_id=241310.

The **Lua** files can be downloaded at http://luaforge.net/project/showfiles.php?group_id=110.

Build Configuration

Libraries and executables were built using speed optimization. In UNIX the dynamic libraries were NOT built with the `-fpic` parameter. In MacOS X the dynamic libraries are in bundle format. The source code along with the "config.mak" files for **Tecmake** are also available.

The DLLs were built using the **cdecl** calling convention. This should be a problem for Visual Basic users.

In Visual C++ we use the single thread C Run Time Library for static libraries and the multi thread C RTL for DLLs. Because this is the default in Visual Studio for new projects. Since Visual C++ 8, both use the multi thread C RTL.

Packaging

The package files available for download are named according to the platform where they were build.

In UNIX all strings are based in the result of the command "uname -a". The package name is a concatenation of the platform **uname**, the system **major** version number and the system **minor** version number. Some times a suffix must be added to complement the name. The compiler used is always gcc. Binaries for 64-bits receive the suffix: "_64". In Linux when there are different versions of gcc for the same uname, the platform name is created adding the major version number of the compiler added as a suffix: "g3" for gcc 3 and "g4" for gcc 4.

In Windows the platform name is the **compiler** and its **major** version number.

All library packages (***_lib***) contains pre-compiled binaries for the specified platform and includes. Packages with "**_bin**" suffix contains executables only.

The package name is a general reference for the platform. If you have the same platform it will work fine, but it may also work in similar platforms.

Here are some examples of packages:

iup2_4_Linux26_lib.tar.gz = IUP 2.4 32-bits Libraries and Includes for Linux with Kernel version 2.6 built with gcc 3.

iup2_4_Linux26g4_64_bin.tar.gz = IUP 2.4 64-bits Executables for Linux with Kernel version 2.6 built with gcc 4.

iup2_4_Win32_vc8_lib.tar.gz = IUP 2.4 32-bits Static Libraries and Includes for Windows to use with Visual C++ 8 (2005).

iup2_4_Win32_dll9_lib.tar.gz = IUP 2.4 32-bits Dynamic Libraries (DLLs), stub libraries and Includes for Windows to use with Visual C++ 9 (2008).

iup2_4_Docs_html.tar.gz = IUP 2.4 documentation files in HTML format (the web site files can be browsed locally).

iup2_4_Win32_bin.tar.gz = IUP 2.4 32-bits Executables for Windows.

The documentation files are in HTML format. They do not include the CHM and PDF versions. These two files are provided as a separate download, but they all have the same documentation.

Installation

For any platform we recommend you to create a folder to contain the third party libraries you download. Then just unpack the packages you download in that folder. The packages already contains a directory structure that separates each library or toolkit. For example:

```
\mylibs\
  iup\
    bin\
    html\
    include\
    lib\Linux26
    lib\Linux26g4_64
    lib\vc8
    src
  cd\
  im\
  lua5.1\
```

This structure will also made the process of building from sources more simple, since the projects and makefiles will assume this structure .

Usage

For makefiles use:

- 1) "-I/mylibs/iup/include" to find include files
- 2) "-L/mylibs/iup/lib/Linux26" to find library files
- 3) "-liup" to specify the library files

For IDEs the configuration involves the same 3 steps above, but each IDE has a different dialog. The IUP toolkit has a Guide for some IDEs:

Borland C++ BuilderX - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/cppbx.html

Code Blocks - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/codeblocks.html

Dev-C++ - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/dev-cpp.html

Eclipse for C++ - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/eclipse.html

Microsoft Visual C++ (Visual Studio 2003) - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/msvc.html

Microsoft Visual C++ (Visual Studio 2005) - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/msvc8.html

Open Watcom - http://www.tecgraf.puc-rio.br/iup/en/ide_guide/owc.html

Available Platforms

The following platforms can be available:

AIX43	IBM AIX 4.3 (ppc) / gcc 2.95 / Motif 2.1
IRIX65	SGI IRIX 6.5 (mips) / gcc 3.0 / Motif 2.1
IRIX6465	SGI IRIX 6.5 (mips) / gcc 3.3 / Motif 1.2
Linux24	Red Hat 7.3 (x86) / Kernel 2.4 / gcc 2.95 / Open Motif 2.1 / GTK 2.0
Linux24g3	CentOS 3.9 (x86) / Kernel 2.4 / gcc 3.2 / Open Motif 2.2 ³ / GTK 2.2
Linux24g3_64	Red Hat E.L. WS 3 (x64) / Kernel 2.4 / gcc 3.2 / Open Motif 2.2 ³ / GTK 2.2
Linux26	CentOS 4.6 (x86) / Kernel 2.6 / gcc 3.4 / Open Motif 2.2 ³ / GTK 2.4
Linux26_64	CentOS 4.6 (x64) / Kernel 2.6 / gcc 3.4 / Open Motif 2.2 ³ / GTK 2.4
Linux26g4	CentOS 5.3 (x86) / Kernel 2.6 / gcc 4.1 / Open Motif 2.2 ³ / GTK 2.12
Linux26g4_64	CentOS 5.2 (x64) / Kernel 2.6 / gcc 4.1 / OpenMotif 2.2 ³ / GTK 2.10
Linux26g4ppc	Ubuntu 7.10 (ppc) / Kernel 2.6 / gcc 4.1 / Open Motif 2.2 ³ / GTK 2.?
Linux26_ia64	Red Hat E.L. AS 4 (ia64) / Kernel 2.6 / gcc 3.4 / Open Motif 2.2 ³ / GTK 2.4
SunOS57	Sun Solaris 7 (sparc) / gcc 2.95 / Motif 2.1
SunOS58	Sun Solaris 8 (sparc) / gcc 3.4 / Motif 2.1
SunOS510x86	Sun Solaris 10 (x86) / gcc 3.3 / Motif 2.1
FreeBSD54	Free BSD 5.4 (x86) / gcc 3.4 / Open Motif 2.2 ³
Darwin811	Mac OS X 10.4.11 (ppc) [Tiger] / Darwin Kernel 8.11 / gcc 4.0 / Open Motif 2.1
Darwin811x86	Mac OS X 10.4.11 (x86) [Tiger] / Darwin Kernel 8.11 / gcc 4.0 / Open Motif 2.1
Darwin94x86	Mac OS X 10.5.4 (x86) [Leopard] / Darwin Kernel 9.4 / gcc 4.0 / Open Motif 2.1
Win32_vc6	Microsoft Visual C++ 6 (static RTL/single thread)

Win32_vc7	Microsoft Visual C++ 7.1 (.NET 2003) (static RTL/single thread) Also compatible with Microsoft Visual C++ Toolkit 2003 - http://msdn.microsoft.com/visualc/vctoolkit2003/ ¹
Win32_vc8	Microsoft Visual C++ 8.0 (2005) (static RTL/multithread) Also compatible with Microsoft Visual C++ 2005 Express Edition - http://msdn.microsoft.com/vstudio/express/visualc/ ¹
Win32_vc9	Microsoft Visual C++ 9.0 (2008) (static RTL/multithread) Also compatible with Microsoft Visual C++ 2008 Express Edition - http://msdn.microsoft.com/vstudio/express/visualc/ ¹
Win32_dll6	built using vc6, creates dependency with MSVCRT.DLL (either other libraries or new applications).
Win32_dll7	built using vc7, creates dependency with MSVCR71.DLL (either other libraries or new applications).
Win32_dll8	built using vc8, creates dependency with MSVCR80.DLL (either other libraries or new applications).
Win32_dll9	built using vc9, creates dependency with MSVCR90.DLL (either other libraries or new applications).
Win64_vc8	Same as Win32_vc8 but for 64-bits systems using x64 standard.
Win64_vc9	Same as Win32_vc9 but for 64-bits systems using x64 standard.
Win64_dll8	Same as Win32_dll8 but for 64-bits systems using x64 standard.
Win64_dll9	Same as Win32_dll9 but for 64-bits systems using x64 standard.
Win32_gcc3	Cygwin gcc 3.4 (Depends on Cygwin DLL 1.5) - http://www.cygwin.com/ ¹
Win32_cygw15	Same as Win32_gcc3 , but using the Cygwin Posix system
Win32_mingw3	MingW gcc 3.4 - http://www.mingw.org/ ¹ Also compatible with Dev-C++ - http://www.bloodshed.net/devcpp.html and with Code Blocks - http://www.codeblocks.org/ ¹
Win32_mingw4	MingW gcc 4.4 - http://www.mingw.org/ ¹
Win32_owc1	Open Watcom 1.5 - http://www.openwatcom.org/
Win32_bc56	Borland C++ BuilderX 1.0 / Borland C++ 5.6 Compiler - http://www.borland.com/products/downloads/download_cbuilderx.html ^{1,2} (the C++ BuilderX IDE can also be configured to use mingw3 or gcc3 versions.)
Win32_bin	Executables only for Windows NT/2000/XP
Win64_bin	Same as Win32_bin but for 64-bits systems using the x64 standard
Win32_cygw15_bin	Executables only for Windows NT/2000/XP, but using the Cygwin Posix system

¹ - Notice that all the Windows compilers with links here are free to download and use.

² - Recently Borland removed the C++ Builder X from download. But if you bought a book that has the CD of the compiler, then it is still free to use.

³ - OpenMotif 2.2 is classified as 'experimental' by the Open Group.

CVS

The CVS repository is at **SourceForge**. It can also be interactively browsed at:

<http://imtoolkit.cvs.sourceforge.net/imtoolkit/>

To checkout use the module name "im" and the CVSROOT:

```
:pserver:anonymous@imtoolkit.cvs.sourceforge.net:/cvsroot/imtoolkit
```

History of Changes

[Version 3.5](#) (02/Oct/2009)

- **New:** functions **imProcessUnsharp** and **imProcessSharp**.
- **New:** function **imImageGetOpenGLData**.
- **New:** functions **im.ConvertDataTypeNew**, **im.ConvertColorSpaceNew** and **im.ConvertToBitmapNew** in Lua.
- **New:** file attributes "FileFormat", "FileCompression" and "FileImageCount" when reading from file. Available for all formats.
- **New:** ASCII compression for RAW format to access text data instead of binary.
- **Changed:** libPNG updated to version 1.2.39. Removed changes to the library that made it incompatible with other libPNG distributions.
- **Changed:** libLZF updated to version 3.5.
- **Changed:** libJPEG updated to version 7.
- **Changed:** libEXIF updated to version 0.6.17.
- **Changed:** libTIFF updated to version 3.9.1.
- **Changed:** library **im_fftw3** to use an external library.
- **Changed:** **imImageCreateBased** and **imConvertColorSpace** now also consider the alpha plane.
- **Changed:** **imProcessPrune** renamed to **imProcessRemoveByArea**, and added a new parameter to select inside or outside the

interval.

- **Changed:** removed IM_UN_INC operation from **imProcessUnArithmeticOp**. It was not an unary operation. Can simply be done in place by **imProcessArithmeticOp** and IM_BIN_ADD.
- **Changed:** now **imProcessUnArithmeticOp**, **imProcessArithmeticConstOp** and **imProcessArithmeticOp** will crop the result to 0-255 if destiny has data type byte.
- **Fixed:** PNG attribute TransparencyIndex. new PNG attribute TransparencyMap. TransparentColor renamed to TransparencyColor.
- **Fixed:** invalid conversion from MAP to GRAY when loading **imImages**.
- **Fixed:** new image size computation of **im.ProcessCropNew** in Lua.
- **Fixed:** loading of RAW data.
- **Fixed:** **imImageClear** to initialize data just like **imImageCreate** does.
- **Fixed:** **imImageReshape** when the image has an alpha plane. Image is not cleared anymore.
- **Fixed:** boolean parameters in **file:ReadImageData**, **im.ConvertDataType**, **im.ConvertToBitmap**, **im.ProcessSplitComplex**, **im.ProcessQuantizeRGBUniform**, **im.ProcessBitPlane**, **im.ProcessRotateRef**, **im.ProcessRotate90**, **im.ProcessBinMorphConvolve**, **im.ProcessMergeComplex**, **im.CalcHistogram**, **im.CalcGrayHistogram** and **im.AnalyzeFindRegions** in Lua. Changed **im.Capture*** functions to use boolean values in Lua.
- **Fixed:** RAW format initialization.

Version 3.4.2 (26/Jun/2009)

- **Changed:** removed "lua5.1.so" dependency in UNIX.
- **Fixed:** AVI format when reading 32 and 16 bpp frames.
- **Fixed:** xmin and ymin check in **im.ProcessCrop** and in **im.ProcessInsert** in Lua.

Version 3.4.1 (15/Dec/2008)

- **Changed:** function **imColorHSI_Smax** removed from public, now it is used only internally. HSI space now uses S already normalized between 0-Smax.
- **Fixed:** **imColorHSI2RGB** conversion.
- **Fixed:** **imConvertDataType** when converting a floating point to integer, there were rounding problems.
- **Fixed:** loading and saving two or more files of the same format at the same time.

Version 3.4 (14/Oct/2008)

- **New:** **imlua_avi**, **imlua_wmv** and **imlua_jp2** libraries so the respective formats can be dynamically loaded using **require**.
- **Changed:** **IMPORTANT** - the "imlua_cd" library moved from IM to CD under the name "cdluaaim".
- **Changed:** **IMPORTANT** - the support services (Downloads, Mailing List and CVS) moved from LuaForge to SourceForge.
- **Changed:** All dll8 and dll9 DLLs now have a Manifest file that specifies the correct MSVCR*.DLL.
- **Changed:** Makefiles for UNIX now uses a compact version of Tecmake that does not need any installation, just type "make".
- **Changed:** premake files are used now only internally and were removed from the distribution.
- **Changed:** Copyright notice modified to reflect the registration at INPI (National Institute of Intellectual Property in Brazil). License continues under the same terms.
- **Fixed:** reviewed and fixed the parameter checking of all IMLua processing functions. Also reviewed all IMLua parameter checking. Thanks to Lucas Lorensi.
- **Fixed:** loading of TIFF format with old JPEG compression.
- **Fixed:** loading and saving of PNM format when data in textual format and gray values are greater than 255.
- **Fixed:** Bicubic and Zero order interpolation for all geometric operations for pixels near the image border when increasing image size.
- **Fixed:** Lua samples.
- **Fixed:** ICON format in 64 bits Linux.

Version 3.3 (26/Nov/2007)

- **New:** read support for ECW using the ERMMapper ECW JPEG 2000 SDK.
- **Changed:** libTIFF updated to version 3.8.2.
- **Changed:** libPNG updated to version 1.2.22.
- **Changed:** libJasper updated to libGeoJasper 1.4.0 (using Jasper version 1.900.1). Better support for counter progress, Geo tags and several speed improvements. New GeoTIFFBox and XMLPacket attributes.
- **Changed:** renamed macro **imPutImage** to **imcdCanvasPutImage**, and added canvas as the first parameter.
- **Changed:** renamed the **imImage** Lua methods to **image:cdCanvasPutImageRect**, **image:wdCanvasPutImageRect** and **image:cdCanvasGetImage**, and added canvas as the first parameter. Now **imlua_cd** depends on **cdlua** from CD version 5.0.
- **Changed:** metatable names in Lua are now the same as the C struct names.
- **Changed:** new read EXIF tags support in TIFF format (no write support yet). Renamed attributes "GeoTransMatrix" and "IntergraphMatrix", to "GeoTransformationMatrix" and "Intergraph TransformationMatrix" for libGeoTIFF compatibility. Better support for known TIFF tags. New support for reading one band of a multiband gray image in TIFF format. New support for DNG files.
- **Fixed:** **imConvertDataType** gamma function when converting real to/from integer.
- **Fixed:** small error at the image border when resampling, rotating or other geometric operations.
- **Fixed:** **imProcessCanny** invalid division by zero when input image is all zero.
- **Fixed:** **imFileReadImageInfo** when loading MAP images with a scrambled gray palette. They were incorrectly converted to GREY.
- **Fixed:** support for IM_ALPHA and 32 bpp in ICO format.
- **Fixed:** number of lines returned in **imProcessHoughLinesDraw**.

Version 3.2 (24/Nov/2006)

- **New:** **imProcessRotateRef** to rotate relative to a reference point.
- **New:** geometric distortion **imProcessSwirl**.
- **New:** **imProcessInterlaceSplit**.
- **New:** function **imGaussianKernelSize2StdDev**.

- **New:** convolutions **imProcessBarlettConvolve**, **imProcessPrewittConvolve**, **imProcessSplineEdgeConvolve**, **imProcessConvolveDual** and **imProcessConvolveSep**.
- **New:** "im_kernel.h" module with simple functions to create know pre-defined kernels like sobel, laplacian, gaussian, etc.
- **New:** **imVideoCaptureSetInOut** to control input and output in capture devices.
- **New:** function **imBinMemoryRelease** to release internal memory allocated by the BinMemory file when saving.
- **New:** functions for capture device information: **imVideoCaptureDeviceExDesc**, **imVideoCaptureDevicePath** and **imVideoCaptureDeviceVendorInfo**.
- **New:** function **imFileOpenAs** to open a file of a specific format.
- **New:** functions **imFormatRegisterInternal** and **imFormatRemoveAll** to control format registration.
- **Changed:** **imProcessGaussianConvolve** to used separable convolution and now is stddev is negative will use its magnitude as the kernel size. Removed Rep functions **imProcessGaussianConvolveRep**, **imProcessDiffOfGaussianConvolveRep** and **imGaussianStdDev2Repetitions**.
- **Changed:** **imProcessBlend** to use an image instead of a constant. Old function renamed to **imProcessBlendConst**.
- **Changed:** **imFileHandle** prototype. Now the function has an index parameter to specify which handle it should return. index=0 is always an imBinFile* handle. Use index=1 or greater to return other internal handles that are format dependent.
- **Changed:** the Removed the include "im.h" to not include "im_lib.h". "im_lib.h" must be included when necessary.
- **Changed:** **imAnalyzeMeasureArea** and **imAnalyzeMeasurePerimeter** prototypes to include the number of regions as a parameter. Fixed: these functions to internally initialize the results array to zero (this was necessary and not documented).
- **Changed:** **imProcessFlip** and **imProcessMirror** so they can be done in-place.
- **Fixed:** missing implementation of **imVideoCaptureOneFrame** in Lua 5.
- **Fixed:** **imAnalyzeFindRegions** when pixel is at the width-1 column.
- **Fixed:** file format identification when **TIFF** identification failed was not closing the file.
- **Fixed:** **imAnalyzeMeasurePerimeter** when perimeter line is at the first or last lines. Thanks to Takeshi Mitsunaga.
- **Fixed:** invalid return value in **imVideoCaptureConnect** in Lua 5.
- **Fixed:** **imProcessRotate** for IM_MAP images.
- **Fixed:** **Lua** binding of **imFileImageSave**, wrong parameters order. New: image:Save(filename, format) alias for imImage objects.
- **Fixed:** **BMP** format implementation when reading and writing RGBA 32 bits images.
- **Fixed:** **imFileLoadImageFrame** and **imFileLoadBitmapFrame** index parameter in Lua.
- **Fixed:** alpha channel allocation in imImage.

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:** **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)

- **New:** Image Transform **imProcessDistanceTransform**.
- **New:** group of functions Image Analysis: **imAnalyzeFindRegions**, **imAnalyzeMeasureArea**, **imAnalyzeMeasurePerimArea**, **imAnalyzeMeasureCentroid**, **imAnalyzeMeasurePrincipalAxis**, **imAnalyzeMeasureHoles**, **imProcessPerimeterLine**, **imAnalyzeMeasurePerimeter**, **imProcessPrune**, **imProcessFillHoles**.
- **New:** **imConvertMapToRGB** to help loading data as RGB.
- **New:** sample iuglcap.
- **New:** **imProcessRenderChessboard** and **imProcessRenderGrid**.
- **Changed:** **imProcessThreshold**, **imProcessRangeContrastThreshold** and **imProcessLocalMaxThreshold** now also supports **IM_USHORT** and **IM_INT** data types.
- **Changed:** the default color conversion to binary so it can be done for all color spaces.
- **Changed:** **im_process.h** to 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.
- **Changed:** 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.
- **Fixed:** file read with bitmap conversion when original data changes only data type.
- **Fixed:** rank convolution operations that did not accept even kernel sizes.
- **Fixed:** **imProcessHoughLinesDraw** that was ignoring some lines.

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.
- **Changed:** logical operations, flag **IM_BIT_NOT** replaced by operation **imProcessBitwiseNot**.
- **Changed:** **imImageSetAttribute** count can be -1 for zero terminated data.
- **Fixed:** operations **imProcessBitwiseNot** and **imProcessNegative** for **IM_BINARY** images.
- **Fixed:** the **color_mode_flags** parameter interpretation by **imFileReadImageData**.
- **Fixed:** **imProcessEqualizeHistogram** and **imProcessExpandHistogram** for color images.
- **Fixed:** **imProcessMultipleStdDev**.
- **Fixed:** **imProcessDifusionErrThreshold** for **IM_GRAY** images.
- **Fixed:** "KRN" format, internal format is topdown.
- **Fixed:** 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

General

- **MOV** (using QuickTime SDK and QT4Linux)
- **DICOM**
- **TIFF Annotations**
- **Linux Capture** (using Video4Linux)
- **Use libavcodec and libavformat in Linux. AVI using libavifile in Linux (UNIX ?)**
- **MPEG-2** (using MSSG?)
- **VC-1 Coded** using Microsoft VC-1 Encoder SDK
- **In SunOS using the Sun WorkShop 6 C++, an error occurs when linking an application.**
- **In AIX we do not have the C++ for AIX installed, so the library is not available.**

For the Processing library:

- Support for OpenMP
- Support for the Intel[®] Integrated Performance Primitives
- Dithering Techniques
- Adaptative Thresholds
- Warping
- Rolling Ball Filter
- Butterworth, Deconvolution
- Inverse Filter, Homomorphic Restoration
- Watershed, Convex Hull
- Other Measures

Our plans for the future include:

- **Imaging Tutorial in the documentation**
- **JPEG and TIFF Thumbnails**
- **Formats: FLI, DV, FPX (Flash Pix), EXR (Industrial Light & Magic High Dynamic Range Format), MNG, Microsoft HD Photo**
- **ECW write**
- **OpenML?**
- **WIA and TWAIN?**

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 slower 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.

Comparison

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 chose 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.

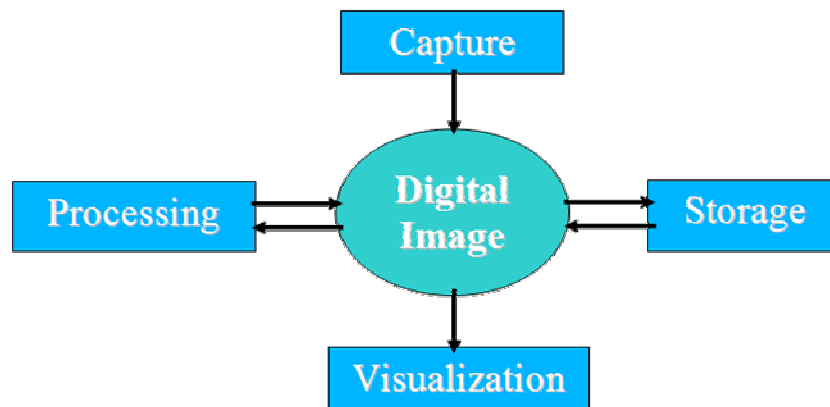
Although OpenCV has many resources, its code is very hard to reuse. The simplicity of the IM code, mainly the image processing routines, make it a good reference to be reused by other applications extracting only the code needed with little changes. And can be used as an complement to learn image processing 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 theses 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 - Canvas Draw](#).

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 works.

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 you 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

In the Downloads you will be able to find pre-compiled binaries for many platforms, all those binaries were built using Tecmake. Tecmake is a command line multi compiler build tool based on GNU make, available at <http://www.tecgraf.puc-rio.br/tecmake>. Tecmake is used by all the Tecgraf libraries and many applications.

In UNIX, you do not need to install Tecmake, a compact version of Tecmake for UNIX is already included in the source code package. Just type `"make"` in the command line on the `"src"` folder and all libraries and executables will be build. Set the `TECTOOLS_HOME` environment variable to the folder where the Lua libraries are installed, by default it will assume `"TECTOOLS_HOME=../.."`.

In Windows, the easiest way to build everything 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. Also there are files named `make_uname.bat` that build the libraries using **Tecmake**. To build for Windows using Visual C 7.0 (2005) for example, just execute `"make_uname vc7"`, or the DLLs with Visual C++ 9 (2008) type `"make_uname dll9"`. The Visual Studio workspaces with the respective projects available in the source package is for debugging purposes only.

Make sure you have all the dependencies for the library you want installed, see the documentation below.

If you are going to build all the libraries, the makefiles and projects expect the following directory tree:

```
\mylibs\
  im\
  lua5.1\
```

Libraries Dependencies

```
im -> libjpeg (included)
    -> libpng (included)
    -> libtiff (included)
    -> zlib (included)
    -> liblzf (included)
    -> libexif (included)
im_jp2 -> im
        -> libJasper (included)
im_avi -> im
        -> vfw32 (system - Windows)
im_wmv -> im
        -> wmvcore (system - Windows)
im_ecw -> im
        -> NCSEcw (system)
im_capture -> strmiids (system - Windows)
im_process -> im
im_fftw -> im
        -> fftw (included)
imlua51 -> im
        -> lua5.1
imlua_capture51 -> imlua51
                -> im_capture
imlua_fftw51 -> imlua51
                -> im_fftw
imlua_process51 -> imlua51
                -> im_process
```

As a general rule (excluding system dependencies and included third party libraries): IM has NO external dependencies, and IMLua depends on Lua.

The Lua binding for IM (Makfiles and Pre-compiled binaries) depend on the [LuaBinaries](#) distribution. So if you are going to build all use the **LuaBinaries** source package also, not the **Lua.org** original source package. If you like to use another location for the Lua files overwrite the `LUAINC`, `LUALIB` and `LUABINDIR` definitions before using Tecmake or Tecmake Compact.

CD Compatibility

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

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

Function **cdCanvasGetImageRGB** captures an image from the active canvas. Functions **cdCanvasPutImageRect*** draw a client image on the active canvas. These functions allow reducing or increasing the image when drawing.

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

When using the **imImage** structure the macro **imcdCanvasPutImage** can be used. It is defined as:

```
#define imcdCanvasPutImage(_canvas, _image, _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax) \
{ \
    if (_image->color_space == IM_RGB) \
    { \
        if (image->has_alpha) \
            cdCanvasPutImageRectRGBA(_canvas, _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 \
            cdCanvasPutImageRectRGB(_canvas, _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 \
        cdCanvasPutImageRectMap(_canvas, _image->width, _image->height, \
            (unsigned char*)_image->data[0], _image->palette, \
            _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->height, image->depth, image->data_type, 0);
glPixelStorei(GL_UNPACK_ALIGNMENT, 1); /* data alignment must be 1 */

glDrawPixels(image->width, image->height, GL_RGB, GL_UNSIGNED_BYTE, (GLvoid*)gl_data);
```

When loading color image data you can use the function **imConvertMapToRGB** 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, packed = 1; /* OpenGL uses packed RGB */
    imFileGetPalette(ifile, palette, &palette_count);
    imConvertMapToRGB(gl_data, width*height, depth, packed, palette, palette_count);
}
```

If you are using the **imImage** structure then you can instead use the function **imImageGetOpenGLData**.

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, (GLvoid*)gl_data);

ifile = imFileNew(filename, format, &error);
error = imFileWriteImageInfo(ifile, width, height, IM_RGB | IM_PACKED, IM_BYTE);
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 driver 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 inevitably used (since we do not like long macros everywhere). But they are used only for processing functions, not classes.

Complete Samples

You can also browse the [examples folder](#).

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. It is very usefull for Computer Vision courses..

You can download the source code and projects for Visual C++, Borland C++ Builder X and Dev-Cpp, here: [iupglcap.zip](#) You will need to download IUP, CD and IM libraries for the compiler you use

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:

```

require"imlua"
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:ReadImageInfo(i)
end
ifile:Close()

```

To edit pixels in an image and save the changes:

```

require"imlua"

local image = im.FileImageLoad(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
        g[row][column] = 0
        b[row][column] = 0
    end
end

image:Save("edit.bmp", "BMP")

```

To render noise:

```

require"imlua"
require"imlua_process"
local image = im.ImageCreate(500, 500, im.RGB, im.BYTE)
im.ProcessRenderRandomNoise(image)
image:Save("noise.tif", "TIFF")

```

To render using the CD library:

```

require"imlua"
require"cdlua"
require"cdluaimg"

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

canvas:Activate()
canvas:Background(cd.EncodeColor(255, 255, 255))
canvas:Clear()
fgcolor = cd.EncodeColor(255, 0, 0) -- red
fgcolor = cd.EncodeAlpha(fgcolor, 50) -- semi transparent
canvas:Foreground(fgcolor)
canvas:Font("Times", cd.BOLD, 24)
canvas:Text(100, 100, "Test")
canvas:Line(0,0,100,100)
canvas:Kill()

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

```

Check the files [samples_imlua5.tar.gz](#) or [samples_imlua5.zip](#) for several samples in Lua. For some of them you will need also the CD and the IUP libraries. You can also browse the [examples folder](#).

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 Lua 5

#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);
}

```

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 5 "loadlib" function. Here is an example on how to dynamically load IM in Lua 5.1:

```
local imlua_open = package.loadlib("imlua51.dll", "imlua_open")
imlua_open()
```

Lua 5.1 "require" can be used for all the **ImLua** libraries. You can use **require"imlua"** and so on, but the LUA_CPATH must also contains the following:

```
"/lib?51.so;"      [in UNIX]
"\\?51.dll;"        [in Windows]
```

The [LuaBinaries](#) distribution already includes these modifications on the default search path.

The simplest form **require"im"** and so on, can not be used because there are IM dynamic libraries with names that will conflict with the names used by **require** during search.

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(...
```

Modifications to the API

New functions (without equivalents in C) were implemented to create and destroy objects that do not exist in C. For instance functions were developed to create and destroy images, files, video capture and palette. All the metatables have the "tostring" method implemented to help debugging.

Some functions were modified to receive those objects as parameters.

Also the functions which receive values by reference in C were modified. Generally, the values of parameters that would have their values modified are now returned by the function in the same order.

In C there is no parameter checking in the processing functions, but in Lua all parameters are checked and a Lua error is emitted when check fails.

Garbage Collection

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

Integration with CDLua

In **CDLua** 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 also the [CD documentation](#) and the [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 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 (waist 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:

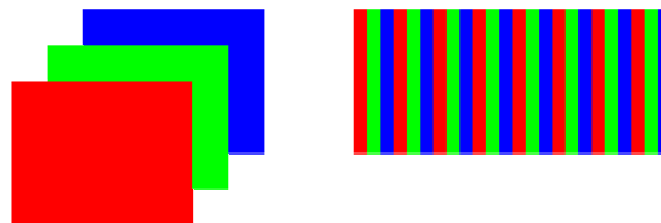
- **IM_ALPHA**
- **IM_PACKED**
- **IM_TOPDOWN**

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 rbgbrgb...) 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

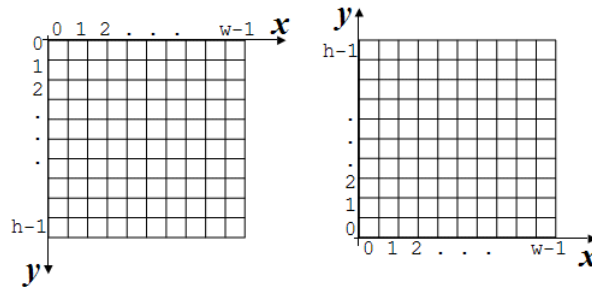
Separated and Packed RGB Components

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 components
 0 1 2 3 0 1 2 3 0 1 2 3

RGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGBRGB - for packed components
 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**.

As the library is designed to work with such a wide range of image data organization, there are no general purpose functions for getting/setting individual pixels, as they would be too complicated and inefficient. Rather, you should use the components of the imImage structure to access image pixels in the most efficient way.

Bitmaps

An important subset of images is what we call a **Bitmap** image. It is an image that can be displayed in graphics devices usually using a graphics library like CD or OpenGL. 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**.

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, palette, palette_count)
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.

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

Image Representation Samples

See the [Representation Guide](#) for simple image representation samples.

Information

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](#)

View Using IUP and CD

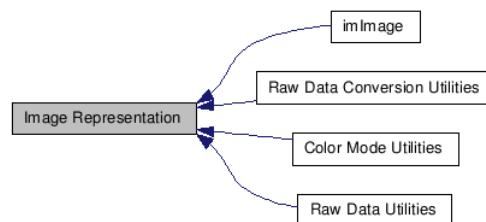
This application uses IUP and CD to create a window with a canvas and draw the image into that canvas. It is a very simple application, no zoom nor scrollbar management. The image is obtained from a file using the IM I/O functions, but using the **imImage** structure to make the implementation easier.

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

Collaboration diagram for Image Representation:



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 }

```

Detailed Description

See [im.h](#)

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 imaginary parts.

```
00019 {
00020     IM\_BYTE,    /**< "unsigned char". 1 byte from 0 to 255.          */
00021     IM\_USHORT,  /**< "unsigned short". 2 bytes from 0 to 65,535.        */
00022     IM\_INT,     /**< "int". 4 bytes from -2,147,483,648 to 2,147,483,647.  */
00023     IM\_FLOAT,  /**< "float". 4 bytes single precision IEEE floating point. */
00024     IM\_CFLOAT  /**< complex "float". 2 float values in sequence, real and imaginary parts. */
00025 };
```

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 intensity value that is not related to color.
IM_BINARY Indexed by 2 colors: black (0) and white (1) (data_type=IM_BYTE).
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, nearly perceptually uniform).
IM_LUV CIE L*u*v*. L* is Lightness (nonlinear Luminance, nearly perceptually uniform).
IM_XYZ CIE XYZ. Linear Light Tristimulus, Y is linear Luminance.

```
00031 {
00032     IM\_RGB,    /**< Red, Green and Blue (nonlinear).          */
00033     IM\_MAP,    /**< Indexed by RGB color map (data_type=IM_BYTE). */
00034     IM\_GRAY,   /**< Shades of gray, luma (nonlinear Luminance), or an intensity value that is not related
00035     IM\_BINARY, /**< Indexed by 2 colors: black (0) and white (1) (data_type=IM_BYTE). */
00036     IM\_CMYK,   /**< Cyan, Magenta, Yellow and Black (nonlinear). */
00037     IM\_YCBCR,  /**< ITU-R 601 Y&apos;CbCr. Y&apos; is luma (nonlinear Luminance). */
00038     IM\_LAB,    /**< CIE L*a*b*. L* is Lightness (nonlinear Luminance, nearly perceptually uniform). */
00039     IM\_LUV,    /**< CIE L*u*v*. L* is Lightness (nonlinear Luminance, nearly perceptually uniform). */
00040     IM\_XYZ,    /**< CIE XYZ. Linear Light Tristimulus, Y is linear Luminance. */
00041 };
```

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

```
00047 {
00048     IM\_ALPHA   = 0x100, /**< adds an Alpha channel */
00049     IM\_PACKED  = 0x200, /**< packed components (rgbrgrgb...) */
00050     IM\_TOPDOWN = 0x400  /**< orientation from top down to bottom */
00051 };
```

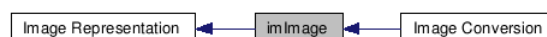


Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

imImage

[[Image Representation](#)]

Collaboration diagram for imImage:



Data Structures

struct [imImage](#)
 imImage Structure Definition. [More...](#)

Modules

[Image Conversion](#)

Defines

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

Typedefs

```
typedef struct
    imImage imImage
```

Functions

```
imImage * imImageCreate (int width, int height, int color_space, int data_type)
imImage * imImageInit (int width, int height, int color_space, int data_type, void *data_buffer, long *palette, int
    palette_count)
imImage * imImageCreateBased (const imImage *image, int width, int height, int color_space, int data_type)
void imImageDestroy (imImage *image)
void imImageAddAlpha (imImage *image)
void imImageReshape (imImage *image, int width, int height)
void imImageCopy (const imImage *src_image, imImage *dst_image)
void imImageCopyData (const imImage *src_image, imImage *dst_image)
imImage * imImageDuplicate (const imImage *image)
imImage * imImageClone (const imImage *image)
void imImageSetAttribute (const imImage *image, const char *attrib, int data_type, int count, const void *data)
const void * imImageGetAttribute (const imImage *image, const char *attrib, int *data_type, int *count)
void imImageGetAttributeList (const imImage *image, char **attrib, int *attrib_count)
void imImageClear (imImage *image)
    int imImageIsBitmap (const imImage *image)
void imImageSetPalette (imImage *image, long *palette, int palette_count)
void imImageCopyAttributes (const imImage *src_image, imImage *dst_image)
    int imImageMatchSize (const imImage *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)
void imImageMakeGray (imImage *image)
```

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](#)

Define Documentation

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

Value:

```
{
    if (_image->color_space == IM\_RGB)
    {
        if (_image->has_alpha)
            cdCanvasPutImageRectRGBA(_canvas, _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
            cdCanvasPutImageRectRGB(_canvas, _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
        cdCanvasPutImageRectMap(_canvas, _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_type IM_BYTE, and color spaces: IM_RGB, IM_MAP, IMGRAY and IM_BINARY.

Typedef Documentation

typedef struct [imImage](#) [imImage](#)

imImage 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

```

imImage* imImageCreate ( int  width,
                          int  height,
                          int  color_space,
                          int  data_type
                        )

```

Creates a new image. See also [imDataType](#) and [imColorSpace](#). Image data is cleared as [imImageClear](#). In Lua the IM image metatable name is "imImage". When converted to a string will return "imImage(%p) [width=%d,height=%d,color_space=%s,data_type=%s,depth=%d]" where p is replaced by the userdata address, and other values are replaced by the respective attributes. If the image is already destroyed by im.ImageDestroy, then it will return also the suffix "-destroyed".

```
im.ImageCreate(width: number, height: number, color_space: number, data_type: number) -> image: imImage [in L
```

```

imImage* imImageInit ( int    width,
                       int    height,
                       int    color_space,
                       int    data_type,
                       void * data_buffer,
                       long * palette,
                       int    palette_count
                     )

```

Initializes the image structure but does not allocates image data. See also [imDataType](#) and [imColorSpace](#).

```

imImage* imImageCreateBased ( const imImage * image,
                              int          width,
                              int          height,
                              int          color_space,
                              int          data_type
                            )

```

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. HasAlpha is copied. See also [imDataType](#) and [imColorSpace](#).

```
im.ImageCreateBased(image: imImage, [width: number], [height: number], [color_space: number], [data_type: num
```

The additional parameters in Lua can be nil, and they can also be functions with the based image as a parameter to return the respective value.

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

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

In Lua if this function is not called, the image is destroyed by the garbage collector.

```
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 * src_image,
                       imImage *      dst_image
                       )
```

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 ( const imImage * image,
                           const char *   attrib,
                           int            data_type,
                           int            count,
                           const void *   data
                           )
```

Changes an extended attribute.

The data will be internally duplicated.

If data is NULL and count==0 the attribute is removed.

If count is -1 and data_type is IM_BYTE then data is zero terminated. See also [imDataType](#).

```
image:SetAttribute(attrib: string, data_type: number, data: table of numbers or string) [in Lua 5]
```

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

```
const void* imImageGetAttribute ( const imImage * image,
                                  const char *   attrib,
                                  int *          data_type,
                                  int *          count
                                  )
```

Returns an extended attribute.

Returns NULL if not found. See also [imDataType](#).

```
image:GetAttribute(attrib: string, [as_string: boolean]) -> data: table of numbers or string, data_type: numb
```

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

```
void imImageGetAttributeList ( const imImage * image,
                               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.

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

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

Sets all image data to zero. But if color space is YCBCR, LAB or LUV, and data type is BYTE or USHORT, then data is initialized with 128 or 32768 accordingly. Alpha is initialized as transparent (0).

```
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 pointer. Only the pointer is stored, so the palette should be a new palette and it can not be a static array.

```
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 color 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 BYTE data (0,255) into a binary data (0,1), done in-place. Color space is not changed.

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

```
void imImageMakeGray ( imImage * image )
```

Changes a binary data (0,1) into a gray BYTE data (0,255), done in-place. Color space is not changed.

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



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

Image Conversion

[\[imImage\]](#)

Collaboration diagram for Image Conversion:



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
* imImageGetOpenGLData (const imImage *image, int *glformat)
```

Detailed Description

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

See [im_convert.h](#)

Enumeration Type Documentation

```
enum imComplex2Real
```

Complex to real conversions

```
00029 {
00030     IM_CPX_REAL,
00031     IM_CPX_IMAG,
00032     IM_CPX_MAG,
00033     IM_CPX_PHASE
00034 };
```

```
enum imGammaFactor
```

Predefined Gamma factors. Gamma can be any real number. When gamma<0 use logarithmic, when gamma>0 use exponential. gamma

$$(x,g) = ((e^{(g*x)}-1)/(exp(g)-1) \text{ gamma}(x,g) = (\log((g*x)+1))/(\log(g+1))$$

```
00042 {
00043     IM_GAMMA_LINEAR    = 0,
00044     IM_GAMMA_LOGLITE   = -10,
00045     IM_GAMMA_LOGHEAVY  = -1000,
00046     IM_GAMMA_EXPLITE   = 2,
00047     IM_GAMMA_EXPHEAVY  = 7
00048 };
```

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 Manipulation](#) Color Manipulation.
IM_CAST_DIRECT direct type cast the value. Only byte and ushort will be cropped.

```
00053 {
00054     IM\_CAST\_MINMAX, /**< scan for min and max values */
00055     IM\_CAST\_FIXED, /**< use predefined 0-max values, see \ref color Color Manipulation. */
00056     IM\_CAST\_DIRECT /**< direct type cast the value. Only byte and ushort will be cropped. */
00057 };
```

Function Documentation

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

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

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

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

Images must be of the same size and color mode.

Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER, see also [imErrorCodes](#). See also [imComplex2Real](#), [imGammaFactor](#) and [imCastMode](#).

```
im.ConvertDataType(src_image: imImage, dst_image: imImage, cpx2real: number, gamma: number, absolute: boolean)
```

```
im.ConvertDataTypeNew(image: imImage, data_type: number, cpx2real: number, gamma: number, absolute: boolean,
```

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

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

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

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

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

Alpha channel is considered and Transparency* attributes are converted to alpha channel.

All other color space conversions assume sRGB and CIE definitions.

Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER, see also [imErrorCodes](#).

```
im.ConvertColorSpace(src_image: imImage, dst_image: imImage) -> error: number [in Lua 5]
```

```
im.ConvertColorSpaceNew(image: imImage, color_space: number, has_alpha: boolean) -> error: number, new_image:
```

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

Converts the image to its bitmap equivalent, uses [imConvertColorSpace](#) and [imConvertDataType](#).

Returns IM_ERR_NONE, IM_ERR_DATA or IM_ERR_COUNTER, see also [imErrorCodes](#). See also [imComplex2Real](#), [imGammaFactor](#) and [imCastMode](#).

The function `im.ConvertToBitmapNew` uses the default conversion result from [imColorModeToBitmap](#) if `color_space` is nil.

```
im.ConvertToBitmap(src_image: imImage, dst_image: imImage, cpx2real: number, gamma: number, absolute: boolean)
im.ConvertToBitmapNew(image: imImage, color_space: number, has_alpha: boolean, cpx2real: number, gamma: number)

void* imImageGetOpenGLData ( const imImage * image,
                             int * glformat
                           )
```

Returns an OpenGL compatible data buffer. Also returns the correspondant pixel format.

The memory allocated is stored in the attribute "GLDATA" with BYTE type. And it will exists while the image exists.

It can be cleared by setting the attribute to NULL.

MAP images are converted to RGB, and BINARY images are converted to GRAY. Alpha channel is considered and Transparency* attributes are converted to alpha channel. So calculate depth from `glformat`, not from image depth.

```
image:GetOpenGLData() -> gldata: userdata, glformat: number [in Lua 5]
```



Generated on Thu Oct 1 11:40:04 2009 for IM by 1.6.1

Raw Data Utilities

[\[Image Representation\]](#)

Collaboration diagram for Raw Data Utilities:



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)

Detailed Description

See [im_util.h](#)

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, data_type: number) -> datasize: number [in Lua 5]
```

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

Returns the size of one line of the data buffer.

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

```
im.ImageLineSize(width: number, color_mode: number, data_type: number) -> linesize: number [in Lua 5]
```

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

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

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

```
im.ImageLineCount(width: number, color_mode: number) -> linecount: number [in Lua 5]
```

```
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: boolean [in Lua 5]
```



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

Raw Data Conversion Utilities

[[Image Representation](#)]

Collaboration diagram for Raw Data Conversion Utilities:



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)

Detailed Description

Utilities for raw data buffers.

See [im_convert.h](#)

Function Documentation

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

Changes the packing of the data buffer. Both must have the same depth.

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

Changes in-place a MAP data into a RGB data. The data must have room for the RGB image.
depth can be 3 or 4. count=width*height.



Generated on Thu Oct 1 11:40:04 2009 for IM by 1.6.1

Color Mode Utilities

[[Image Representation](#)]

Collaboration diagram for Color Mode Utilities:



Defines

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

Functions

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

Detailed Description

See [im_util.h](#)

Define Documentation

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

Returns the color space of the color mode.

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

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

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

```
im.ColorModeMatch(color_mode1: number, color_mode2: number) -> match: boolean [in Lua 5]
```

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

Check if the color mode has an alpha channel.

```
im.ColorModeHasAlpha(color_mode: number) -> has_alpha: boolean [in Lua 5]
```

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

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

```
im.ColorModeIsPacked(color_mode: number) -> is_packed: boolean [in Lua 5]
```

```
#define imColorModeIsTopDown ( _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 Lua 5]
```

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_MAP or IM_BINARY.

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

```
int imColorModeIsBitmap (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]
```



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

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 start 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.

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_mode, &data_type);
    if (error != IM_ERR_NONE)
        // handle the error

    // prepare data
```

```

    error = imFileReadImageData(ifile, data, 0, -1); // no bitmap conversion, use original color mode flags
    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, &data_type);

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_BYTE);
    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, data_type);

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 = "Mem ria Insuficiente.";
                break;
            case IM_ERR_ACCESS:
                msg = "Erro Acessando Arquivo.";
                break;
            case IM_ERR_DATA:
                msg = "Tipo de Imagem n o Suportado.";
                break;
            case IM_ERR_FORMAT:
                msg = "Formato Inv lido.";
                break;
            case IM_ERR_COMPRESS:
                msg = "Compress o Inv lida ou n o 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.

More Storage Samples

See the [Storage Guide](#) for simple storage samples.

Information

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](#)

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](#)

Load Bitmap from Resource File

In Windows if you have a bitmap stored in a resource file, like this:

```
bitmap_test BITMAP bitmap_test.bmp
```

The you could retrieve it using the following code:

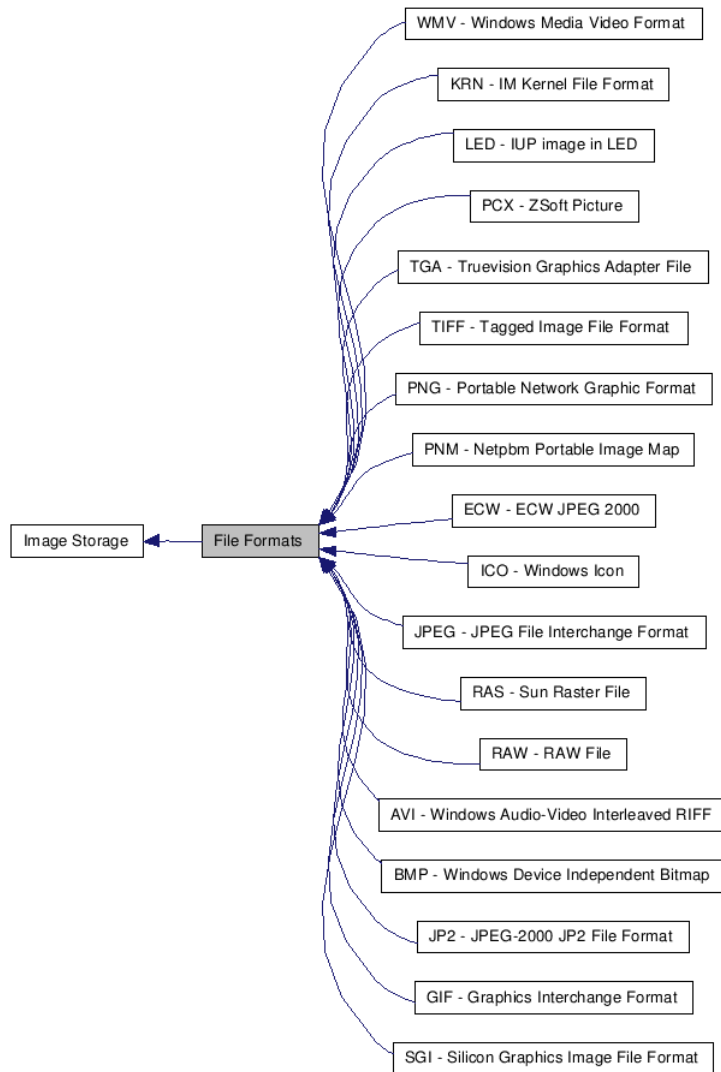
```
#include <windows.h>
#include <im.h>
#include <im_dib.h>

HBITMAP hBmp = LoadBitmap(hInstance, "bitmap_test");
imDib* dib = imDibFromHBitmap(hBmp, NULL);
imImage* image imDibToImage(dib);
imDibDestroy(dib);
```

File Formats

[\[Image Storage\]](#)

Collaboration diagram for File Formats:



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](#)
[ECW - ECW JPEG 2000](#)
[JP2 - JPEG-2000 JP2 File Format](#)
[RAW - RAW File](#)
[WMV - Windows Media Video Format](#)

Functions

void [imFormatRegisterInternal](#) (void)
 void [imFormatRemoveAll](#) (void)
 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)

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

Function Documentation

```
void imFormatRegisterInternal ( void )
```

Registers all the internal formats.

It is automatically called internally when a format is accessed, but can be called to force the internal formats to be registered before other formats. Notice that additional formats when registered will be registered before the internal formats if `imFormatRegisterInternal` is not called yet.

To control the register order is usefull when two format drivers handle the same format. The first registered format will always be used first.

```
void imFormatRemoveAll ( void )
```

Remove all registered formats.

```
void imFormatList ( char ** format_list,
                   int *   format_count
                   )
```

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 * format,
                  char *      desc,
                  char *      ext,
                  int *       can_sequence
                  )
```

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. See also [File Formats](#).

```
im.FormatInfo(format: string) -> error: number, desc: string, ext: string, can_sequence: boolean [in Lua 5]
```

```
int imFormatCompressions ( const char * format,
                          char **      comp,
                          int *        comp_count,
                          int          color_mode,
                          int          data_type
                          )
```

)

Returns the format compressions.

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

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

If you use them they will select only the allowed compressions checked like in [imFormatCanWriteImage](#).

Returns an error code. See also [File Formats](#), [imErrorCodes](#), [imDataType](#), [imColorSpace](#) and [imColorModeConfig](#).

```
im.FormatCompressions(format: string, [color_mode: number], [data_type: number]) -> error: number, comp: tabl
```

```
int imFormatCanWriteImage ( const char * format,
                           const char * compression,
                           int          color_mode,
                           int          data_type
                           )
```

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

Returns an error code. See also [File Formats](#), [imErrorCodes](#), [imDataType](#), [imColorSpace](#) and [imColorModeConfig](#).

```
im.FormatCanWriteImage(format: string, compression: string, color_mode: number, data_type: number) -> can_wri
```



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

RAW - RAW File

[\[File Formats\]](#)

Collaboration diagram for RAW - RAW File:



Functions

[imFile](#) * [imFileOpenRaw](#) (const char *file_name, int *error)

[imFile](#) * [imFileNewRaw](#) (const char *file_name, int *error)

Detailed Description

The file must be open/created with the functions [imFileOpenRaw](#) and [imFileNewRaw](#).

Description

Internal Implementation.

Supports RAW binary images. This is an unstructured and uncompressed binary data. It is NOT a Camera RAW file generated in many professional digital cameras.

You must know image parameters a priori and 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.

If the compression is ASCII the data is stored in textual format, instead of binary. In this case SwitchType and ByteOrder are ignored, and Padding should be 0.

See [im_raw.h](#)

Features

Data Types: <all>

Color Spaces: all, except MAP.

Compressions:

NONE - no compression [default]

ASCII (textual data)

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.

Attributes:

```
Width, Height, ColorMode, DataType IM_INT (1)
ImageCount[1], StartOffset[0], SwitchType[FALSE], ByteOrder[IM_LITTLEENDIAN], Padding[0] IM_INT (1)
```

Comments:

In fact ASCII is an expansion, not a compression, because the file will be larger than binary data.

Function Documentation

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

Opens a RAW image file. See also [imErrorCodes](#).

```
im.FileOpenRaw(file_name: string) -> ifile: imFile, error: number [in Lua 5]
```

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

Creates a RAW image file. See also [imErrorCodes](#).

```
im.FileNewRaw(file_name: string) -> ifile: imFile, error: number [in Lua 5]
```

Generated on Thu Oct 1 11:40:05 2009 for IM by

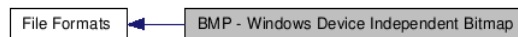


1.6.1

BMP - Windows Device Independent Bitmap

[\[File Formats\]](#)

Collaboration diagram for BMP - Windows Device Independent Bitmap:



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 Lenght 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 are saved always as bottom up.

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.

Generated on Thu Oct 1 11:40:05 2009 for IM by



1.6.1

GIF - Graphics Interchange Format

[\[File Formats\]](#)

Collaboration diagram for GIF - Graphics Interchange Format:



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.

Attributes:

ScreenHeight, ScreenWidth IM_USHORT (1) screen size [default to the first image size]

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) [time to wait between frames in 1/100 of a second]

Iterations IM_USHORT (1) (NETSCAPE2.0 Application Extension) [The number of times to repeat the animati

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 Extensions...

Generated on Thu Oct 1 11:40:05 2009 for IM by



1.6.1

ICO - Windows Icon

[File Formats]

Collaboration diagram for ICO - Windows Icon:



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.

Can have an alpha channel (only for RGB)

Internally the components are always packed.

Internally the lines are arranged from bottom up to top.

Attributes:

TransparencyIndex IM_BYTE (1)

Comments:

If the user specifies an alpha channel, the AND mask is loaded as alpha if

```

the file color mode does not contain the IM_ALPHA flag.
For MAP imagens, if the user does not specifies an alpha channel
the TransparencyIndex is used to initialize the AND mask when writing,
and if the user does specifies an alpha channel
the most repeated index with transparency will be the transparent index.
Although any size and common bpp can be used is recomended to use the typical configurations:
16x16, 32x32, 48x48, 64x64 or 96x96
2 colors, 16 colors, 256 colors, 24bpp or 32bpp

```

Generated on Thu Oct 1 11:40:05 2009 for IM by

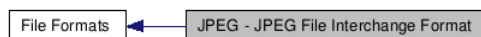


1.6.1

JPEG - JPEG File Interchange Format

[File Formats]

Collaboration diagram for JPEG - JPEG File Interchange Format:



Description

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

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

Access to the JPEG file format uses libJPEG version 7.

<http://www.ijg.org>

Copyright (C) 1994-2009, Thomas G. Lane, Guido Vollbeding
from the Independent JPEG Group.

Access to the EXIF attributes uses libEXIF version 0.6.17.

<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(1) returns jpeg_decompress_struct* when reading, and
jpeg_compress_struct* when writing (libJPEG structures).

Attributes:

AutoYCbCr IM_INT (1) (controls YCbCr auto conversion) default 1

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 JFERROR.

jinclude.h - standard JFFLUSH and JFERROR definitions, and new macro HAVE_JFIO.

jdhuf.c - added support for OJPEG_SUPPORT in libTIFF.

new file created: jconfig.h

Changes to libEXIF:

new files config.h and _stdint.h

changed "exif-tag.c" to add new function

changed "exif-entry.c" to improve exif_entry_initialize

Comments:

Other APPx markers are ignored.

No thumbnail support.

RGB images are automatically converted to YCbCr when saved.

Also YcbCr are converted to RGB when loaded. Use AutoYCbCr=0 to disable this behavior.

Generated on Thu Oct 1 11:40:05 2009 for IM by



1.6.1

KRN - IM Kernel File Format

[\[File Formats\]](#)

Collaboration diagram for KRN - IM Kernel File Format:



Description

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

Internal Implementation.

Features

```

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

Attributes:
  Description (string)

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

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
  
```

Generated on Thu Oct 1 11:40:05 2009 for IM by



1.6.1

LED - IUP image in LED

[\[File Formats\]](#)

Collaboration diagram for LED - IUP image in LED:



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.
  
```

Attributes:
none

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

Generated on Thu Oct 1 11:40:05 2009 for IM by



1.6.1

PCX - ZSoft Picture

[\[File Formats\]](#)

Collaboration diagram for PCX - ZSoft Picture:



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 Lenght Encoding [default - since uncompressed PCX is not well supported]
Only one image.
No alpha channel.
Internally the components are always packed.
Internally the lines are arranged from top down to bottom.

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.

Generated on Thu Oct 1 11:40:05 2009 for IM by

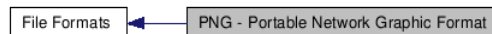


1.6.1

PNG - Portable Network Graphic Format

[\[File Formats\]](#)

Collaboration diagram for PNG - Portable Network Graphic Format:



Description

Access to the PNG file format uses libPNG version 1.2.39.
<http://www.libpng.org>
Copyright (C) 1998-2009 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(1) returns png_structp libPNG structure.

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, 2: Saturation, 3: Absolute colorimetric]
TransparencyMap IM_BYTE (N) (for MAP images is the alpha value of the corresponding palette index)
TransparencyIndex IM_BYTE (1) (for MAP images is the first index that has minimum alpha in Transparency)
TransparencyColor IM_BYTE (3) (for RGB images is the color that is full transparent)
CalibrationName, CalibrationUnits (string)
CalibrationLimits IM_INT (2)
CalibrationEquation IM_BYTE (1) [0-Linear,1-Exponential,2-Arbitrary,3-HyperbolicSine]
CalibrationParam (string) [params separated by '\\n']
Title, Author, Description, Copyright, DateTime (string)
Software, Disclaimer, Warning, Source, Comment, ... (string)
DateTimeModified (string) [when writing uses the current system time]
ICCProfile IM_BYTE (N)
ScaleUnit (string) ["meters", "radians"]
XScale, YScale IM_FLOAT (1)

```

Comments:

```

When saving PNG image with TransparencyIndex or TransparencyMap, TransparencyMap has precedence, so set
Attributes after the image are ignored.
Define PNG_NO_CONSOLE_IO to avoid printf's. We also define PNG_TIME_RFC1123_SUPPORTED.
Added the following files to the makefile to optimize the library:
pngvcrd.c - PNG_USE_PNGVCRD
            For Intel x86 CPU and Microsoft Visual C++ compiler
pnggccrd.c - PNG_USE_PNGGCCRD
            For Intel x86 CPU (Pentium-MMX or later) and GNU C compiler.

```

Generated on Thu Oct 1 11:40:05 2009 for IM by

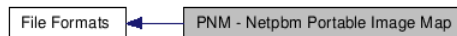


1.6.1

PNM - Netpbm Portable Image Map

[\[File Formats\]](#)

Collaboration diagram for PNM - Netpbm Portable Image Map:



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.

```

```

Attributes:
    Description (string)

```

```

Comments:
    In fact ASCII is an expansion, not a compression, because the file will be larger than binary data.

```

Generated on Thu Oct 1 11:40:05 2009 for IM by



1.6.1

RAS - Sun Raster File

[\[File Formats\]](#)

Collaboration diagram for RAS - Sun Raster File:



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.

Attributes:
 none

Generated on Thu Oct 1 11:40:05 2009 for IM by

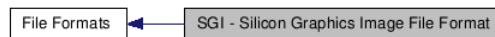


1.6.1

SGI - Silicon Graphics Image File Format

[\[File Formats\]](#)

Collaboration diagram for SGI - Silicon Graphics Image File Format:



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 palette when reading only)
 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.

Attributes:
 Description (string)

Generated on Thu Oct 1 11:40:05 2009 for IM by

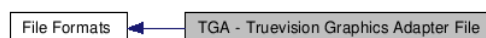


1.6.1

TGA - Truevision Graphics Adapter File

[\[File Formats\]](#)

Collaboration diagram for TGA - Truevision Graphics Adapter File:



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 top down to bottom.

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 time]
 Gamma IM_FLOAT (1)

Generated on Thu Oct 1 11:40:05 2009 for IM by



1.6.1

TIFF - Tagged Image File Format

[File Formats]

Collaboration diagram for TIFF - Tagged Image File Format:



Description

Copyright (c) 1986-1988, 1992 by Adobe Systems Incorporated.
Originally created by a group of companies, the Aldus Corporation kept the copyright until Aldus was acquired by Adobe.
TIFF Revision 6.0 Final □ June 3, 1992
<http://www.adobe.com/Support/TechNotes.html>

Access to the TIFF file format uses libTIFF version 3.9.0

<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 for Binary]
 CCITTFAX3 - CCITT Group 3 fax (binary only)
 CCITTFAX4 - CCITT Group 4 fax (binary only)
 LZW - Lempel-Ziv & Welch [default]
 JPEG - ISO JPEG [default for YBCBR]
 NEXT - NeXT 2-bit RLE (2 bpp only)
 CCITTRLEW - CCITT modified Huffman RLE with word alignment (binary only)
 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 float)
 DEFLATE - LZ77 variation (ZIP)
 ADOBE_DEFLATE - Adobe LZ77 variation
 SGILOG - SGI Log Luminance RLE for L and Luv (only byte, ushort and float) [default for XYZ]
 SGILOG24 - SGI Log 24-bit packed for Luv (only byte, ushort and float)
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(1) returns a TIFF* libTIFF structure.

Attributes:
 Photometric IM_USHORT (1) (when writing this will complement the color_mode information, for Mask, Mini
 ExtraSampleInfo IM_USHORT (1) (description of alpha channel: 0- unknown, 1- pre-multiplied, 2-normal)
 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 (string)
InkNames (strings separated by '0's)
InkSet IM_USHORT (1)
NumberOfInks IM_USHORT (1)
DotRange IM_USHORT (2)
TransferFunction0, TransferFunction1, TransferFunction3 IM_USHORT [gray=0, rgb=012]
ReferenceBlackWhite IM_FLOAT (6)
WhitePoint IM_FLOAT (2)
PrimaryChromaticities IM_FLOAT (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)
MultiBandCount IM_USHORT (1)    [Number of bands in a multiband gray image.]
MultiBandSelect IM_USHORT (1)   [Band number to read one band of a multiband gray image. Must be set be
and other TIFF tags as they are described in the TIFF documentation.
GeoTIFF tags:
    GeoTiePoints, GeoTransMatrix, IntergraphMatrix, GeoPixelScale, GeoDoubleParams IM_FLOAT (N)
    GeoASCIIParams (string)
Read-only support for EXIF tags as they are described in the EXIF 2.2 documentation. See http://www.exi
DNG tags as they are described in the DNG documentation. See http://www.adobe.com/br/products/dng/
    Tags BlackLevel, DefaultCropOrigin and DefaultCropSize are incorrectly interpreted by libTIFF so they
    Raw image is loaded in place of the thumbnail image in the main IFD.
    SubIFDCount IM_USHORT (1)    [Number of subifds of the current image.]
    SubIFDSelect IM_USHORT (1)   [Subifd number to be read. Must be set before reading image info.]
(other attributes can be obtained by using libTIFF directly using the Handle(1) function)

```

Comments:

LogLuv is in fact Y'+CIE(u,v), so we choose to always convert it to XYZ.
SubIFD is handled only for DNG.
Since LZW patent expired, LZW compression is enabled. LZW Copyright Unisys.
libGeoTIFF can be used without XTIFF initialization. Use Handle(1) to obtain a TIFF*.

Changes:

"tiff_jpeg.c" - commented "downsampled_output = TRUE" in 2 places.
New file "tif_config.h" to match our needs.
New file "tiff_binfile.c" that implement I/O routines using imBinFile.

Generated on Thu Oct 1 11:40:04 2009 for IM by

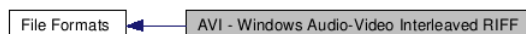


1.6.1

AVI - Windows Audio-Video Interleaved RIFF

[File Formats]

Collaboration diagram for AVI - Windows Audio-Video Interleaved RIFF:



Functions

void [imFormatRegisterAVI](#) (void)

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. In Lua call require"imlua_avi".

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) (old)
  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 charaters of the fourfcc code)
Can have more than one image.
Can have an alpha channel (only for RGB)
Internally the components are always packed.
Lines arranged from top down to bottom or bottom up to top. But are saved always as bottom up.
Handle(0) returns NULL. imBinFile is not supported.
Handle(1) returns PAVIFILE.
Handle(2) returns PAVISTREAM.

Attributes:
  FPS IM_FLOAT (1) (should set when writing, default 15)
  AVIQuality IM_INT (1) [1-10000, default -1] (write only) [unused if compression=CUSTOM]
  KeyFrameRate IM_INT (1) (write only) [key frame frequency, if 0 not using key frames, default 15, unused]
  DataRate IM_INT (1) (write only) [kilobits/second, default 2400, unused if compression=CUSTOM]

Comments:
  Reads only the first video stream. Other streams are ignored.
  All the images have the same size, you must call imFileReadImageInfo/imFileWriteImageInfo
  at least once.
  For codecs comparsion and download go to:
  http://graphics.lcs.mit.edu/~tbuehler/video/codecs/
  http://www.fourcc.org

```

Function Documentation

`void imFormatRegisterAVI(void)`

Register the AVI Format.

In Lua, when using require"imlua_avi" this function will be automatically called.

Generated on Thu Oct 1 11:40:05 2009 for IM by

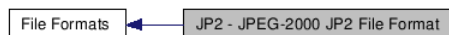


1.6.1

JP2 - JPEG-2000 JP2 File Format

[File Formats]

Collaboration diagram for JP2 - JPEG-2000 JP2 File Format:



Functions

`void imFormatRegisterJP2(void)`

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. In Lua call require"imlua_jp2".

Access to the JPEG2000 file format uses libJasper version 1.900.1

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

Copyright (c) 2001-2006 Michael David Adams.

and GeoJasPer 1.4.0

Copyright (c) 2003-2007 Dmitry V. Fedorov.
<http://www.dimin.net/software/geojasper/>

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(1) returns `jas_image_t*`
 Handle(2) returns `jas_stream_t*`

Attributes:
 CompressionRatio IM_FLOAT (1) [write only, example: Ratio=7 just like 7:1]
 GeoTIFFBox IM_BYTE (n)
 XMLPacket IM_BYTE (n)

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 `base/jas_stream.c` to export `jas_stream_create` and `jas_stream_initbuf`.
 Changed `jp2/jp2_dec.c` and `jpc/jpc_cs.c` to remove "uint" and "ulong" usage.
 The counter is restarted many times, because it has many phases.

Function Documentation

`void imFormatRegisterJP2 (void)`

Register the JP2 Format.

In Lua, when using `require"imlua_jp2"` this function will be automatically called.

Generated on Thu Oct 1 11:40:05 2009 for IM by

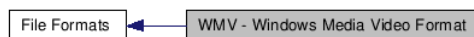


1.6.1

WMV - Windows Media Video Format

[File Formats]

Collaboration diagram for WMV - Windows Media Video Format:



Functions

`void imFormatRegisterWMV (void)`

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. In Lua call `require"imlua_wmv"`.

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.aspx>

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(0) return NULL. imBinFile is not supported.
 Handle(1) returns IWMSyncReader* when reading, IWMWriter* when writing.

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 interval in miliseconds, default 5 seconds]
- DataRate IM_INT (1) (write only) [kilobits/second, default 2400]
- VBR IM_INT (1) [0, 1] (write only) [0 - Constant Bit Rate (default), 1 - Variable Bit Rate (Quality-Bas

(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, IsVBR IM_INT (1) [0, 1]
- NumberOfFrames IM_INT (1)

Comments:

- IMPORTANT - The "image_count" and the "FPS" attribute may not be available from the file, we try to estimate from the duration and from the average time between frames, or using the default v
- We do not handle DRM protected files (Digital Rights Management).
- Reads only the first video stream. Other streams are ignored.
- All the images have the same size, you must call imFileReadImageInfo/imFileWriteImageInfo at least once.
- For optimal random reading, the file should be indexed previously.
- 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 CoUninitialize() are called every Open/Close.

Function Documentation

void imFormatRegisterWMV (void)

Register the WMF Format.
 In Lua, when using require"imlua_wmv" this function will be automatically called.



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

ECW - ECW JPEG 2000

[File Formats]

Collaboration diagram for ECW - ECW JPEG 2000:



Functions

void [imFormatRegisterECW](#) (void)

Detailed Description

Description

ECW JPEG 2000 Copyright 1998 Earth Resource Mapping Ltd. Two formats are supported with this module. The ECW (Enhanced Compression Wavelet) format and the ISO JPEG 2000 format.

Access to the ECW format uses the ECW JPEG 2000 SDK version 3.3. Available in Windows, Linux and Solaris Only. But source code is also available.

You must link the application with "im_ecw.lib" and you must call the function [imFormatRegisterECW](#) once to register the format into the IM core library.

Depends also on the ECW JPEG 2000 SDK libraries (NCSEcw.lib).

When using other JPEG 2000 libraries the first registered library will be used to guess the file format. Use the extension *.ecw to shortcut to this implementation of the JPEG 2000 format.

See [im_format_ecw.h](#)

<http://www.ermapper.com/ecw/>

The three types of licenses available for the ECW JPEG 2000 SDK are as follows:

- ECW JPEG 2000 SDK Free Use License Agreement - This license governs the free use of the ECW JPEG 2000 SDK with Unlimited Decompression and Limited Compression (Less than 500MB).
- ECW JPEG 2000 SDK Public Use License Agreement - This license governs the use of the ECW SDK with Unlimited Decompression and Unlimited Compression for applications licensed under a GNU General Public style license.
- ECW JPEG 2000 SDK Commercial Use License Agreement - This license governs the use of the ECW JPEG 2000 SDK with Unlimited Decompression and Unlimited Compression for commercial applications.

Features

Data Types: Byte, UShort, Float

Color Spaces: BINARY, GRAY, RGB, YCBCR

Compressions:

ECW - Enhanced Compression Wavelet

JPEG-2000 - ISO JPEG 2000

Only one image.

Can have an alpha channel

Internally the components are always packed.

Lines arranged from top down to bottom.

Handle() returns NCSFileView* when reading, NCSEcwCompressClient* when writing.

Attributes:

CompressionRatio IM_FLOAT (1) [example: Ratio=7 just like 7:1]

OriginX, OriginY IM_FLOAT (1)

Rotation IM_FLOAT (1)

CellIncrementX, CellIncrementY IM_FLOAT (1)

CellUnits (string)

Datum (string)

Projection (string)

ViewWidth, ViewHeight IM_INT (1) [view zoom]

ViewXmin, ViewYmin, ViewXmax, ViewYmax IM_INT (1) [view limits]

MultiBandCount IM_USHORT (1) [Number of bands in a multiband gray image.]

MultiBandSelect IM_USHORT (1) [Band number to read one band of a multiband gray image. Must be set be

Comments:

Only read support is implemented.

To read a region of the image you must set the View* attributes before reading the image data.

After reading a partial image the width and height returned in ReadImageInfo is the view size.

The view limits define the region to be read.

The view size is the actual size of the image, so the result can be zoomed.

Function Documentation

void imFormatRegisterECW (void)

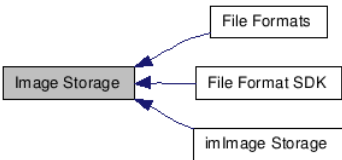
Register the ECW Format



Generated on Thu Oct 1 11:40:05 2009 for IM by 1.6.1

Image Storage

Collaboration diagram for Image Storage:



Data Structures

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

Modules

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

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 * imFileOpenAs (const char *file_name, const char *format, int *error)
imFile * imFileNew (const char *file_name, const char *format, int *error)
void imFileClose (imFile *ifile)
void * imFileHandle (imFile *ifile, int index)
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)
```

Detailed Description

See [im.h](#)

Enumeration Type Documentation

enum [imErrorCodes](#)

File Access Error Codes

Enumerator:

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

```
00058 {
00059     IM\_ERR\_NONE,          /**< No error. */
00060     IM\_ERR\_OPEN,          /**< Error while opening the file (read or write). */
00061     IM\_ERR\_ACCESS,       /**< Error while accessing the file (read or write). */
00062     IM\_ERR\_FORMAT,       /**< Invalid or unrecognized file format. */
00063     IM\_ERR\_DATA,          /**< Invalid or unsupported data. */
00064     IM\_ERR\_COMPRESS,     /**< Invalid or unsupported compression. */
00065     IM\_ERR\_MEM,           /**< Insuficient memory */
00066     IM\_ERR\_COUNTER       /**< Interrupted by the counter */
00067 };
```

Function Documentation

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

Opens the file for reading. It must exists. Also reads file header. It will try to identify the file format. See also [imErrorCodes](#).
In Lua the IM file metatable name is "imFile". When converted to a string will return "imFile(%p)" where p is replaced by the userdata address. If the file is already closed by im.FileClose, then it will return also the suffix "-closed".

```
im.FileOpen(file_name: string) -> ifile: imFile, error: number [in Lua 5]
```

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

Opens the file for reading using a specific format. It must exists. Also reads file header. See also [imErrorCodes](#) and [File Formats](#).

```
im.FileOpenAs(file_name, format: string) -> ifile: imFile, error: number [in Lua 5]
```

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

Creates a new file for writing using a specific format. If the file exists will be replaced.
It will only initialize the format driver and create the file, no data is actually written. See also [imErrorCodes](#) and [File Formats](#).

```
im.FileNew(file_name: string, format: string) -> ifile: imFile, error: number [in Lua 5]
```

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

Closes the file.
In Lua if this function is not called, the file is closed by the garbage collector.

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

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

```
void* imFileHandle ( imFile * ifile,
                   int      index
                   )
```

Returns an internal handle. index=0 returns always an imBinFile* handle, but for some formats returns NULL because they do not use imBinFile (like AVI and WMV). index=1 return an internal structure used by the format, usually is a handle to a third party library structure. This is file format dependent.

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

```
void imFileGetInfo ( imFile * ifile,
                   char *   format,
                   char *   compression,
                   int *    image_count
                   )
```

Returns file information. image_count is the number of images in a stack or the number of frames in a video/animation or the depth of a volume data.
compression and image_count can be NULL.
These informations are also available as attributes:

```
FileFormat (string)
```

```
FileCompression (string)
```

```
FileImageCount IM_INT (1)
```

See also [File Formats](#).

```
ifile:GetInfo() -> format: string, compression: string, image_count: number [in Lua 5]
```

```
void imFileSetInfo ( imFile *   ifile,
                   const char * compression
                   )
```

Changes the write compression method.
If the compression is not supported will return an error code when writting.

Use NULL to set the default compression. You can use the `imFileGetInfo` to retrieve the actual compression but only after [imFileWriteImageInfo](#). Only a few formats allow you to change the compression between frames.

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

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

Changes an extended attribute.

The data will be internally duplicated.

If data is NULL the attribute is removed. If data_type is BYTE then count can be -1 to indicate a NULL terminated string. See also [imDataType](#).

```
ifile:SetAttribute(attrib: string, data_type: number, data: table of numbers or string) [in Lua 5]
```

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. See also [imDataType](#).

```
ifile:GetAttribute(attrib: string, [as_string: boolean]) -> data: table of numbers or string, data_type: numb
```

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

```
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 *   palette,
                        int *    palette_count
                      )
```

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 * ifile,
                        long *   palette,
                        int      palette_count
                      )
```

Changes the palette.

"palette_count" is >0 and <=256.

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

```
int imFileReadImageInfo ( imFile * ifile,
                           int      index,
                           int *    width,
                           int *    height,
                           int *    file_color_mode,
                           int *    file_data_type
                         )
```


Reads the image header if any and returns image information.

Reads also the extended image attributes, so other image attributes will be available only after calling this function.

Returns an error code. index specifies the image number between 0 and image_count-1.

Some drivers reads only in sequence, so "index" can be ignored by the format driver.

Any parameters can be NULL. This function must be called at least once, check each format documentation. See also [imErrorCodes](#), [imDataType](#), [imColorSpace](#) and [imColorModeConfig](#).

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

Default index is 0.

```
int imFileWriteImageInfo ( imFile * ifile,
                          int      width,
                          int      height,
                          int      user_color_mode,
                          int      user_data_type
                        )
```

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

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

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

Returns an error code. This function must be called at least once, check each format documentation. See also [imErrorCodes](#), [imDataType](#), [imColorSpace](#) and [imColorModeConfig](#).

```
ifile:WriteImageInfo(width: number, height: number, user_color_mode: number, user_data_type: number) -> error
```

```
int imFileReadImageData ( imFile * ifile,
                          void *   data,
                          int      convert2bitmap,
                          int      color_mode_flags
                        )
```

Reads the image data with or without conversion.

The data can be converted to bitmap when reading. Data type conversion to byte will always scan for min-max then scale to 0-255, except integer values that min-max are already between 0-255. Complex to real conversions will use the magnitude.

Color mode flags contains packed, alpha and top-bottom information. If flag is 0 means unpacked, no alpha and bottom up. If flag is -1 the file original flags are used.

Returns an error code. See also [imErrorCodes](#), [imDataType](#), [imColorSpace](#) and [imColorModeConfig](#).

```
ifile:ReadImageData(data: userdata, convert2bitmap: boolean, color_mode_flags: number) -> error: number [in L
```

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

Writes the image data.

Returns an error code.

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



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

imImage Storage

[\[Image Storage\]](#)

Collaboration diagram for imImage Storage:



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)

[imImage](#)[imFileLoadImageRegion](#) ([imFile](#) *ifile, int index, int bitmap, int *error, int xmin, int xmax, int ymin, int ymax, int width, int height)

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)
imImageimFileImageLoadRegion (const char *file_name, int index, int bitmap, int *error, int xmin, int xmax, int ymin, int ymax, int
    * width, int height)
    int imFileImageSave (const char *file_name, const char *format, const imImage *image)

```

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](#)

Function Documentation

```

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

```

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.

Attributes from the file will be stored at the image. See also [imErrorCodes](#).

```

ifile:LoadImage([index: number]) -> image: imImage, error: number [in Lua 5]

```

Default index is 0.

```

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

```

Loads an image from an already open file. Returns NULL if failed.

This function assumes that the image in the file has the same parameters as the given 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.

Attributes from the file will be stored at the image. See also [imErrorCodes](#).

```

ifile:LoadImageFrame(index: number, image: imImage) -> error: number [in Lua 5]

```

Default index is 0.

```

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

```

Loads an image from an already open file, but forces the image to be a bitmap.

The returned image will be always a Bitmap image, with color_space RGB, MAP, GRAY or BINARY, and data_type IM_BYTE.

index specifies the image number between 0 and image_count-1.

Returns NULL if failed. Attributes from the file will be stored at the image. See also [imErrorCodes](#).

```

ifile:LoadBitmap([index: number]) -> image: imImage, error: number [in Lua 5]

```

Default index is 0.

```

imImage* imFileLoadImageRegion ( imFile * ifile,
                                 int      index,
                                 int      bitmap,
                                 int *    error,
                                 int      xmin,
                                 int      xmax,
                                 int      ymin,
                                 int      ymax,
                                 int      width,
                                 int      height

```

)

Loads an image region 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, or will be a Bitmap image.

Attributes from the file will be stored at the image. See also [imErrorCodes](#).

For now, it works only for the ECW file format.

```
ifile:LoadRegion(index, bitmap, xmin, xmax, ymin, ymax, width, height: number) -> image: imImage, error: numb
```

Default index is 0.

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

Loads an image from an already open file, but forces the image to be a bitmap.

This function assumes that the image in the file has the same parameters as the given image.

The image must be a Bitmap image, with color_space RGB, MAP, GRAY or BINARY, and data_type IM_BYTE.

index specifies the image number between 0 and image_count-1.

Returns NULL if failed. Attributes from the file will be stored at the image. See also [imErrorCodes](#).

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

Default index is 0.

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

Saves the image to an already open file.

This will call [imFileWriteImageInfo](#) and [imFileWriteImageData](#).

Attributes from the image will be stored at the file. Returns error code.

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

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

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. Attributes from the file will be stored at the image. See also [imErrorCodes](#).

```
im.FileImageLoad(file_name: string, [index: number]) -> image: imImage, error: number [in Lua 5]
```

Default index is 0.

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

Loads an image from file, but forces the image to be a bitmap. Open, loads and closes the file.

index specifies the image number between 0 and image_count-1.

Returns NULL if failed. Attributes from the file will be stored at the image. See also [imErrorCodes](#).

```
im.FileImageLoadBitmap(file_name: string, [index: number]) -> image: imImage, error: number [in Lua 5]
```

Default index is 0.

```
imImage* imFileImageLoadRegion ( const char * file_name,
                                 int          index,
                                 int          bitmap,
                                 int *       error,
                                 int          xmin,
                                 int          xmax,
                                 int          ymin,
                                 int          ymax,
```

```

        int      width,
        int      height
    )

```

Loads an image region from file. Open, loads and closes the file.

index specifies the image number between 0 and image_count-1.

Returns NULL if failed. Attributes from the file will be stored at the image. See also [imErrorCodes](#).

For now, it works only for the ECW file format.

```
im.FileImageLoadRegion(file_name: string, index, bitmap, xmin, xmax, ymin, ymax, width, height: number, ) ->
```

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.

Attributes from the image will be stored at the file.

```
im.FileImageSave(file_name: string, format: string, image: imImage) -> error: number [in Lua 5]
```

```
image:Save(file_name: string, format: string) -> error: number [in Lua 5]
```



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

File Format SDK

[[Image Storage](#)]

Collaboration diagram for File Format SDK:



Data Structures

```

struct imFile
    Image File Format Base Class (SDK Use Only). More...
class imFileFormatBase
    Image File Format Virtual Class (SDK Use Only). More...
class imFormat
    Image File Format Descriptor (SDK Use Only). More...

```

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 imFileSetBaseAttributes (imFile *ifile)
void imFormatRegister (imFormat *iformat)

```

Detailed Description

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

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

See [im_file.h](#)

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 imFileSetBaseAttributes ( imFile * ifile )
```

Set the attributes FileFormat, FileCompression and FileImageCount.
Used in imFileOpen and imFileOpenAs, and after the attribute list cleared with RemoveAll.

```
void imFormatRegister ( imFormat * iformat )
```

Register a format driver.



Generated on Thu Oct 1 11:40:04 2009 for IM by 1.6.1

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.

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. In Lua call require"imlua_capture".

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

Capture Samples

Capture and GLUT

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](#)

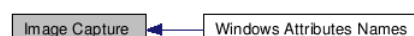
Capture and IUP

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

Collaboration diagram for Image Capture:



Modules

[Windows Attributes Names](#)

Functions

```

    int IM_DECL imVideoCaptureDeviceCount (void)
    const char *IM_DECL imVideoCaptureDeviceDesc (int device)
    const char *IM_DECL imVideoCaptureDeviceExDesc (int device)
    const char *IM_DECL imVideoCaptureDevicePath (int device)
    const char *IM_DECL imVideoCaptureDeviceVendorInfo (int device)
    int IM_DECL imVideoCaptureReloadDevices (void)
imVideoCapture *IM_DECL imVideoCaptureCreate (void)
    void IM_DECL imVideoCaptureDestroy (imVideoCapture *vc)
    int IM_DECL imVideoCaptureConnect (imVideoCapture *vc, int device)
    void IM_DECL imVideoCaptureDisconnect (imVideoCapture *vc)
    int IM_DECL imVideoCaptureDialogCount (imVideoCapture *vc)
    int IM_DECL imVideoCaptureShowDialog (imVideoCapture *vc, int dialog, void *parent)
    int IM_DECL imVideoCaptureSetInOut (imVideoCapture *vc, int input, int output, int cross)
    const char *IM_DECL imVideoCaptureDialogDesc (imVideoCapture *vc, int dialog)
    int IM_DECL imVideoCaptureFormatCount (imVideoCapture *vc)
    int IM_DECL imVideoCaptureGetFormat (imVideoCapture *vc, int format, int *width, int *height, char *desc)
    int IM_DECL imVideoCaptureSetFormat (imVideoCapture *vc, int format)
    void IM_DECL imVideoCaptureGetImageSize (imVideoCapture *vc, int *width, int *height)
    int IM_DECL imVideoCaptureSetImageSize (imVideoCapture *vc, int width, int height)
    int IM_DECL imVideoCaptureFrame (imVideoCapture *vc, unsigned char *data, int color_mode, int timeout)
    int IM_DECL imVideoCaptureOneFrame (imVideoCapture *vc, unsigned char *data, int color_mode)
    int IM_DECL imVideoCaptureLive (imVideoCapture *vc, int live)
    int IM_DECL imVideoCaptureResetAttribute (imVideoCapture *vc, const char *attrib, int fauto)
    int IM_DECL imVideoCaptureGetAttribute (imVideoCapture *vc, const char *attrib, float *percent)
    int IM_DECL imVideoCaptureSetAttribute (imVideoCapture *vc, const char *attrib, float percent)
    const char **IM_DECL imVideoCaptureGetAttributeList (imVideoCapture *vc, int *num_attrib)

```

Detailed Description

Functions to capture images from live video devices.

See [im_capture.h](#)

Function Documentation

int IM_DECL `imVideoCaptureDeviceCount` (void)

Returns the number of available devices.

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

const char* IM_DECL `imVideoCaptureDeviceDesc` (int *device*)

Returns the device description. Returns NULL only if it is an invalid device.

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

const char* IM_DECL `imVideoCaptureDeviceExDesc` (int *device*)

Returns the extendend device description. May return NULL.

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

const char* IM_DECL `imVideoCaptureDevicePath` (int *device*)

Returns the device path configuration. This is a unique string.

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

const char* IM_DECL `imVideoCaptureDeviceVendorInfo` (int *device*)

Returns the vendor information. May return NULL.

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

int IM_DECL `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* IM_DECL `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.

In Lua the IM videocapture metatable name is "imVideoCapture". When converted to a string will return "imVideoCapture(%p)" where p is replaced by the userdata address. If the videocapture is already destroyed by im.VideoCaptureDestroy, then it will return also the suffix "-destroyed".

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

```
void IM_DECL imVideoCaptureDestroy ( imVideoCapture * vc )
```

Destroys a imVideoCapture object.

In Lua if this function is not called, the videocapture is destroyed by the garbage collector.

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

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

```
int IM_DECL imVideoCaptureConnect ( imVideoCapture * vc,
                                   int          device
                                   )
```

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 IM_DECL imVideoCaptureDisconnect ( imVideoCapture * vc )
```

Disconnect from a capture device.

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

```
int IM_DECL imVideoCaptureDialogCount ( imVideoCapture * vc )
```

Returns the number of available configuration dialogs.

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

```
int IM_DECL imVideoCaptureShowDialog ( imVideoCapture * vc,
                                       int          dialog,
                                       void *       parent
                                       )
```

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: boolean [in Lua 5]
```

```
int IM_DECL imVideoCaptureSetInOut ( imVideoCapture * vc,
                                     int          input,
                                     int          output,
                                     int          cross
                                     )
```

Allows to control the input and output of devices that have multiple input and outputs. The cross index controls in which stage the input/output will be set. Usually use 1, but some capture boards has a second stage. In Direct X it controls the crossbars.

```
vc:SetInOut(input, output, cross: number) -> error: boolean [in Lua 5]
```

```
const char* IM_DECL imVideoCaptureDialogDesc ( imVideoCapture * vc,
                                                int          dialog
                                                )
```

Returns the description of a configuration dialog. dialog can be from 0 to [imVideoCaptureDialogCount](#).

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

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

Returns the number of available video formats.

Returns zero if failed.


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

```
int IM_DECL imVideoCaptureGetFormat ( imVideoCapture * vc,
                                     int          format,
                                     int *        width,
                                     int *        height,
                                     char *       desc
                                     )
```

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 can be available using [imVideoCaptureSetImageSize](#).

Returns zero if failed.

```
vc:GetFormat(format: number) -> error: boolean, width: number, height: number, desc: string [in Lua 5]
```

```
int IM_DECL imVideoCaptureSetFormat ( imVideoCapture * vc,
                                     int          format
                                     )
```

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: boolean | format: number [in Lua 5]
```

```
void IM_DECL imVideoCaptureGetImageSize ( imVideoCapture * vc,
                                          int *          width,
                                          int *          height
                                          )
```

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 IM_DECL imVideoCaptureSetImageSize ( imVideoCapture * vc,
                                          int          width,
                                          int          height
                                          )
```

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: boolean [in Lua 5]
```

```
int IM_DECL imVideoCaptureFrame ( imVideoCapture * vc,
                                  unsigned char *  data,
                                  int              color_mode,
                                  int              timeout
                                  )
```

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.

Data type is always IM_BYTE.

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: boolean [in Lua 5]
```

```
int IM_DECL imVideoCaptureOneFrame ( imVideoCapture * vc,
                                     unsigned char *  data,
                                     int              color_mode
                                     )
```

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: boolean [in Lua 5]
```

```
int IM_DECL imVideoCaptureLive ( imVideoCapture * vc,
                                int             live
                                )
```

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

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

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

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

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

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

Returns a camera or video attribute in percentage of the valid range value.
Returns zero if failed or attribute not supported.

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

```
int IM_DECL imVideoCaptureSetAttribute ( imVideoCapture * vc,
                                          const char *      attrib,
                                          float             percent
                                          )
```

Changes a camera or video attribute in percentage of the valid range value.
Returns zero if failed or attribute not supported.

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

```
const char** IM_DECL imVideoCaptureGetAttributeList ( imVideoCapture * vc,
                                                       int *            num_attrib
                                                       )
```

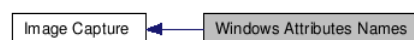
Returns a list of the description of the valid attributes for the device class.
But each device may still not support some of the returned attributes.
Use the return value of [imVideoCaptureGetAttribute](#) to check if the attribute is supported.

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

Windows Attributes Names

[\[Image Capture\]](#)

Collaboration diagram for Windows Attributes Names:



Not all attributes are supported by each device. Use the return value of [imVideoCaptureGetAttribute](#) to check if the attribute is supported.

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 in degrees Kelvin.
 VideoBacklightCompensation - Specifies the backlight compensation setting. (0/100)
 VideoGain - Specifies the gain adjustment.
 CameraPanAngle - Specifies the camera's pan angle. To 100 rotate right, To 0 rotate left (view from above).
 CameraTiltAngle - Specifies the camera's tilt angle. To 100 rotate up, To 0 rotate down.
 CameraRollAngle - Specifies the camera's roll angle. To 100 rotate right, To 0 rotate left.
 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 to the optimally focused target.
 FlipHorizontal - Specifies the video will be flipped in the horizontal direction.
 FlipVertical - Specifies the video will be flipped in the vertical direction.
 AnalogFormat - Specifies the video format standard NTSC, PAL, etc. Valid values:
 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

Generated on Thu Oct 1 11:40:04 2009 for IM by



1.6.1

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 **imProcessCrop**, **imProcess***, then **imProcessInsert** the result in the original image.

The image data of the output image is assumed to be zero before any operation. This is always true after creating a new image, but if you are reusing an image for several operation use **imImageClear** to zero the image data between operations.

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. In Lua call require"imlua_process".

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.

The image data of the output image is assumed to be zero before any operation. This is always true after creating a new image, but if you are reusing an image for several operation use **imImageClear** to zero the image data between operations.

New Operations

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

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

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

            data[offset] = 0;
        }
    }
}

void SampleProc(imImage* image)
{
    // a loop for all the color planes
    for (int d = 0; d < image->depth; d++)
    {
        // Notice that the same operation may be used to process each color component
        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 distribution of the data.
// 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->data[0], total_count);
            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

Fourier Transform

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);

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

imConvertColorSpace(rgb, gray);

// very important step, the quality of the detected lines are highly dependent on
// the quality of the binary image
// Using a simple threshold like in here maybe not a good solution for your image
imProcessPercentThreshold(gray, binary, percent=50);

// eliminates unwanted objects, depending on the quality of the threshold
// this step can be skiped
imProcessBinMorphClose(binary, binary2, 3, 1);
imProcessPrune(binary2, binary, 4, size=100, 0);

// Is there any holes in the objects?
// Holes also have borders...
imProcessFillHoles(binary, binary2, 4);

// leave only the object borders
imProcessPerimeterLine(binary2, binary);

// here you should have near only the lines you want to detect.
// if there are more or less lines that you want redo the previous steps

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

// this is optional, it will draw the results
imProcessHoughLinesDraw(gray,hough_binary,draw_hough);
```

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
```

```
where:
  X = x - width/2
  Y = y - height/2
```

because the origin of the transform is in the center of the image

Each coordinate in the transform has values in the intervals:

```
theta = 0 .. 179  (horizontal coordinate of the hough space)
rho = -rhomax .. rhomax  (vertical coordinate of the hough space,
                          vertically centered in the image)
```

```
where:
  rhomax = sqrt(width*width + height*height) /2          (width and height of the original image)
```

For each (xi, yi) 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(theta);
```

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

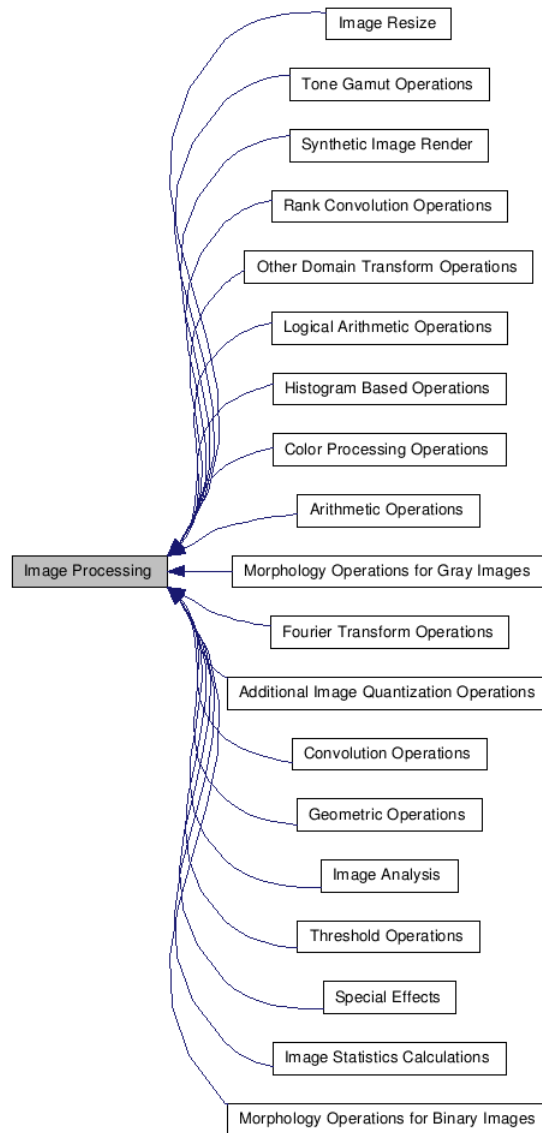
The following pseudo code illustrates the sequence of operations to measure regions. This is also called Blob Analysis.

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

Collaboration diagram for Image Processing:



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](#)

Detailed Description

Several image processing functions based on the [imImage](#) structure.

You must link the application with "im_process.lib/.a/.so". In Lua call require"imlua_process".

Some complex operations use the [Counter](#).

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



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

Synthetic Image Render

[Image Processing]

Collaboration diagram for Synthetic Image Render:



Typedefs

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

Functions

```
int imProcessRenderOp (imImage *image, imRenderFunc render_func, char *render_name, float *param, int plus)
int imProcessRenderCondOp (imImage *image, imRenderCondFunc render_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 (imImage *image, float *value)
int imProcessRenderWheel (imImage *image, int internal_radius, int external_radius)
int imProcessRenderCone (imImage *image, int radius)
int imProcessRenderTent (imImage *image, int tent_width, int tent_height)
int imProcessRenderRamp (imImage *image, int start, int end, int vert_dir)
int imProcessRenderBox (imImage *image, int box_width, int box_height)
int imProcessRenderSinc (imImage *image, float x_period, float y_period)
int imProcessRenderGaussian (imImage *image, float stddev)
int imProcessRenderLapOfGaussian (imImage *image, float stddev)
int imProcessRenderCosine (imImage *image, float x_period, float y_period)
int imProcessRenderGrid (imImage *image, int x_space, int y_space)
int imProcessRenderChessboard (imImage *image, int x_space, int y_space)
```

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](#)

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) -> value: number [in Lua 5]
```

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

Render Conditional Funtion.

```
render_cond_func(x: number, y: number, d: number, param: table of number) -> value: number, cond: boolean [in
```

Function Documentation

```
int imProcessRenderOp (imImage *   image,
                     imRenderFunc render_func,
                     char *      render_name,
                     float *      param,
                     int          plus
                     )
```

Render a synthetic image using a render function.

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

Returns zero if the counter aborted.


```

im.ProcessRenderOp(image: imImage, render_func: function, render_name: string, param: table of number, plus: 0)

int imProcessRenderCondOp ( imImage *      image,
                           imRenderCondFunc render_cond_func,
                           char *          render_name,
                           float *         param
                           )

```

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: table of number, plus: 0)

int imProcessRenderAddSpeckleNoise ( const imImage * src_image,
                                     imImage *      dst_image,
                                     float           percent
                                     )

```

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

```

im.ProcessRenderAddSpeckleNoise(src_image: imImage, dst_image: imImage, percent: number) -> counter: boolean

im.ProcessRenderAddSpeckleNoiseNew(src_image: imImage, percent: number) -> counter: boolean, new_image: imImage

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

```

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

```

im.ProcessRenderAddGaussianNoise(src_image: imImage, dst_image: imImage, mean: number, stddev: number) -> counter: boolean

im.ProcessRenderAddGaussianNoiseNew(src_image: imImage, mean: number, stddev: number) -> counter: boolean, new_image: imImage

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: imImage, mean: number, stddev: number) -> counter: boolean

im.ProcessRenderAddUniformNoiseNew(src_image: imImage, mean: number, stddev: number) -> counter: boolean, new_image: imImage

int imProcessRenderRandomNoise ( imImage * image )

```

Render random noise.

```

im.ProcessRenderRandomNoise(image: imImage) -> counter: boolean [in Lua 5]

```

```

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) -> counter: boolean [in Lua 5]

```

```

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

```

Render a centered wheel.

```

im.ProcessRenderWheel(image: imImage, internal_radius: number, external_radius: number) -> counter: boolean [in Lua 5]

```

```

int imProcessRenderCone ( imImage * image,
                          int       radius
                          )

```

)

Render a centered cone.

```
im.ProcessRenderCone(image: imImage, radius: number) -> counter: boolean [in Lua 5]
```

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

Render a centered tent.

```
im.ProcessRenderTent(image: imImage, tent_width: number, tent_height: number) -> counter: boolean [in Lua 5]
```

```
int imProcessRenderRamp ( imImage * image,
                        int      start,
                        int      end,
                        int      vert_dir
                      )
```

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

```
im.ProcessRenderRamp(image: imImage, start: number, end: number, vert_dir: boolean) -> counter: boolean [in L
```

```
int imProcessRenderBox ( imImage * image,
                        int      box_width,
                        int      box_height
                      )
```

Render a centered box.

```
im.ProcessRenderBox(image: imImage, box_width: number, box_height: number) -> counter: boolean [in Lua 5]
```

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

Render a centered sinc.

```
im.ProcessRenderSinc(image: imImage, x_period: number, y_period: number) -> counter: boolean [in Lua 5]
```

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

Render a centered gaussian.

```
im.ProcessRenderGaussian(image: imImage, stddev: number) -> counter: boolean [in Lua 5]
```

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

Render the laplacian of a centered gaussian.

```
im.ProcessRenderLapOfGaussian(image: imImage, stddev: number) -> counter: boolean [in Lua 5]
```

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

Render a centered cosine.

```
im.ProcessRenderCosine(image: imImage, x_period: number, y_period: number) -> counter: boolean [in Lua 5]
```

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

Render a centered grid.

```
im.ProcessRenderGrid(image: imImage, x_space: number, y_space: number) -> counter: boolean [in Lua 5]
```

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

Render a centered chessboard.

```
im.ProcessRenderChessboard(image: imImage, x_space: number, y_space: number) -> counter: boolean [in Lua 5]
```



Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Image Resize

[[Image Processing](#)]

Collaboration diagram for Image Resize:



Functions

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

Detailed Description

Operations to change the image size.

See [im_process_loc.h](#)

Function Documentation

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

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. If image type is IM_MAP or IM_BINARY, must use order=0. Returns zero if the counter aborted.

```
im.ProcessReduce(src_image: imImage, dst_image: imImage, order: number) -> counter: boolean [in Lua 5]
```

```
im.ProcessReduceNew(image: imImage, order: number) -> counter: boolean, new_image: imImage [in Lua 5]
```

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

Change the image size using the given interpolation order.

Supported interpolation orders:

- 0 - zero order (near neighborhood)
- 1 - first order (bilinear interpolation)
- 3 - third order (bicubic interpolation) Images must be of the same type. If image type is IM_MAP or IM_BINARY, must use order=0. Returns zero if the counter aborted.

```
im.ProcessResize(src_image: imImage, dst_image: imImage, order: number) -> counter: boolean [in Lua 5]
```

```
im.ProcessResizeNew(image: imImage, order: number) -> counter: boolean, new_image: imImage [in Lua 5]
```

```
void imProcessReduceBy4 ( const imImage * src_image,
                          imImage *      dst_image
                        )
```

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

Images must be of the same type. Destiny image size must be source image width/2, height/2. Can not operate on IM_MAP nor IM_BINARY images.

```
im.ProcessReduceBy4(src_image: imImage, dst_image: imImage) [in Lua 5]
```

```
im.ProcessReduceBy4New(image: imImage) -> new_image: imImage [in Lua 5]
```

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

Extract a rectangular region from an image.

Images must be of the same type. Destiny image size must be smaller than source image width-xmin, height-ymin. ymin and xmin must be >0 and <size.

```
im.ProcessCrop(src_image: imImage, dst_image: imImage, xmin: number, ymin: number) [in Lua 5]
```

```
im.ProcessCropNew(image: imImage, xmin: number, xmax: number, ymin: number, ymax: number) -> new_image: imImage
```

```
void imProcessInsert ( const imImage * src_image,
                       const imImage * region_image,
                       imImage *      dst_image,
                       int            xmin,
                       int            ymin
                     )
```

Insert a rectangular region in an image.

Images must be of the same type. Region image size can be larger than source image.

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, xmin: number, ymin: number) [
```

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

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

Increase the image size by adding pixels with zero value.

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

```
im.ProcessAddMargins(src_image: imImage, dst_image: imImage, xmin: number, ymin: number) [in Lua 5]
```

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

Generated on Thu Oct 1 11:40:07 2009 for IM by



1.6.1

Geometric Operations

[[Image Processing](#)]

Collaboration diagram for Geometric Operations:



Functions

void [imProcessCalcRotateSize](#) (int width, int height, int *new_width, int *new_height, double cos0, double sin0)

int [imProcessRotate](#) (const [imImage](#) *src_image, [imImage](#) *dst_image, double cos0, double sin0, int order)

int [imProcessRotateRef](#) (const [imImage](#) *src_image, [imImage](#) *dst_image, double cos0, double sin0, int x, int y, int to_origin, int order)

```

void imProcessRotate90 (const imImage *src_image, imImage *dst_image, int dir_clockwise)
void imProcessRotate180 (const imImage *src_image, imImage *dst_image)
void imProcessMirror (const imImage *src_image, imImage *dst_image)
void imProcessFlip (const imImage *src_image, imImage *dst_image)
    int imProcessRadial (const imImage *src_image, imImage *dst_image, float k1, int order)
    int imProcessSwirl (const imImage *src_image, imImage *dst_image, float k1, int order)
void imProcessInterlaceSplit (const imImage *src_image, imImage *dst_image1, imImage *dst_image2)

```

Detailed Description

Operations to change the shape of the image.

See [im_process_loc.h](#)

Function Documentation

```

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

```

Calculates the size of the new image after rotation.

```
im.ProcessCalcRotateSize(width: number, height: number, cos0: number, sin0: number) [in Lua 5]
```

```

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

```

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

Images must be of the same type. The destiny size can be calculated using [imProcessCalcRotateSize](#) to fit the new image size, or can be any size, including the original size. The rotation is relative to the center of the image.

Returns zero if the counter aborted.

```
im.ProcessRotate(src_image: imImage, dst_image: imImage, cos0: number, sin0: number, order: number) -> counter
```

```
im.ProcessRotateNew(image: imImage, cos0: number, sin0: number, order: number) -> counter: boolean, new_image
```

```

int imProcessRotateRef ( const imImage * src_image,
                        imImage *      dst_image,
                        double        cos0,
                        double        sin0,
                        int            x,
                        int            y,
                        int            to_origin,
                        int            order
                        )

```

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

Images must be of the same type. Destiny can have any size, including the original size.

The rotation is relative to the reference point. But the result can be shifted to the origin.

Returns zero if the counter aborted.

```
im.ProcessRotateRef(src_image: imImage, dst_image: imImage, cos0: number, sin0: number, x: number, y: number,
```

```
im.ProcessRotateRefNew(image: imImage, cos0: number, sin0: number, x: number, y: number, to_origin: boolean,
```

```

void imProcessRotate90 ( const imImage * src_image,
                        imImage *      dst_image,
                        int            dir_clockwise
                        )

```

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

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

Direction can be clockwise (1) or counter clockwise (-1).

```
im.ProcessRotate90(src_image: imImage, dst_image: imImage, dir_clockwise: boolean) [in Lua 5]
im.ProcessRotate90New(image: imImage, dir_clockwise: boolean) -> new_image: imImage [in Lua 5]
```

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

Rotates 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]
im.ProcessRotate180New(image: imImage) -> new_image: imImage [in Lua 5]
```

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

Mirror the image in a horizontal flip. Swap columns.
Images must be of the same type and size. Can be done in-place.

```
im.ProcessMirror(src_image: imImage, dst_image: imImage) [in Lua 5]
im.ProcessMirrorNew(image: imImage) -> new_image: imImage [in Lua 5]
```

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

Apply a vertical flip. Swap lines.
Images must be of the same type and size. Can be done in-place.

```
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 imProcessResize).
Images must be of the same type and size. Returns zero if the counter aborted.

```
im.ProcessRadial(src_image: imImage, dst_image: imImage, k1: number, order: number) -> counter: boolean [in L
im.ProcessRadialNew(image: imImage, k1: number, order: number) -> counter: boolean, new_image: imImage [in Lu
```

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

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

```
im.ProcessSwirl(src_image: imImage, dst_image: imImage, k: number, order: number) -> counter: boolean [in Lua
im.ProcessSwirlNew(image: imImage, k: number, order: number) -> counter: boolean, new_image: imImage [in Lua
```

```
void imProcessInterlaceSplit ( const imImage * src_image,
                              imImage *      dst_image1,
                              imImage *      dst_image2
                            )
```

Split the image in two images, one containing the odd lines and other containing the even lines.
Images must be of the same type. Height of the output images must be half the height of the input image. If the height of the input image is odd then the first image must have height equals to half+1.

```
im.ProcessInterlaceSplit(src_image: imImage, dst_image1: imImage, dst_image2: imImage) [in Lua 5]
```

```
im.ProcessInterlaceSplitNew(image: imImage) -> new_image1: imImage, new_image2: imImage [in Lua 5]
```

Generated on Thu Oct 1 11:40:07 2009 for IM by

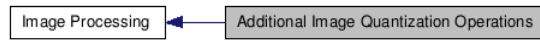


1.6.1

Additional Image Quantization Operations

[\[Image Processing\]](#)

Collaboration diagram for Additional Image Quantization Operations:



Functions

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

Detailed Description

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

See [im_process_pon.h](#)

Function Documentation

```
void imProcessQuantizeRGBUniform ( const imImage * src_image,
                                   imImage *      dst_image,
                                   int             do_dither
                                   )
```

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

```
im.ProcessQuantizeRGBUniform(src_image: imImage, dst_image: imImage, do_dither: boolean) [in Lua 5]
```

```
im.ProcessQuantizeRGBUniformNew(src_image: imImage, do_dither: boolean) -> new_image: imImage [in Lua 5]
```

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

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

```
im.ProcessQuantizeGrayUniform(src_image: imImage, dst_image: imImage, grays: number) [in Lua 5]
```

```
im.ProcessQuantizeGrayUniformNew(src_image: imImage, grays: number) -> new_image: imImage [in Lua 5]
```

Generated on Thu Oct 1 11:40:07 2009 for IM by



1.6.1

Color Processing Operations

[\[Image Processing\]](#)

Collaboration diagram for Color Processing Operations:



Functions

```
void imProcessSplitYChroma (const imImage *src_image, imImage *y_image, imImage *chroma_image)
void imProcessSplitHSI (const imImage *src_image, imImage *h_image, imImage *s_image, imImage *i_image)
void imProcessMergeHSI (const imImage *h_image, const imImage *s_image, const imImage *i_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)
```

Detailed Description

Operations to change the color components configuration.

See [im_process_pon.h](#)

Function Documentation

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

Split a RGB image into luma and chroma.

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

luma image is IM_GRAY/IM_BYTE and chroma is IM_RGB/IM_BYTE.

Source and destiny must have the same size.

```
im.ProcessSplitYChroma(src_image: imImage, y_image: imImage, chroma_image: imImage) [in Lua 5]
```

```
im.ProcessSplitYChromaNew(src_image: imImage) -> y_image: imImage, chroma_image: imImage [in Lua 5]
```

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

Split a RGB image into HSI planes.

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

Source images must be normalized to 0-1 if type is IM_FLOAT ([imProcessToneGamut](#) can be used). See [HSI Color Coordinate 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: imImage, i_image: imImage) [in Lua 5]
```

```
im.ProcessSplitHSINew(src_image: imImage) -> h_image: imImage, s_image: imImage, i_image: imImage [in Lua 5]
```

```
void imProcessMergeHSI ( const imImage * h_image,
                         const imImage * s_image,
                         const imImage * i_image,
                         imImage *      dst_image
                       )
```

Merge HSI planes into a RGB image.

Source images must be IM_GRAY/IM_FLOAT. Destiny image can be IM_RGB/IM_BYTE,IM_FLOAT.

Source and destiny must have the same size. See [HSI Color Coordinate System Conversions](#) for a definition of the color conversion.

```
im.ProcessMergeHSI(h_image: imImage, s_image: imImage, i_image: imImage, dst_image: imImage) [in Lua 5]
```

```
im.ProcessMergeHSINew(h_image: imImage, s_image: imImage, i_image: imImage) -> dst_image: imImage [in Lua 5]
```

```
void imProcessSplitComponents ( const imImage * src_image,
                                imImage **      dst_image_list
                              )
```

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) [in Lua 5]
```

```
im.ProcessSplitComponentsNew(src_image: imImage) -> dst_image_list: table of imImage [in Lua 5]
```

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

Merges separate components into a multicomponent image.

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

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


```

im.ProcessMergeComponents(src_image_list: table of imImage, dst_image: imImage) [in Lua 5]

im.ProcessMergeComponentsNew(src_image_list: table of imImage) -> dst_image: imImage [in Lua 5]

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

```

Normalize the color components by their sum. Example: $c1 = c1/(c1+c2+c3)$.
Destiny image must be IM_FLOAT.

```

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

im.ProcessNormalizeComponentsNew(src_image: imImage) -> new_image: imImage [in Lua 5]

```

```

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 numbers, dst_color: table of numbers) [in Lua 5]

im.ProcessReplaceColorNew(src_image: imImage, src_color: table of numbers, dst_color: table of numbers) -> new_image: imImage [in Lua 5]

```

Generated on Thu Oct 1 11:40:07 2009 for IM by  1.6.1

Histogram Based Operations

[\[Image Processing\]](#)

Collaboration diagram for Histogram Based Operations:



Functions

```

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

```

Detailed Description

See [im_process_pon.h](#)

Function Documentation

```

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

```

Performs an histogram expansion based on a percentage of the number of pixels.
Percentage defines an amount of pixels to include at the lowest level and at the highest level. If its is zero only empty counts of the histogram will be considered.
Images must be IM_BYTE/(IM_RGB or IM_GRAY). Can be done in place.
To expand the gammut without using the histogram, by just specifying the lowest and highest levels use the [IM_GAMUT_EXPAND](#) tone gammut operation ([imProcessToneGamut](#)).

```

im.ProcessExpandHistogram(src_image: imImage, dst_image: imImage, percent: number) [in Lua 5]

im.ProcessExpandHistogramNew(src_image: imImage, percent: number) -> new_image: imImage [in Lua 5]

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

```

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) [in Lua 5]

im.ProcessEqualizeHistogramNew(src_image: imImage) -> new_image: imImage [in Lua 5]
```



Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Threshold Operations

[Image Processing]

Collaboration diagram for Threshold Operations:



Functions

```

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

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](#)

Function Documentation

```

int imProcessRangeContrastThreshold ( const imImage * src_image,
                                     imImage *      dst_image,
                                     int              kernel_size,
                                     int              min_range
                                     )
  
```

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

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

Returns zero if the counter aborted.

```

im.ProcessRangeContrastThreshold(src_image: imImage, dst_image: imImage, kernel_size: number, min_range: numb
im.ProcessRangeContrastThresholdNew(image: imImage, kernel_size: number, min_range: number) -> counter: boole
  
```

```

int imProcessLocalMaxThreshold ( const imImage * src_image,
                                 imImage *      dst_image,
                                 int              kernel_size,
                                 int              min_level
                                 )
  
```

Threshold using a rank convolution with a local max function.

Returns zero if the counter aborted.

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

```
im.ProcessLocalMaxThreshold(src_image: imImage, dst_image: imImage, kernel_size: number, min_level: number) -
im.ProcessLocalMaxThresholdNew(image: imImage, kernel_size: number, min_level: number) -> counter: boolean, n

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

Apply a manual threshold.

threshold = a <= level ? 0: value

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

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

```
im.ProcessThreshold(src_image: imImage, dst_image: imImage, level: number, value: number) [in Lua 5]
im.ProcessThresholdNew(src_image: imImage, level: number, value: number) -> new_image: imImage [in Lua 5]

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

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, dst_image: imImage) [in Lua 5]
im.ProcessThresholdByDiffNew(src_image1: imImage, src_image2: imImage) -> new_image: imImage [in Lua 5]

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

Apply a threshold by the Hysteresis method.

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

Supports only IM_BYTE images. Note: could not find the original source code author name.

```
im.ProcessHysteresisThreshold(src_image: imImage, dst_image: imImage, low_thres: number, high_thres: number)
im.ProcessHysteresisThresholdNew(src_image: imImage, low_thres: number, high_thres: number) -> new_image: imImage

void imProcessHysteresisThresEstimate ( const imImage * image,
                                       int *          low_level,
                                       int *          high_level
                                      )
```

Estimates hysteresis low and high threshold levels.

Supports only IM_BYTE images. Usefull for [imProcessHysteresisThreshold](#).

```
im.ProcessHysteresisThresEstimate(image: imImage) -> low_level: number, high_level: number [in Lua 5]

int imProcessUniformErrThreshold ( const imImage * src_image,
                                   imImage *      dst_image
                                 )
```

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

Supports only IM_BYTE images. 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) -> level: number [in Lua 5]
im.ProcessUniformErrThresholdNew(src_image: imImage) -> level: number, new_image: imImage [in Lua 5]

void imProcessDifusionErrThreshold ( const imImage * src_image,
                                     imImage *      dst_image,
                                     int           level
                                   )
```

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

```
im.ProcessDifusionErrThreshold(src_image: imImage, dst_image: imImage, level: number) [in Lua 5]
im.ProcessDifusionErrThresholdNew(src_image: imImage, level: number) -> new_image: imImage [in Lua 5]

int imProcessPercentThreshold ( const imImage * src_image,
                                imImage *      dst_image,
                                float          percent
                              )
```

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

Supports only IM_BYTE images. Returns the used level.

```
im.ProcessPercentThreshold(src_image: imImage, dst_image: imImage, percent: number) -> level: number [in Lua 5]
im.ProcessPercentThresholdNew(src_image: imImage, percent: number) -> level: number, new_image: imImage [in L

int imProcessOtsuThreshold ( const imImage * src_image,
                              imImage *      dst_image
                            )
```

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

Returns the used level.

Supports only IM_BYTE images. Original implementation by Flavio Szenberg.

```
im.ProcessOtsuThreshold(src_image: imImage, dst_image: imImage) -> level: number [in Lua 5]
im.ProcessOtsuThresholdNew(src_image: imImage) -> level: number, new_image: imImage [in Lua 5]

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 destiny.

```
im.ProcessMinMaxThreshold(src_image: imImage, dst_image: imImage) -> level: number [in Lua 5]
im.ProcessMinMaxThresholdNew(src_image: imImage) -> level: number, new_image: imImage [in Lua 5]

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

Estimates Local Max threshold level for IM_BYTE images.

```
im.ProcessLocalMaxThresEstimate(image: imImage) -> level: number [in Lua 5]

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

Apply a manual threshold using an interval.

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

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

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

```
im.ProcessSliceThreshold(src_image: imImage, dst_image: imImage, start_level: number, end_level: number) [in
im.ProcessSliceThresholdNew(src_image: imImage, start_level: number, end_level: number) -> new_image: imImage
```



Arithmetic Operations

[Image Processing]

Collaboration diagram for Arithmetic Operations:



Enumerations

```

enum imUnaryOp {
    IM\_UN\_EQL, IM\_UN\_ABS, IM\_UN\_LESS, 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\_DIF, IM\_BIN\_POW, IM\_BIN\_MIN, IM\_BIN\_MAX
}
  
```

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 imProcessBlendConst (const imImage *src_image1, const imImage *src_image2, imImage *dst_image, float alpha)
void imProcessBlend (const imImage *src_image1, const imImage *src_image2, const imImage *alpha_image, imImage *dst_image)
void imProcessSplitComplex (const imImage *src_image, imImage *dst_image1, imImage *dst_image2, int 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)
  
```

Detailed Description

Simple math operations for images.

See [im_process_pon.h](#)

Enumeration Type Documentation

enum [imUnaryOp](#)

Unary Arithmetic Operations.

Inverse and log may lead to math exceptions.

Enumerator:

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

```

00028 {
00029     IM\_UN\_EQL,    /**< equal          =    a          */
00030     IM\_UN\_ABS,    /**< absolute       =    |a|         */
00031     IM\_UN\_LESS,   /**< less          =    -a          */
00032     IM\_UN\_INV,    /**< invert        =    1/a         (#) */
00033     IM\_UN\_SQR,    /**< square        =    a*a         */
00034     IM\_UN\_SQRT,   /**< square root   =    a^(1/2)    */
00035     IM\_UN\_LOG,    /**< natural logarithm = ln(a)    (#) */
  
```

```

00036 IM\_UN\_EXP,      /**< exponential      = exp(a)      */
00037 IM\_UN\_SIN,      /**< sine          = sin(a)      */
00038 IM\_UN\_COS,      /**< cosine         = cos(a)      */
00039 IM\_UN\_CONJ,     /**< complex conjugate = ar - ai*i      */
00040 IM\_UN\_CPXNORM /**< complex normalization by magnitude = a / cpxmag(a) */
00041 };

```

enum [imBinaryOp](#)

Binary Arithmetic Operations.

Divide 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

```

```

00061 {
00062 IM\_BIN\_ADD,      /**< add          = a+b      */
00063 IM\_BIN\_SUB,      /**< subtract     = a-b      */
00064 IM\_BIN\_MUL,      /**< multiply     = a*b      */
00065 IM\_BIN\_DIV,      /**< divide       = a/b      (#) */
00066 IM\_BIN\_DIFF,     /**< difference   = |a-b|     */
00067 IM\_BIN\_POW,      /**< power        = a^b      */
00068 IM\_BIN\_MIN,      /**< minimum      = (a < b)? a: b */
00069 IM\_BIN\_MAX,      /**< maximum      = (a > b)? a: b */
00070 };

```

Function Documentation

```

void imProcessUnArithmeticOp ( const imImage * src_image,
                               imImage *      dst_image,
                               int           op
                             )

```

Apply an arithmetic unary operation.

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 If destiny is byte, then the result is cropped to 0-255.

```
im.ProcessUnArithmeticOp(src_image: imImage, dst_image: imImage, op: number) [in Lua 5]
```

```
im.ProcessUnArithmeticOpNew(image: imImage, op: number) -> new_image: imImage [in Lua 5]
```

```

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

```

Apply a binary arithmetic operation.

Can be done in place, images must match size.

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

- byte -> byte, ushort, int, float
- ushort -> ushort, int, float
- int -> int, float
- float -> float
- complex -> complex One exception is that you can combine complex with float resulting complex. If destiny is byte, then the result is cropped to 0-255.

```
im.ProcessArithmeticOp(src_image1: imImage, src_image2: imImage, dst_image: imImage, op: number) [in Lua 5]
```

```
im.ProcessArithmeticOpNew(image1: imImage, image2: imImage, op: number) -> new_image: imImage [in Lua 5]
```

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

```
void imProcessArithmeticConstOp ( const imImage * src_image,
                                float      src_const,
                                imImage *   dst_image,
                                int         op
                                )
```

Apply a binary arithmetic operation with a constant value.
Can be done in place, images must match size.
Destiny image can be several types depending on source:

- byte -> byte, ushort, int, float
- ushort -> byte, ushort, int, float
- int -> byte, ushort, int, float
- float -> float
- complex -> complex The constant value is type casted to an appropriate type before the operation. If destiny is byte, then the result is cropped to 0-255.

```
im.ProcessArithmeticConstOp(src_image: imImage, src_const: number, dst_image: imImage, op: number) [in Lua 5]
```

```
im.ProcessArithmeticConstOpNew(image: imImage, src_const: number, op: number) -> new_image: imImage [in Lua 5]
```

```
void imProcessBlendConst ( const imImage * src_image1,
                           const imImage * src_image2,
                           imImage *       dst_image,
                           float          alpha
                           )
```

Blend two images using an alpha value = $[a * \alpha + b * (1 - \alpha)]$.
Can be done in place, images must match size and type.
alpha value must be in the interval $[0.0 - 1.0]$.

```
im.ProcessBlendConst(src_image1: imImage, src_image2: imImage, dst_image: imImage, alpha: number) [in Lua 5]
```

```
im.ProcessBlendConstNew(image1: imImage, image2: imImage, alpha: number) -> new_image: imImage [in Lua 5]
```

```
void imProcessBlend ( const imImage * src_image1,
                      const imImage * src_image2,
                      const imImage * alpha_image,
                      imImage *       dst_image
                      )
```

Blend two images using an alpha channel = $[a * \alpha + b * (1 - \alpha)]$.
Can be done in place, images must match size and type.
alpha_image must have the same data type except for complex images that must be float, and color_space must be IM_GRAY. integer alpha values must be:

```
0 - 255          IM_BYTE
0 - 65535        IM_USHORT
0 - 2147483647  IM_INT
```

that will be normalized to 0 - 1.

```
im.ProcessBlend(src_image1: imImage, src_image2: imImage, alpha_image: imImage, dst_image: imImage) [in Lua 5]
```

```
im.ProcessBlendNew(image1: imImage, image2: imImage, alpha_image: imImage) -> new_image: imImage [in Lua 5]
```

```
void imProcessSplitComplex ( const imImage * src_image,
                             imImage *       dst_image1,
                             imImage *       dst_image2,
                             int             polar
                             )
```

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, polar: boolean) [in Lua 5]
```

```
im.ProcessSplitComplexNew(image: imImage, polar: boolean) -> dst_image1: imImage, dst_image2: imImage [in Lua 5]
```

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

Merges two images as the real and imaginary parts of a complex image,
or as magnitude and phase parts (polar = 1).
Source images must be IM_FLOAT, destiny image must be IM_CFLOAT.

```
im.ProcessMergeComplex(src_image1: imImage, src_image2: imImage, dst_image: imImage, polar: boolean) [in Lua 5]
im.ProcessMergeComplexNew(image1: imImage, image2: imImage, polar: boolean) -> new_image: imImage [in Lua 5]
```

```
void imProcessMultipleMean ( const imImage ** src_image_list,
                             int src_image_count,
                             imImage * dst_image
                           )
```

Calculates the mean of multiple images.
Images must match size and type.

```
im.ProcessMultipleMean(src_image_list: table of imImage, dst_image: imImage) [in Lua 5]
im.ProcessMultipleMeanNew(src_image_list: table of imImage) -> new_image: imImage [in Lua 5]
```

```
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) [in Lua 5]
im.ProcessMultipleStdDevNew(src_image_list: table of imImage, mean_image: imImage) -> new_image: imImage [in Lua 5]
```

```
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.
Destiny is IM_FLOAT.

```
im.ProcessAutoCovariance(src_image: imImage, mean_image: imImage, dst_image: imImage) -> counter: boolean [in Lua 5]
im.ProcessAutoCovarianceNew(src_image: imImage, mean_image: imImage) -> counter: boolean, new_image: imImage [in Lua 5]
```

```
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. Conj(img1) * img2
Can be done in-place.

```
im.ProcessMultiplyConj(src_image1: imImage, src_image2: imImage, dst_image: imImage) [in Lua 5]
im.ProcessMultiplyConjNew(src_image1: imImage, src_image2: imImage) -> new_image: imImage [in Lua 5]
```



Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Logical Arithmetic Operations

[\[Image Processing\]](#)

Collaboration diagram for Logical Arithmetic Operations:



Enumerations

```
enum imLogicOp { IM\_BIT\_AND, IM\_BIT\_OR, IM\_BIT\_XOR }
```


Functions

```
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)
```

Detailed Description

Logical binary math operations for images.

See [im_process_pon.h](#)

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)

```
00318 {
00319     IM\_BIT\_AND,    /**< and   =   a & b   */
00320     IM\_BIT\_OR,      /**< or    =   a | b   */
00321     IM\_BIT\_XOR,     /**< xor   =  ~(a | b)  */
00322 };
```

Function Documentation

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

Apply a logical operation.

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

```
im.ProcessBitwiseOp(src_image1: imImage, src_image2: imImage, dst_image: imImage, op: number) [in Lua 5]
```

```
im.ProcessBitwiseOpNew(src_image1: imImage, src_image2: imImage, op: number) -> new_image: imImage [in Lua 5]
```

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

Apply a logical NOT operation.

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

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

```
im.ProcessBitwiseNotNew(src_image: imImage) -> new_image: imImage [in Lua 5]
```

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

Apply a bit mask.

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

Images must have data type IM_BYTE. It is valid only for AND, OR and XOR. Can be done in place.

```
im.ProcessBitMask(src_image: imImage, dst_image: imImage, mask: string, op: number) [in Lua 5]
```

```
im.ProcessBitMaskNew(src_image: imImage, mask: string, op: number) -> new_image: imImage [in Lua 5]
```

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

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

Extract or Reset a bit plane. For ex: 000X0000 or XXX0XXXX (plane=3).
Images must have data type IM_BYTE. Can be done in place.

```
im.ProcessBitPlane(src_image: imImage, dst_image: imImage, plane: number, do_reset: boolean) [in Lua 5]
```

```
im.ProcessBitPlaneNew(src_image: imImage, plane: number, do_reset: boolean) -> new_image: imImage [in Lua 5]
```

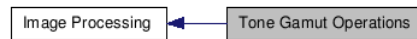


Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Tone Gamut Operations

[[Image Processing](#)]

Collaboration diagram for Tone Gamut Operations:



Enumerations

```
enum imToneGamut {
    IM\_GAMUT\_NORMALIZE, IM\_GAMUT\_POW, IM\_GAMUT\_LOG, IM\_GAMUT\_EXP,
    IM\_GAMUT\_INVERT, IM\_GAMUT\_ZEROSTART, IM\_GAMUT\_SOLARIZE, IM\_GAMUT\_SLICE,
    IM\_GAMUT\_EXPAND, IM\_GAMUT\_CROP, IM\_GAMUT\_BRIGHTCONT
}
```

Functions

```
void imProcessToneGamut (const imImage *src_image, imImage *dst_image, int op, float *param)
void imProcessUnNormalize (const imImage *src_image, imImage *dst_image)
void imProcessDirectConv (const imImage *src_image, imImage *dst_image)
void imProcessNegative (const imImage *src_image, imImage *dst_image)
```

Detailed Description

Operations that try to preserve the min-max interval in the output (the dynamic range).

See [im_process_pon.h](#)

Enumeration Type Documentation

```
enum imToneGamut
```

Tone Gamut Operations.

Enumerator:

IM_GAMUT_NORMALIZE	normalize = (a-min) / (max-min) (destiny image must be IM_FLOAT)
IM_GAMUT_POW	pow = ((a-min) / (max-min))^gamma * (max-min) + min param[0]=gamma
IM_GAMUT_LOG	log = log(K * (a-min) / (max-min) + 1) * (max-min) / log(K+1) + min param[0]=K (K>0)
IM_GAMUT_EXP	exp = (exp(K * (a-min) / (max-min)) - 1) * (max-min) / (exp(K)-1) + min param[0]=K
IM_GAMUT_INVERT	invert = max - (a-min)
IM_GAMUT_ZEROSTART	zerostart = a - min
IM_GAMUT_SOLARIZE	solarize = a < level ? a : (level * (max-min) - a * (level-min)) / (max-level) param[0]=level percentage (0-100) relative to min-max photography solarization effect.
IM_GAMUT_SLICE	slice = start < a a > end ? min : binarize ? max : a param[0]=start, param[1]=end, param[2]=binarize
IM_GAMUT_EXPAND	expand = a < start ? min : a > end ? max : (a-start) * (max-min) / (end-start) + min param[0]=start, param[1]=end
IM_GAMUT_CROP	crop = a < start ? start : a > end ? end : a param[0]=start, param[1]=end
IM_GAMUT_BRIGHTCONT	brightcont = a < min ? min : a > max ? max : a * tan(c_a) + b_s + (max-min) * (1 - tan(c_a)) / 2

param[0]=bright_shift (-100%..+100%), param[1]=contrast_factor (-100%..+100%)
change brightness and contrast simultaneously.

```

00506      {
00507      IM\_GAMUT\_NORMALIZE, /**< normalize = (a-min) / (max-min)      (destiny image must be IM_FLOAT)      */
00508      IM\_GAMUT\_POW,      /**< pow      = ((a-min) / (max-min))^gamma * (max-min) + min      \
00509                        param[0]=gamma      */
00510      IM\_GAMUT\_LOG,      /**< log      = log(K * (a-min) / (max-min) + 1)*(max-min)/log(K+1) + min      \
00511                        param[0]=K      (K>0)      */
00512      IM\_GAMUT\_EXP,      /**< exp      = (exp(K * (a-min) / (max-min)) - 1)*(max-min)/(exp(K)-1) + min \
00513                        param[0]=K      */
00514      IM\_GAMUT\_INVERT, /**< invert      = max - (a-min)      */
00515      IM\_GAMUT\_ZEROSTART, /**< zerostart = a - min      */
00516      IM\_GAMUT\_SOLARIZE, /**< solarize = a < level ? a: (level * (max-min) - a * (level-min)) / (max-le
00517                        param[0]=level percentage (0-100) relative to min-max      \n
00518                        photography solarization effect. */
00519      IM\_GAMUT\_SLICE,      /**< slice      = start < a || a > end ? min: binarize? max: a
00520                        param[0]=start, param[1]=end, param[2]=binarize      */
00521      IM\_GAMUT\_EXPAND,      /**< expand      = a < start ? min: a > end ? max : (a-start)*(max-min)/(end-star
00522                        param[0]=start, param[1]=end      */
00523      IM\_GAMUT\_CROP,      /**< crop      = a < start ? start: a > end ? end : a
00524                        param[0]=start, param[1]=end      */
00525      IM\_GAMUT\_BRIGHTCONT /**< brightcont = a < min ? min: a > max ? max: a * tan(c_a) + b_s + (max-min
00526                        param[0]=bright_shift (-100%..+100%), param[1]=contrast_factor
00527                        change brightness and contrast simultaneously. */
00528 };

```

Function Documentation

```

void imProcessToneGamut ( const imImage * src_image,
                        imImage *      dst_image,
                        int          op,
                        float *      param
                        )

```

Apply a gamut operation with arguments.

Supports all data types except IM_CFLOAT.

The linear operation do a special conversion when min > 0 and max < 1, it forces min=0 and max=1.

IM_BYTE images have min=0 and max=255 always.

Can be done in place. When there is no extra params, can use NULL.

```
im.ProcessToneGamut(src_image: imImage, dst_image: imImage, op: number, param: table of number) [in Lua 5]
```

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

```

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

```

Converts from (0-1) to (0-255), crop out of bounds values.

Source image must be IM_FLOAT, and destiny image must be IM_BYTE.

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

```
im.ProcessUnNormalizeNew(src_image: imImage) -> new_image: imImage [in Lua 5]
```

```

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

```

Directly converts IM_USHORT, IM_INT and IM_FLOAT into IM_BYTE images.

This can also be done using [imConvertDataType](#) with IM_CAST_DIRECT.

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

```
im.ProcessDirectConvNew(src_image: imImage) -> new_image: imImage [in Lua 5]
```

```

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

```

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

Supports all color spaces and all data types except IM_CFLOAT.

Can be done in place.

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

```
im.ProcessNegativeNew(src_image: imImage) -> new_image: imImage [in Lua 5]
```



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

Convolution Operations

[Image Processing]

Collaboration diagram for Convolution Operations:



Modules

[Kernel Generators](#)

Functions

```

int imProcessConvolve (const imImage *src_image, imImage *dst_image, const imImage *kernel)
int imProcessConvolveSep (const imImage *src_image, imImage *dst_image, const imImage *kernel)
int imProcessConvolveDual (const imImage *src_image, imImage *dst_image, const imImage *kernel1, const imImage *kernel2)
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 imProcessLapOfGaussianConvolve (const imImage *src_image, imImage *dst_image, float stddev)
int imProcessMeanConvolve (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGaussianConvolve (const imImage *src_image, imImage *dst_image, float stddev)
int imProcessBarlettConvolve (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessSobelConvolve (const imImage *src_image, imImage *dst_image)
int imProcessPrewittConvolve (const imImage *src_image, imImage *dst_image)
int imProcessSplineEdgeConvolve (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 imGaussianStdDev2KernelSize (float stddev)
float imGaussianKernelSize2StdDev (int kernel_size)
int imProcessUnsharp (const imImage *src_image, imImage *dst_image, float stddev, float amount, float threshold)
int imProcessSharp (const imImage *src_image, imImage *dst_image, float amount, float threshold)
int imProcessSharpKernel (const imImage *src_image, const imImage *kernel, imImage *dst_image, float amount, float threshold)

```

Detailed Description

See [im_process_loc.h](#)

Function Documentation

```

int imProcessConvolve ( const imImage * src_image,
                        imImage *      dst_image,
                        const imImage * kernel
                      )

```

Base Convolution with a kernel.

Kernel can be IM_INT or IM_FLOAT, but always IM_GRAY. Use kernel size odd for better results.

Supports all data types. The border is mirrored.

Returns zero if the counter aborted. Most of the convolutions use this function.

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

```
im.ProcessConvolve(src_image: imImage, dst_image: imImage, kernel: imImage) -> counter: boolean [in Lua 5]
```

```
im.ProcessConvolveNew(image: imImage, kernel: imImage) -> counter: boolean, new_image: imImage [in Lua 5]
```

```

int imProcessConvolveSep ( const imImage * src_image,
                           imImage *      dst_image,
                           const imImage * kernel
                         )

```

Base convolution when the kernel is separable. Only the first line and the first column will be used.

Returns zero if the counter aborted.

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

```
im.ProcessConvolveSep(src_image: imImage, dst_image: imImage, kernel: imImage) -> counter: boolean [in Lua 5]
```

```
im.ProcessConvolveSepNew(image: imImage, kernel: imImage) -> counter: boolean, new_image: imImage [in Lua 5]
```

```
int imProcessConvolveDual ( const imImage * src_image,
                           imImage *      dst_image,
                           const imImage * kernell,
                           const imImage * kernel2
                           )
```

Base Convolution with two kernels. The result is the magnitude of the result of each convolution.
 Kernel can be IM_INT or IM_FLOAT, but always IM_GRAY. Use kernel size odd for better results.
 Supports all data types. The border is mirrored.
 Returns zero if the counter aborted. Most of the convolutions use this function.
 If the kernel image attribute "Description" exists it is used by the counter.

```
im.ProcessConvolveDual(src_image: imImage, dst_image: imImage, kernell, kernel2: imImage) -> counter: boolean
```

```
im.ProcessConvolveDualNew(image: imImage, kernell, kernel2: imImage) -> counter: boolean, new_image: imImage
```

```
int imProcessConvolveRep ( const imImage * src_image,
                           imImage *      dst_image,
                           const imImage * kernel,
                           int            count
                           )
```

Repeats the convolution a number of times.
 Returns zero if the counter aborted.
 If the kernel image attribute "Description" exists it is used by the counter.

```
im.ProcessConvolveRep(src_image: imImage, dst_image: imImage, kernel: imImage, count: number) -> counter: boo
```

```
im.ProcessConvolveRepNew(image: imImage, kernel: imImage, count: number) -> counter: boolean, new_image: imIm
```

```
int imProcessCompassConvolve ( const imImage * src_image,
                                imImage *      dst_image,
                                imImage *      kernel
                                )
```

Convolve with a kernel rotating it 8 times and getting the absolute maximum value.
 Kernel must be square.
 The rotation is implemented only for kernel sizes 3x3, 5x5 and 7x7.
 Supports all data types except IM_CFLOAT. Returns zero if the counter aborted.
 If the kernel image attribute "Description" exists it is used by the counter.

```
im.ProcessCompassConvolve(src_image: imImage, dst_image: imImage, kernel: imImage) -> counter: boolean [in Lu
```

```
im.ProcessCompassConvolveNew(image: imImage, kernel: imImage) -> counter: boolean, new_image: imImage [in Lua
```

```
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 destiny image must be of type IM_INT.

```
im.ProcessDiffOfGaussianConvolve(src_image: imImage, dst_image: imImage, stddev1: number, stddev2: number) ->
```

```
im.ProcessDiffOfGaussianConvolveNew(image: imImage, stddev1: number, stddev2: number) -> counter: boolean, ne
```

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

Convolution with a laplacian of a gaussian kernel.
 Supports all data types, but if source is IM_BYTE or IM_USHORT destiny image must be of type IM_INT.

```
im.ProcessLapOfGaussianConvolve(src_image: imImage, dst_image: imImage, stddev: number) -> counter: boolean [
```

```

im.ProcessLapOfGaussianConvolveNew(image: imImage, stddev: number) -> counter: boolean, new_image: imImage [in Lua]

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

```

Convolution with a kernel full of "1"s inside a circle.
Supports all data types.

```

im.ProcessMeanConvolve(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in Lua]

im.ProcessMeanConvolveNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in Lua]

int imProcessGaussianConvolve ( const imImage * src_image,
                                imImage *      dst_image,
                                float          stddev
                                )

```

Convolution with a float gaussian kernel.
If stddev is negative its magnitude will be used as the kernel size.
Supports all data types.

```

im.ProcessGaussianConvolve(src_image: imImage, dst_image: imImage, stddev: number) -> counter: boolean [in Lua]

im.ProcessGaussianConvolveNew(image: imImage, stddev: number) -> counter: boolean, new_image: imImage [in Lua]

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

```

Convolution with a barlett kernel.
Supports all data types.

```

im.ProcessBarlettConvolve(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in Lua]

im.ProcessBarlettConvolveNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in Lua]

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

```

Magnitude of the sobel convolution.
Supports all data types.

```

im.ProcessSobelConvolve(src_image: imImage, dst_image: imImage) -> counter: boolean [in Lua 5]

im.ProcessSobelConvolveNew(image: imImage) -> counter: boolean, new_image: imImage [in Lua 5]

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

```

Magnitude of the prewitt convolution.
Supports all data types.

```

im.ProcessPrewittConvolve(src_image: imImage, dst_image: imImage) -> counter: boolean [in Lua 5]

im.ProcessPrewittConvolveNew(image: imImage) -> counter: boolean, new_image: imImage [in Lua 5]

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

```

Spline edge detection.
Supports all data types.

```

im.ProcessSplineEdgeConvolve(src_image: imImage, dst_image: imImage) -> counter: boolean [in Lua 5]

im.ProcessSplineEdgeConvolveNew(image: imImage) -> counter: boolean, new_image: imImage [in Lua 5]

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

```

Finds the zero crossings of IM_INT and IM_FLOAT images. Crossings are marked with non zero values indicating the intensity of the edge. It is usually used after a second derivative, laplace.

Extracted from XITE, Copyright 1991, Blab, UiO

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

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

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

```
void imProcessCanny ( const imImage * src_image,
                     imImage *      dst_image,
                     float          stddev
                   )
```

First part of the Canny edge detector. Includes the gaussian filtering and the nonmax suppression.

After using this you could apply a Hysteresis Threshold, see [imProcessHysteresisThreshold](#).

Image must be IM_BYTE/IM_GRAY.

Implementation from the book:

```
J. R. Parker
"Algorithms for Image Processing and Computer Vision"
WILEY
```

```
im.ProcessCanny(src_image: imImage, dst_image: imImage, stddev: number) [in Lua 5]
```

```
im.ProcessCannyNew(image: imImage, stddev: number) -> new_image: imImage [in Lua 5]
```

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

Calculates the kernel size given the standard deviation.

If stddev is negative its magnitude will be used as the kernel size.

```
im.GaussianStdDev2KernelSize(stddev: number) -> kernel_size: number [in Lua 5]
```

```
float imGaussianKernelSize2StdDev ( int kernel_size )
```

Calculates the standard deviation given the kernel size.

```
im.GaussianKernelSize2StdDev(kernel_size: number) -> stddev: number [in Lua 5]
```

```
int imProcessUnsharp ( const imImage * src_image,
                     imImage *      dst_image,
                     float          stddev,
                     float          amount,
                     float          threshold
                   )
```

Edge enhancement using Unsharp mask. stddev control the gaussian filter, amount controls how much the edges will enhance the image (0<amount<1), and threshold controls which edges will be considered, it compares to twice of the absolute size of the edge. Although very similar to [imProcessSharp](#), produces better results.

```
im.ProcessUnsharp(src_image: imImage, dst_image: imImage, stddev: number, amount: number, threshold: number)
```

```
im.ProcessUnsharpNew(image: imImage, stddev: number, amount: number, threshold: number) -> new_image: imImage
```

```
int imProcessSharp ( const imImage * src_image,
                     imImage *      dst_image,
                     float          amount,
                     float          threshold
                   )
```

Edge enhancement using Laplacian8 mask. amount controls how much the edges will enhance the image (0<amount<1), and threshold controls which edges will be considered, it compares to twice of the absolute size of the edge.

```
im.ProcessSharp(src_image: imImage, dst_image: imImage, amount: number, threshold: number) [in Lua 5]
```

```
im.ProcessSharpNew(image: imImage, amount: number, threshold: number) -> new_image: imImage [in Lua 5]
```

```
int imProcessSharpKernel ( const imImage * src_image,
                          const imImage * kernel,
                          imImage *      dst_image,
                          float          amount,
                          float          threshold
                        )
```

Edge enhancement using a given kernel. If kernel has all positive values, then the unsharp technique is used, else sharp is used. amount controls how much the edges will enhance the image ($0 < \text{amount} < 1$), and threshold controls which edges will be considered, it compares to twice of the absolute size of the edge.

```
im.ProcessSharp(src_image: imImage, dst_image: imImage, amount: number, threshold: number) [in Lua 5]
```

```
im.ProcessSharpNew(image: imImage, amount: number, threshold: number) -> new_image: imImage [in Lua 5]
```



Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Kernel Generators

[[Convolution Operations](#)]

Collaboration diagram for Kernel Generators:



Functions

[imImage](#) * [imKernelSobel](#) (void)
[imImage](#) * [imKernelPrewitt](#) (void)
[imImage](#) * [imKernelKirsh](#) (void)
[imImage](#) * [imKernelLaplacian4](#) (void)
[imImage](#) * [imKernelLaplacian8](#) (void)
[imImage](#) * [imKernelLaplacian5x5](#) (void)
[imImage](#) * [imKernelLaplacian7x7](#) (void)
[imImage](#) * [imKernelGradian3x3](#) (void)
[imImage](#) * [imKernelGradian7x7](#) (void)
[imImage](#) * [imKernelSculpt](#) (void)
[imImage](#) * [imKernelMean3x3](#) (void)
[imImage](#) * [imKernelMean5x5](#) (void)
[imImage](#) * [imKernelCircularMean5x5](#) (void)
[imImage](#) * [imKernelMean7x7](#) (void)
[imImage](#) * [imKernelCircularMean7x7](#) (void)
[imImage](#) * [imKernelGaussian3x3](#) (void)
[imImage](#) * [imKernelGaussian5x5](#) (void)
[imImage](#) * [imKernelBarlett5x5](#) (void)
[imImage](#) * [imKernelTopHat5x5](#) (void)
[imImage](#) * [imKernelTopHat7x7](#) (void)
[imImage](#) * [imKernelEnhance](#) (void)

Detailed Description

Creates several known kernels

See [im_kernel.h](#)

Function Documentation

[imImage](#)* [imKernelSobel](#) (void)

Creates a kernel with the following values:

```

 1  2  1
 0  0  0
-1 -2 -1
  
```

```
im.KernelSobel() -> kernel: imImage [in Lua 5]
```

[imImage](#)* [imKernelPrewitt](#) (void)

Creates a kernel with the following values:

```

 1  1  1
 0  0  0
-1 -1 -1
  
```

```
im.KernelPrewitt() -> kernel: imImage [in Lua 5]
```

[imImage](#)* [imKernelKirsh](#) (void)

Creates a kernel with the following values:

```
5 5 5
-3 0 -3
-3 -3 -3
```

```
im.KernelKirsh() -> kernel: imImage [in Lua 5]
```

[imImage](#)* imKernelLaplacian4 (void)

Creates a kernel with the following values:

```
0 -1 0
-1 4 -1
0 -1 0
```

```
im.KernelLaplacian4() -> kernel: imImage [in Lua 5]
```

[imImage](#)* imKernelLaplacian8 (void)

Creates a kernel with the following values:

```
-1 -1 -1
-1 8 -1
-1 -1 -1
```

```
im.KernelLaplacian8() -> kernel: imImage [in Lua 5]
```

[imImage](#)* imKernelLaplacian5x5 (void)

Creates a kernel with the following values:

```
0 -1 -1 -1 0
-1 0 1 0 -1
-1 1 8 1 -1
-1 0 1 0 -1
0 -1 -1 -1 0
```

```
im.KernelLaplacian5x5() -> kernel: imImage [in Lua 5]
```

[imImage](#)* imKernelLaplacian7x7 (void)

Creates a kernel with the following values:

```
-1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
-1 -1 -1 48 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1
```

```
im.KernelLaplacian7x7() -> kernel: imImage [in Lua 5]
```

[imImage](#)* imKernelGradian3x3 (void)

Creates a kernel with the following values:

```
0 -1 0
0 1 0
0 0 0
```

```
im.KernelGradian3x3() -> kernel: imImage [in Lua 5]
```

[imImage](#)* imKernelGradian7x7 (void)

Creates a kernel with the following values:

```
0 -1 -1 0 1 1 0
-1 -2 -2 0 2 2 1
-1 -2 -3 0 3 2 1
-1 -2 -3 0 3 2 1
-1 -2 -3 0 3 2 1
-1 -2 -2 0 2 2 1
0 -1 -1 0 1 1 0
```

```
im.KernelGradian7x7() -> kernel: imImage [in Lua 5]
```

[imImage](#)* imKernelSculpt (void)

Creates a kernel with the following values:

```
-1 0 0
 0 0 0
 0 0 1
```

```
im.KernelSculpt() -> kernel: imImage [in Lua 5]
```

[imImage](#)* `imKernelMean3x3 (void)`

Creates a kernel with the following values:

```
1 1 1
1 1 1
1 1 1
```

```
im.KernelMean3x3() -> kernel: imImage [in Lua 5]
```

[imImage](#)* `imKernelMean5x5 (void)`

Creates a kernel with the following values:

```
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
```

```
im.KernelMean5x5() -> kernel: imImage [in Lua 5]
```

[imImage](#)* `imKernelCircularMean5x5 (void)`

Creates a kernel with the following values:

```
0 1 1 1 0
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
0 1 1 1 0
```

```
im.KernelMean5x5() -> kernel: imImage [in Lua 5]
```

[imImage](#)* `imKernelMean7x7 (void)`

Creates a kernel with the following values:

```
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
```

```
im.KernelMean7x7() -> kernel: imImage [in Lua 5]
```

[imImage](#)* `imKernelCircularMean7x7 (void)`

Creates a kernel with the following values:

```
0 0 1 1 1 0 0
0 1 1 1 1 1 0
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
0 1 1 1 1 1 0
0 0 1 1 1 0 0
```

```
im.KernelCircularMean7x7() -> kernel: imImage [in Lua 5]
```

[imImage](#)* `imKernelGaussian3x3 (void)`

Creates a kernel with the following values:

```
1 2 1
2 4 2
1 2 1
```

```
im.KernelGaussian3x3() -> kernel: imImage [in Lua 5]
```

[imImage](#)* `imKernelGaussian5x5 (void)`

Creates a kernel with the following values:

```

1  4  6  4  1
4 16 24 16  4
6 24 36 24  6
4 16 24 16  4
1  4  6  4  1

```

```
im.KernelGaussian5x5() -> kernel: imImage [in Lua 5]
```

[imImage](#)* [imKernelBarlett5x5](#)(void)

Creates a kernel with the following values:

```

1 2 3 2 1
2 4 6 4 2
3 6 9 6 3
2 4 6 4 2
1 2 3 2 1

```

```
im.KernelBarlett5x5() -> kernel: imImage [in Lua 5]
```

[imImage](#)* [imKernelTopHat5x5](#)(void)

Creates a kernel with the following values:

```

0 -1 -1 -1  0
-1 -1  3 -1 -1
-1  3  4  3 -1
-1 -1  3 -1 -1
0 -1 -1 -1  0

```

```
im.KernelTopHat5x5() -> kernel: imImage [in Lua 5]
```

[imImage](#)* [imKernelTopHat7x7](#)(void)

Creates a kernel with the following values:

```

0  0 -1 -1 -1  0  0
0 -1 -1 -1 -1 -1  0
-1 -1  3  3  3 -1 -1
-1 -1  3  4  3 -1 -1
-1 -1  3  3  3 -1 -1
0 -1 -1 -1 -1 -1  0
0  0 -1 -1 -1  0  0

```

```
im.KernelTopHat7x7() -> kernel: imImage [in Lua 5]
```

[imImage](#)* [imKernelEnhance](#)(void)

Creates a kernel with the following values:

```

0 -1 -2 -1  0
-1 -4  0 -4 -1
-2  0 40  0 -2
-1 -4  0 -4 -1
0 -1 -2 -1  0

```

```
im.KernelEnhance() -> kernel: imImage [in Lua 5]
```

Generated on Thu Oct 1 11:40:06 2009 for IM by



1.6.1

Rank Convolution Operations

[Image Processing]

Collaboration diagram for Rank Convolution Operations:



Functions

```

int imProcessMedianConvolve(const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessRangeConvolve(const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessRankClosestConvolve(const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessRankMaxConvolve(const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessRankMinConvolve(const imImage *src_image, imImage *dst_image, int kernel_size)

```

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](#)

Function Documentation

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

Rank convolution using the median value.
Returns zero if the counter aborted.
Supports all data types except IM_CFLOAT. Can be applied on color images.

```
im.ProcessMedianConvolve(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in 0]
im.ProcessMedianConvolveNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in 0]
```

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

Rank convolution using (maximum-minimum) value.
Returns zero if the counter aborted.
Supports all data types except IM_CFLOAT. Can be applied on color images.

```
im.ProcessRangeConvolve(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in 0]
im.ProcessRangeConvolveNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in 0]
```

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

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 images.

```
im.ProcessRankClosestConvolve(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in 0]
im.ProcessRankClosestConvolveNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in 0]
```

```
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 images.

```
im.ProcessRankMaxConvolve(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in 0]
im.ProcessRankMaxConvolveNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in 0]
```

```
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 images.

```
im.ProcessRankMinConvolve(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in 0]
im.ProcessRankMinConvolveNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in 0]
```

Morphology Operations for Binary Images

[Image Processing]

Collaboration diagram for Morphology Operations for Binary Images:



Functions

```

int imProcessBinMorphConvolve (const imImage *src_image, imImage *dst_image, const imImage *kernel, int hit_white, int iter)
int imProcessBinMorphErode (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphDilate (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphOpen (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphClose (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphOutline (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
void imProcessBinMorphThin (const imImage *src_image, imImage *dst_image)
  
```

Detailed Description

See [im_process_loc.h](#)

Function Documentation

```

int imProcessBinMorphConvolve ( const imImage * src_image,
                                imImage * dst_image,
                                const imImage * kernel,
                                int hit_white,
                                int iter
                                )
  
```

Base binary morphology convolution.

Images are all IM_BINARY. Kernel is IM_INT, but values can be only 1, 0 or -1. Use kernel size odd for better results.

Hit white means hit=1 and miss=0, or else hit=0 and miss=1.

Use -1 for don't care positions in kernel. Kernel values are simply compared with image values.

The operation can be repeated by a number of iterations. The border is zero extended.

Almost all the binary morphology operations use this function.

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

```

im.ProcessBinMorphConvolve(src_image: imImage, dst_image: imImage, kernel: imImage, hit_white: boolean, iter:
im.ProcessBinMorphConvolveNew(image: imImage, kernel: imImage, hit_white: boolean, iter: number) -> counter:
  
```

```

int imProcessBinMorphErode ( const imImage * src_image,
                              imImage * dst_image,
                              int kernel_size,
                              int iter
                              )
  
```

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

```

im.ProcessBinMorphErode(src_image: imImage, dst_image: imImage, kernel_size: number, iter: number) -> counter
im.ProcessBinMorphErodeNew(image: imImage, kernel_size: number, iter: number) -> counter: boolean, new_image:
  
```

```

int imProcessBinMorphDilate ( const imImage * src_image,
                              imImage * 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, kernel_size: number, iter: number) -> counte
im.ProcessBinMorphDilateNew(image: imImage, kernel_size: number, iter: number) -> counter: boolean, new_image
  
```

```

int imProcessBinMorphOpen ( const imImage * src_image,
  
```

```

    imImage *      dst_image,
    int           kernel_size,
    int           iter
)

```

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

```
im.ProcessBinMorphOpen(src_image: imImage, dst_image: imImage, kernel_size: number, iter: number) -> counter:
```

```
im.ProcessBinMorphOpenNew(image: imImage, kernel_size: number, iter: number) -> counter: boolean, new_image:
```

```

int imProcessBinMorphClose ( const imImage * src_image,
                             imImage *      dst_image,
                             int           kernel_size,
                             int           iter
)

```

Dilate+Erode.

```
im.ProcessBinMorphClose(src_image: imImage, dst_image: imImage, kernel_size: number, iter: number) -> counter
```

```
im.ProcessBinMorphCloseNew(image: imImage, kernel_size: number, iter: number) -> counter: boolean, new_image:
```

```

int imProcessBinMorphOutline ( const imImage * src_image,
                               imImage *      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, kernel_size: number, iter: number) -> count
```

```
im.ProcessBinMorphOutlineNew(image: imImage, kernel_size: number, iter: number) -> counter: boolean, new_imag
```

```

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

```

Thins the supplied binary image using Rosenfeld's parallel thinning algorithm.

Reference:

"Efficient Binary Image Thinning using Neighborhood Maps"

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

in "Graphics Gems IV", Academic Press, 1994

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

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

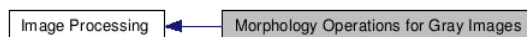


Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Morphology Operations for Gray Images

[Image Processing]

Collaboration diagram for Morphology Operations for Gray Images:



Functions

```
int imProcessGrayMorphConvolve (const imImage *src_image, imImage *dst_image, const imImage *kernel, int ismax)
```

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

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

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

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

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

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

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

Detailed Description

See [im_process_loc.h](#)

Function Documentation

```
int imProcessGrayMorphConvolve ( const imImage * src_image,
                                imImage *      dst_image,
                                const imImage * kernel,
                                int          ismax
                                )
```

Base gray morphology convolution.

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

Kernel is always IM_INT. Use kernel size odd for better results.

Use -1 for don't care positions in kernel. Kernel values are added to image values, then you can use the maximum or the minimum within the kernel area.

No border extensions are used. All the gray morphology operations use this function.

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

```
im.ProcessGrayMorphConvolve(src_image: imImage, dst_image: imImage, kernel: imImage, ismax: boolean) -> counter: boolean [in L]
im.ProcessGrayMorphConvolveNew(image: imImage, kernel: imImage, ismax: boolean) -> counter: boolean, new_image: imImage [in L]
```

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

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

```
im.ProcessGrayMorphErode(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in L]
im.ProcessGrayMorphErodeNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in L]
```

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

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

```
im.ProcessGrayMorphDilate(src_image: imImage, dst_image: imImage, kernel_size: number) -> counter: boolean [in L]
im.ProcessGrayMorphDilateNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in L]
```

```
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) -> counter: boolean [in L]
im.ProcessGrayMorphOpenNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in L]
```

```
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) -> counter: boolean [in L]
im.ProcessGrayMorphCloseNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in L]
```

```
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) -> counter: boolean [in
im.ProcessGrayMorphTopHatNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in L

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) -> counter: boolean [in
im.ProcessGrayMorphWellNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [in L

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) -> counter: boolean
im.ProcessGrayMorphGradientNew(image: imImage, kernel_size: number) -> counter: boolean, new_image: imImage [

```



Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Fourier Transform Operations

[\[Image Processing\]](#)

Collaboration diagram for Fourier Transform Operations:



Functions

```

void imProcessFFT (const imImage *src_image, imImage *dst_image)
void imProcessIFFT (const imImage *src_image, imImage *dst_image)
void imProcessFFTraw (imImage *image, int inverse, int center, int normalize)
void imProcessSwapQuadrants (imImage *image, int center2origin)

```

Detailed Description

All Fourier transforms use FFTW library.

The pre-compiled binaries for FFTW version 2.1.5 includes all the necessary files. The pre-compiled binaries for FFTW version 3.x depends on an external library, not provided. To build the code that uses FFTW version 3 you must define USE_FFTW3.

FFTW Copyright Matteo Frigo, Steven G. Johnson and the MIT.

<http://www.fftw.org>

See "fftw.h"

Must link with "im_fftw" library.

IMPORTANT: 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](#)

Function Documentation

```

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

```

Forward FFT.

The result has its lowest frequency at the center of the image.

This is an unnormalized fft.

Images must be of the same size. Destiny image must be of type complex.


```

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      normalize
                      )

```

Raw in-place FFT (forward or inverse).

The lowest frequency can be centered after forward, or can be restored to the origin before inverse.

The result can be normalized after the transform by sqrt(w*h) [1] or by (w*h) [2], or left unnormalized [0].

Images must be of the same size and both must be of type complex.

```

im.ProcessFFTraw(image: imImage, inverse: number, center: number, normalize: number) [in Lua 5]

void imProcessSwapQuadrants ( imImage * image,
                              int      center2origin
                              )

```

Auxiliary function for the raw FFT.

This is the function used internally to change the lowest frequency position in the image.

If the image size has even dimensions the flag "center2origin" is useless. But if it is odd, you must specify if its from center to origin (usually used before inverse) or from origin to center (usually used after forward).

Notice that this function is used for images in the the frequency domain.

Image type must be complex.

```

im.ProcessSwapQuadrants(image: imImage, center2origin: number) [in Lua 5]

```

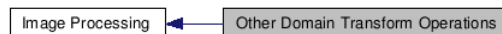


Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Other Domain Transform Operations

[\[Image Processing\]](#)

Collaboration diagram for Other Domain Transform Operations:



Functions

```

int imProcessHoughLines (const imImage *src_image, imImage *dst_image)
int imProcessHoughLinesDraw (const imImage *src_image, const imImage *hough, const imImage *hough_points, imImage
                             *dst_image)
void imProcessCrossCorrelation (const imImage *src_image1, const imImage *src_image2, imImage *dst_image)
void imProcessAutoCorrelation (const imImage *src_image, imImage *dst_image)
void imProcessDistanceTransform (const imImage *src_image, imImage *dst_image)
void imProcessRegionalMaximum (const imImage *src_image, imImage *dst_image)

```

Detailed Description

Hough, Distance.

See [im_process_glo.h](#)

Function Documentation

```
int imProcessHoughLines ( const imImage * src_image,
                        imImage *      dst_image
                      )
```

Hough Lines Transform.

It will detect white lines in a black background. So the source image must be a IM_BINARY image with the white lines of interest enhanced. The better the threshold with the white lines the better the line detection.

The destiny image must have IM_GRAY, IM_INT, hg_width=180, hg_height=2*rmax+1, where rmax is the image diagonal/2 (rmax = sqrt(width*width + height*height)).

The hough transform defines " $\cos(\theta) * X + \sin(\theta) * Y = \rho$ " and the parameters are in the interval:

$\theta = "0 .. 179"$, $\rho = "-hg_height/2 .. hg_height/2"$.

Where ρ is the perpendicular distance from the center of the image and θ the angle with the normal. So do not confuse θ with the line angle, they are perpendicular.

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: boolean [in Lua 5]
```

```
im.ProcessHoughLinesNew(image: imImage) -> counter: boolean, new_image: imImage [in Lua 5]
```

```
int imProcessHoughLinesDraw ( const imImage * src_image,
                             const imImage * hough,
                             const imImage * hough_points,
                             imImage *      dst_image
                           )
```

Draw detected hough lines.

The source image must be IM_GRAY and IM_BYTE. The destiny image can be a clone of the source image or it can be the source image for in place processing.

If the hough transform is not NULL, then the hough points are filtered to include only lines that are significantly different from each other.

The hough image is the hough transform image, but it is optional and can be NULL. If not NULL then it will be used to filter lines that are very similar.

The hough points image is a hough transform image that was thresholded to a IM_BINARY image, usually using a Local Max threshold operation (see [imProcessLocalMaxThreshold](#)). Again the better the threshold the better the results.

The destiny image will be set to IM_MAP, and the detected lines will be drawn using a red color.

Returns the number of detected lines.

```
im.ProcessHoughLinesDraw(src_image: imImage, hough: imImage, hough_points: imImage, dst_image: imImage) -> li
```

```
im.ProcessHoughLinesDrawNew(image: imImage, hough: imImage, hough_points: imImage) -> lines: number, new_imag
```

```
void imProcessCrossCorrelation ( const imImage * src_image1,
                                const imImage * src_image2,
                                imImage *      dst_image
                              )
```

Calculates the Cross Correlation in the frequency domain.

$\text{CrossCorr}(a,b) = \text{IFFT}(\text{Conj}(\text{FFT}(a)) * \text{FFT}(b))$

Images must be of the same size and only destiny image must be of type complex.

```
im.ProcessCrossCorrelation(src_image1: imImage, src_image2: imImage, dst_image: imImage) [in Lua 5]
```

```
im.ProcessCrossCorrelationNew(image1: imImage, image2: imImage) -> new_image: imImage [in Lua 5]
```

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

Calculates the Auto Correlation in the frequency domain.

Uses the cross correlation. Images must be of the same size and only destiny image must be of type complex.

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

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

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

Calculates the Distance Transform of a binary image using an approximation of the euclidian distance.

Each white pixel in the binary image is assigned a value equal to its distance from the nearest black pixel.

Uses a two-pass algorithm incrementally calculating the distance.

Source image must be IM_BINARY, destiny must be IM_FLOAT.

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

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

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

Marks all the regional maximum of the distance transform.
source is IMGRAY/IM_FLOAT destiny in IM_BINARY.
We consider maximum all connected pixel values that have smaller pixel values around it.

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

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



Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

Special Effects

[\[Image Processing\]](#)

Collaboration diagram for Special Effects:



Functions

```
void imProcessPixelate (const imImage *src_image, imImage *dst_image, int box_size)
void imProcessPosterize (const imImage *src_image, imImage *dst_image, int level)
```

Detailed Description

Operations to change image appearance.

See [im_process_pon.h](#)

Function Documentation

```
void imProcessPixelate ( const imImage * src_image,
                          imImage *      dst_image,
                          int           box_size
                          )
```

Generates a zoom in effect averaging colors inside a square region.
Operates only on IM_BYTE images.

```
im.ProcessPixelate(src_image: imImage, dst_image: imImage, box_size: number) [in Lua 5]
```

```
im.ProcessPixelateNew(src_image: imImage, box_size: number) -> new_image: imImage [in Lua 5]
```

```
void imProcessPosterize ( const imImage * src_image,
                           imImage *      dst_image,
                           int           level
                           )
```

A simple Posterize effect. It reduces the number of colors in the image eliminating less significant bit planes. Can have 1 to 7 levels. See [imProcessBitMask](#).
Images must have data type IM_BYTE.

```
im.ProcessPosterize(src_image: imImage, dst_image: imImage, level: number) [in Lua 5]
```

```
im.ProcessPosterizeNew(src_image: imImage, level: number) -> new_image: imImage [in Lua 5]
```



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

Image Statistics Calculations

[\[Image Processing\]](#)

Collaboration diagram for Image Statistics Calculations:



Data Structures

struct [imStats](#)

Typedefs

typedef struct [_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 imCalcHistoImageStatistics (const imImage *image, int *median, int *mode)
  
```

Detailed Description

Operations to calculate some statistics over images.

See [im_process_ana.h](#)

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 [in Lua 5]
```

```

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 [in Lua 5]
```

```
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: boolean) -> histo: table of numbers [in Lua 5]
```

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 * data,
                             int count,
                             unsigned long * histo,
                             int cumulative
                           )
```

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 * image,
                           unsigned long * histo,
                           int cumulative
                         )
```

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: boolean) -> histo: table [in Lua 5]
```

```
void imCalcImageStatistics ( const imImage * image,
                             imStats * stats
                           )
```

Calculates the statistics about the image data.
There is one stats for each depth plane. For ex: stats[0]=red stats, stats[0]=green stats, ...
Supports all data types except IM_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 imImage * image,
                                 imStats * stats
                               )
```

Calculates the statistics about the image histogram data.
There is one stats for each depth plane. For ex: stats[0]=red stats, stats[0]=green stats, ...
Only IM_BYTE images are supported.

```
im.CalcHistogramStatistics(image: imImage) -> stats: table [in Lua 5]
```

```
void imCalcHistoImageStatistics ( const imImage * image,
                                  int * median,
                                  int * mode
                                )
```

Calculates some extra statistics about the image histogram data.
There is one stats for each depth plane.
Only IM_BYTE images are supported.
mode will be -1 if more than one max is found.

```
im.CalcHistoImageStatistics(image: imImage) -> median: number, mode: number [in Lua 5]
```



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

Image Analysis

[\[Image Processing\]](#)

Collaboration diagram for Image Analysis:



Functions

```

int imAnalyzeFindRegions (const imImage *src_image, imImage *dst_image, int connect, int touch_border)
void imAnalyzeMeasureArea (const imImage *image, int *area, int region_count)
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, int region_count)
void imProcessPerimeterLine (const imImage *src_image, imImage *dst_image)
void imProcessRemoveByArea (const imImage *src_image, imImage *dst_image, int connect, int start_size, int end_size, int inside)
void imProcessFillHoles (const imImage *src_image, imImage *dst_image, int connect)

```

Detailed Description

See [im_process_ana.h](#)

Function Documentation

```

int imAnalyzeFindRegions ( const imImage * src_image,
                           imImage *   dst_image,
                           int         connect,
                           int         touch_border
                           )

```

Find white regions in binary image.

Result is IM_GRAY/IM_USHORT type. Regions can be 4 connected or 8 connected.

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: number, touch_border: boolean) -> coun
```

```
im.AnalyzeFindRegionsNew(image: imImage, connect: number, touch_border: boolean) -> count: number, new_image:
```

```

void imAnalyzeMeasureArea ( const imImage * image,
                            int *       area,
                            int         region_count
                            )

```

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 [imAnalyzeFindRegions](#)).

area has size the number of regions.

```
im.AnalyzeMeasureArea(image: imImage, [region_count: number]) -> area: table of numbers [in Lua 5]
```

The returned table is zero indexed.

```

void imAnalyzeMeasurePerimArea ( const imImage * image,
                                float *       perimarea
                                )

```

Measure the polygonal area limited by the perimeter line of all regions. Holes are not included.

Notice that some regions may have polygonal area zero.

Source image is IM_GRAY/IM_USHORT type (the result of [imAnalyzeFindRegions](#)).

perimarea has size the number of regions.

```
im.AnalyzeMeasurePerimArea(image: imImage, [region_count: number]) -> perimarea: table of numbers [in Lua 5]
```

The returned table is zero indexed.

```

void imAnalyzeMeasureCentroid ( const imImage * image,
                                const int *   area,
                                int            region_count,
                                float *       cx,
                                float *       cy
                                )

```

Calculate the centroid position of all regions. Holes are not included.

Source image is IM_GRAY/IM_USHORT type (the result of [imAnalyzeFindRegions](#)).

area, cx and cy have size the number of regions. If area is NULL will be internally calculated.

```
im.AnalyzeMeasureCentroid(image: imImage, [area: table of numbers], [region_count: number]) -> cx: table of n
```

The returned tables are zero indexed.

```
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
                                   )
```

Calculate the principal major axis slope of all regions.

Source image is IM_GRAY/IM_USHORT type (the result of [imAnalyzeFindRegions](#)).

data has size the number of regions. If area or centroid are NULL will be internally calculated.

Principal (major and minor) axes are defined to be those axes that pass through the centroid, about which the moment of inertia of the region is, respectively maximal or minimal.

```
im.AnalyzeMeasurePrincipalAxis(image: imImage, [area: table of numbers], [cx: table of numbers], [cy: table of numbers]
-> major_slope: table of numbers, major_length: table of numbers, minor_slope: table of numbers, minor_length: table of numbers)
```

The returned tables are zero indexed.

```
void imAnalyzeMeasureHoles ( const imImage * image,
                             int          connect,
                             int *       holes_count,
                             int *       area,
                             float *     perim
                           )
```

Measure the number and area of holes of all regions.

Source image is IM_USHORT type (the result of [imAnalyzeFindRegions](#)).

area and perim has size the number of regions, if some is NULL it will be not calculated.

```
im.AnalyzeMeasureHoles(image: imImage, connect: number, [region_count: number]) -> holes_count: number, area: table of numbers, perim: table of numbers
```

The returned tables are zero indexed.

```
void imAnalyzeMeasurePerimeter ( const imImage * image,
                                 float *        perim,
                                 int          region_count
                               )
```

Measure the total perimeter of all regions (external and internal).

Source image is IM_GRAY/IM_USHORT type (the result of [imAnalyzeFindRegions](#)).

It uses a half-pixel inter distance for 8 neighbors in a perimeter of a 4 connected 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 numbers [in Lua 5]
```

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

Isolates the perimeter line of gray integer images. Background is defined as being black (0).

It just checks if at least one of the 4 connected neighbors is non zero. Image borders are extended with zeros.

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

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

```
void imProcessRemoveByArea ( const imImage * src_image,
                             imImage *      dst_image,
                             int          connect,
                             int          start_size,
                             int          end_size,
                             int          inside
                           )
```

Eliminates regions that have area size outside or inside the given interval.

Source and destiny are a binary images. Regions can be 4 connected or 8 connected.

Can be done in-place. end_size can be zero to indicate no upper limit or an area with width*height size.

When searching inside the region the limits are inclusive (\leq size \geq), when searching outside the limits are exclusive ($>$ size $<$).

```
im.ProcessRemoveByArea(src_image: imImage, dst_image: imImage, connect: number, start_size: number, end_size:
im.ProcessRemoveByAreaNew(image: imImage, connect: number, start_size: number, end_size: number, inside: bool
```

```
void imProcessFillHoles ( const imImage * src_image,
                        imImage * dst_image,
                        int connect
                        )
```

Fill holes inside white regions.

Source and destiny are a binary images. Regions can be 4 connected or 8 connected.

Can be done in-place.

```
im.ProcessFillHoles(src_image: imImage, dst_image: imImage, connect: number) [in Lua 5]
im.ProcessFillHolesNew(image: imImage, connect: number) -> new_image: imImage [in Lua 5]
```



Generated on Thu Oct 1 11:40:07 2009 for IM by 1.6.1

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](#)
 - [ECW - ECW JPEG 2000](#)
 - [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](#)
 - [Kernel Generators](#)
 - [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](#)

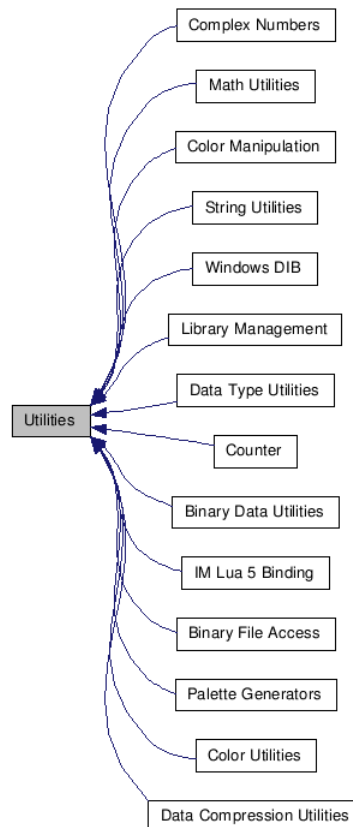
Generated on Thu Oct 1 11:40:08 2009 for IM by



1.6.1

Utilities

Collaboration diagram for Utilities:



Data Structures

- class [imAttribTable](#)
 Attributes Table. [More...](#)
- class [imAttribArray](#)
 Attributes Table. [More...](#)

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](#)

Defines

```
#define IM_MIN(_a, _b)  (_a < _b? _a: _b)
#define IM_MAX(_a, _b)  (_a > _b? _a: _b)
```

Detailed Description

See [im_util.h](#)

 1.6.1
 Generated on Thu Oct 1 11:40:08 2009 for IM by

Library Management

[\[Utilities\]](#)

Collaboration diagram for Library Management:



Defines

```
#define IM_AUTHOR    "Antonio Scuri"
#define IM_COPYRIGHT "Copyright (C) 1994-2009 Tecgraf, PUC-Rio."
#define IM_VERSION   "3.5"
#define IM_VERSION_NUMBER 305000
#define IM_VERSION_DATE "2009/10/02"
#define IM_DESCRIPTION "Image Representation, Storage, Capture and Processing"
#define IM_NAME       "IM - An Imaging Toolkit"
```

Functions

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

Detailed Description

Usefull definitions for about dialogs and for comparing the compiled version with the linked version of the library.

```
im._AUTHOR [in Lua 5]
im._COPYRIGHT [in Lua 5]
im._VERSION [in Lua 5]
im._VERSION_DATE [in Lua 5]
im._VERSION_NUMBER [in Lua 5]
im._DESCRIPTION [in Lua 5]
im._NAME [in Lua 5]
```

See [im_lib.h](#)

Function Documentation

```
const char* imVersion ( void )
```

Returns the library current version. Returns the definition IM_VERSION plus the bug fix number.

```
im.Version() -> version: string [in Lua 5]
```

```
const char* imVersionDate ( void )
```

Returns the library current version release date. Returns the definition IM_VERSION_DATE.

```
im.VersionDate() -> date: string [in Lua 5]
```

```
int imVersionNumber ( void )
```

Returns the library current version number. Returns the definition IM_VERSION_NUMBER plus the bug fix number. Can be compared in run time with IM_VERSION_NUMBER to compare compiled and linked versions of the library.

```
im.VersionNumber() -> version: number [in Lua 5]
```



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

IM Lua 5 Binding

[Utilities]

Collaboration diagram for IM Lua 5 Binding:



Functions

```

int imlua\_open (lua_State *L)
void imlua\_pushimage (lua_State *L, imImage *image)
imImage * imlua\_checkimage (lua_State *L, int param)
int imlua\_open\_capture (lua_State *L)
int imlua\_open\_process (lua_State *L)
int imlua\_open\_fftw (lua_State *L)
  
```

Detailed Description

Binding for the Lua 5 scripting language.
 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](#)

Function Documentation

```
int imlua_open ( lua_State * L )
```

Initializes the Lua binding of the main IM library.
 Returns 1 (leaves the "im" table on the top of the stack). You must link the application with the "imlua51" library.

```

void imlua_pushimage ( lua_State * L,
                      imImage * image
                    )
  
```

Pushes an image as a metatable on the stack.

```

imImage* imlua_checkimage ( lua_State * L,
                           int      param
                         )
  
```

Gets an image as a metatable from the stack, checks for correct type.

```
int imlua_open_capture ( lua_State * L )
```

Initializes the Lua binding of the capture library.
 Returns 1 (leaves the "im" table on the top of the stack). You must link the application with the "imlua_capture51" library.

```
int imlua_open_process ( lua_State * L )
```

Initializes the Lua binding of the process library.

Returns 1 (leaves the "im" table on the top of the stack). You must link the application with the "imlua_process51" library.

```
int imlua_open_fftw ( lua_State * L )
```

Initializes the Lua binding of the fourier transform library.

Returns 1 (leaves the "im" table on the top of the stack). You must link the application with the "imlua_fftw51" library.

Generated on Thu Oct 1 11:40:08 2009 for IM by



1.6.1

Color Utilities

[Utilities]

Collaboration diagram for Color Utilities:



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)

Detailed Description

See [im_util.h](#)

Function Documentation

```
long imColorEncode ( unsigned char red,
                    unsigned char green,
                    unsigned char blue
                    )
```

Encode RGB components in a long for palette usage.
"long" definition is compatible with the CD library definition.

```
im.ColorEncode(red: number, green: number, blue: number) -> color: lightuserdata [in Lua 5]
```

```
void imColorDecode ( unsigned char * red,
                    unsigned char * green,
                    unsigned char * blue,
                    long color
                    )
```

Decode RGB components from a long for palette usage.
"long" definition is compatible with the CD library definition.

```
im.ColorDecode(color: lightuserdata) -> red: number, green: number, blue: number [in Lua 5]
```

Generated on Thu Oct 1 11:40:08 2009 for IM by



1.6.1

Color Manipulation

[Utilities]

Collaboration diagram for Color Manipulation:



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)

```

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 aproximates the perception of brightness. It is nearly perceptual uniform. It has a range of 0 to 100.

Y' is luma, a nonlinear luminance that aproximates 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 distante 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)

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.

```
00077 {
00078     float zero[] = {128.0f, 32768.0f, 8388608.0f, 0.5f};
00079     return zero[data_type];
00080 }
```

int imColorMax (int *data_type*) [inline]

Returns the maximum value for color conversion purposes.

```
00085 {
00086     int max[] = {255, 65535, 16777215, 1};
00087     return max[data_type];
00088 }
```

template<class T>

T imColorQuantize (const float & *value*,
 const T & *max*
) [inline]

Quantize r=0-1 values into q=0-max. *max* is the maximum value. *max* and the returned value are usually integers, but the dummy quantizer uses real values. See also [Math Utilities](#).

```
00098 {
00099     if (max == 1) return (T)value; // to allow a dummy quantizer
00100     if (value >= 1) return max;
00101     if (value <= 0) return 0;
00102     /* return (T)imRound(value*(max + 1) - 0.5f); not necessary since all values are positive */
00103     return (T)(value*(max + 1));
00104 }
```

template<class T>

float imColorReconstruct (const T & *value*,
 const T & *max*
) [inline]

Reconstruct 0-max values into 0-1.
max is the maximum value. *max* and the given value are usually integers, but the dummy reconstructor uses real values. See also [Math Utilities](#).

```
00114 {
00115     if (max == 1) return (float)value; // to allow a dummy reconstructor
00116     if (value <= 0) return 0;
00117     if (value >= max) return 1;
00118     return (((float)value + 0.5f)/((float)max + 1.0f));
00119 }
```

template<class T>

void imColorYCbCr2RGB (const T *Y*,
 const T *Cb*,
 const T *Cr*,
 T & *R*,
 T & *G*,
 T & *B*,
 const T & *zero*,
 const T & *max*
) [inline]

Converts Y'CbCr to R'G'B' (all nonlinear).
ITU-R Recommendation 601-1 with no headroom/footroom.

0 <= Y <= 1 ; -0.5 <= CbCr <= 0.5 ; 0 <= RGB <= 1

R' = Y' + 0.000 *Cb + 1.402 *Cr
G' = Y' - 0.344 *Cb - 0.714 *Cr
B' = Y' + 1.772 *Cb + 0.000 *Cr

```
00135 {
00136     float r = float(Y + 1.402f * (Cr - zero));
00137     float g = float(Y - 0.344f * (Cb - zero) - 0.714f * (Cr - zero));
```

```

00138     float b = float(Y + 1.772f * (Cb - zero));
00139
00140     // now we should enforce 0<= rgb <= max
00141
00142     R = (T)IM_CROPMAX(r, max);
00143     G = (T)IM_CROPMAX(g, max);
00144     B = (T)IM_CROPMAX(b, max);
00145 }

```

```

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'

00161 {
00162     Y = (T)( 0.299f *R + 0.587f *G + 0.114f *B);
00163     Cb = (T)(-0.169f *R - 0.331f *G + 0.500f *B + (float)zero);
00164     Cr = (T)( 0.500f *R - 0.419f *G - 0.081f *B + (float)zero);
00165
00166     // there is no need for cropping here, YCrCr is already at the limits
00167 }

```

```

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)

00182 {
00183     T W = max - K;
00184     R = (T)((W * (max - C)) / max);
00185     G = (T)((W * (max - M)) / max);
00186     B = (T)((W * (max - Y)) / max);
00187
00188     // there is no need for cropping here, RGB is already at the limits
00189 }

```

```

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
```

```
00204 {
00205     float r = 3.2406f *X - 1.5372f *Y - 0.4986f *Z;
00206     float g = -0.9689f *X + 1.8758f *Y + 0.0415f *Z;
00207     float b = 0.0557f *X - 0.2040f *Y + 1.0570f *Z;
00208
00209     // we need to crop because not all XYZ colors are visible
00210
00211     R = (T)IM_CROPMAX(r, max);
00212     G = (T)IM_CROPMAX(g, max);
00213     B = (T)IM_CROPMAX(b, max);
00214 }
```

```
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
```

```
00229 {
00230     X = (T)(0.4124f *R + 0.3576f *G + 0.1805f *B);
00231     Y = (T)(0.2126f *R + 0.7152f *G + 0.0722f *B);
00232     Z = (T)(0.0193f *R + 0.1192f *G + 0.9505f *B);
00233
00234     // there is no need for cropping here, XYZ is already at the limits
00235 }
```

```
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)
```

```
00261 {
00262     float fX = X / 0.9505f; // white point D65
00263     float fY = Y / 1.0f;
00264     float fZ = Z / 1.0890f;
00265
00266     fX = IM_FWLAB(fX);
00267     fY = IM_FWLAB(fY);
00268     fZ = IM_FWLAB(fZ);
00269
00270     L = 1.16f * fY - 0.16f;
00271     a = 2.5f * (fX - fY);
00272     b = (fY - fZ);
```



```
00273 }
```

```
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 <= L <= 1 ; -0.5 <= ab <= +0.5 ; 0 <= XYZ <= 1

```
00286 {
00287     float fY = (L + 0.16f) / 1.16f;
00288     float gY = IM_GWLAB(fY);
00289
00290     float fgY = IM_FWLAB(gY);
00291     float gX = fgY + a / 2.5f;
00292     float gZ = fgY - b;
00293     gX = IM_GWLAB(gX);
00294     gZ = IM_GWLAB(gZ);
00295
00296     X = gX * 0.9505f;      // white point D65
00297     Y = gY * 1.0f;
00298     Z = gZ * 1.0890f;
00299 }
```

```
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)

00326 {
00327     float XYZ = (float)(X + 15 * Y + 3 * Z);
00328     float fY = Y / 1.0f;
00329
00330     if (XYZ != 0)
00331     {
00332         L = 1.16f * IM_FWLAB(fY) - 0.16f;
00333         u = 6.5f * L * ((4 * X)/XYZ - 0.1978f);
00334         v = 6.5f * L * ((9 * Y)/XYZ - 0.4683f);
00335     }
00336     else
00337     {
00338         L = u = v = 0;
00339     }
00340 }
```

```
void imColorLuv2XYZ ( const float L,
                      const float u,
                      const float v,
                      float &   X,
```

```

        float &    Y,
        float &    Z
    )                [inline]

```

Converts CIE L*u*v* (nonlinear) to CIE XYZ (linear).

The white point is D65. $0 \leq L \leq 1$; $-0.5 \leq uv \leq +0.5$; $0 \leq XYZ \leq 1$

```

00349 {
00350     float fY = (L + 0.16f) / 1.16f;
00351     Y = IM_GWLAB(fY) * 1.0f;
00352
00353     float ul = 0.1978f, vl = 0.4683f;
00354     if (L != 0)
00355     {
00356         ul = u / (6.5f * L) + 0.1978f;
00357         vl = v / (6.5f * L) + 0.4683f;
00358     }
00359
00360     X = ((9 * ul) / (4 * vl)) * Y;
00361     Z = ((12 - 3 * ul - 20 * vl) / (4 * vl)) * Y;
00362 }

```

```
float imColorTransfer2Linear ( const float & nonlinear_value ) [inline]
```

Converts nonlinear values to linear values.

We use the sRGB transfer function. sRGB uses ITU-R 709 primaries and D65 white point.

```

    0 <= 1 <= 1 ; 0 <= v <= 1

    if (v < 0.03928)
        l = v / 12.92
    else
        l = pow((v + 0.055) / 1.055, 2.4)

00376 {
00377     if (nonlinear_value < 0.03928f)
00378         return nonlinear_value / 12.92f;
00379     else
00380         return powf((nonlinear_value + 0.055f) / 1.055f, 2.4f);
00381 }

```

```
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 <= 1 <= 1 ; 0 <= v <= 1

    if (l < 0.0031308)
        v = 12.92 * l
    else
        v = 1.055 * pow(l, 1/2.4) - 0.055

```

Referenced by [imColorRGB2RGBNonlinear\(\)](#).

```

00395 {
00396     if (value < 0.0031308f)
00397         return 12.92f * value;
00398     else
00399         return 1.055f * powf(value, 1.0f/2.4f) - 0.055f;
00400 }

```

```

void imColorRGB2RGBNonlinear ( const float  RL,
                               const float  GL,
                               const float  BL,
                               float &      R,
                               float &      G,
                               float &      B
                               )                [inline]

```

Converts RGB (linear) to R'G'B' (nonlinear).

References [imColorTransfer2Nonlinear\(\)](#).

```

00406 {
00407     R = imColorTransfer2Nonlinear(RL);
00408     G = imColorTransfer2Nonlinear(GL);
00409     B = imColorTransfer2Nonlinear(BL);
00410 }

```

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'

00419 {
00420     return (T)((299 * R + 587 * G + 114 * B) / 1000);
00421 }

```

```
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

00437 {
00438     return 1.16f * IM_FWLAB(Y) - 0.16f;
00439 }

```

```
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)

00455 {
00456     float fY = (L + 0.16f) / 1.16f;
00457     return IM_GWLAB(fY);
00458 }

```

Generated on Thu Oct 1 11:40:04 2009 for IM by

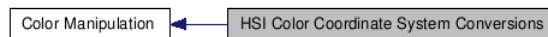


1.6.1

HSI Color Coordinate System Conversions

[[Color Manipulation](#)]

Collaboration diagram for HSI Color Coordinate System Conversions:



Functions

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)

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 1.

"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.

See [im_colorhsi.h](#)

Function Documentation

```
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 b,
                          float *        h,
                          float *        s,
                          float *        i
                        )
```

Converts from RGB (byte) to HSI.

```
void imColorHSI2RGB ( float   h,
                     float   s,
                     float   i,
                     float *  r,
                     float *  g,
                     float *  b
                   )
```

Converts from HSI to RGB.

```
void imColorHSI2RGBbyte ( float   h,
                          float   s,
                          float   i,
                          unsigned char * r,
                          unsigned char * g,
                          unsigned char * b
                        )
```

Converts from HSI to RGB (byte).

Generated on Thu Oct 1 11:40:04 2009 for IM by



1.6.1

Palette Generators

[\[Utilities\]](#)

Collaboration diagram for Palette Generators:



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)

```

Detailed Description

Creates several standard palettes. The palette is just an array of encoded color values. See also [Color Utilities](#).

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. The IM palette metatable name is "imPalette".

When converted to a string will return "imPaleta(%p)" where p is replaced by the userdata address. If the palette is already destroyed by `im.PaletteDestroy`, then it will return also the suffix "-destroyed".

In Lua, to destroy a palette you can call `im.PaletteDestroy`. If this function is not called, the palette is destroyed by the garbage collector.

```
im.PaletteDestroy(pal: imPalette) [in Lua 5]
```

In Lua, array access is enabled so you can do:.

```

color = pal[index]

pal[index] = color

count = #pal

```

See [im_palette.h](#)

Function Documentation

```

int imPaletteFindNearest ( const long * palette,
                          int          palette_count,
                          long         color
                          )

```

Searches for the nearest color on the table and returns the color index if successful. It looks in all palette entries and finds the minimum euclidian square distance. If the color matches the given color it returns immediately. See also [Color Utilities](#).

```
im.PaletteFindNearest(pal: imPalette, color: lightuserdata) -> index: number [in Lua 5]
```

```

int imPaletteFindColor ( const long * palette,
                        int          palette_count,
                        long         color,
                        unsigned char tol
                        )

```

Searches for the color on the table and returns the color index if successful. If the tolerance is 0 search for the exact match in the palette else search for the first color that fits in the tolerance range. See also [Color Utilities](#).

```
im.PaletteFindColor(pal: imPalette, color: lightuserdata, tol: number) -> index: number [in Lua 5]
```

```
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 trough 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  color,
                                     int    x,
                                     int    y
                                   )
```

Returns the index of the correspondent RGB color of an uniform palette. Uses an 8x8 ordered dither to lookup the index in a halftone matrix. The spatial position used by the halftone method.

```
im.PaletteUniformIndexHalftoned(color: lightuserdata, x: number, y: number) -> index: number [in Lua 5]
```



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

Binary Data Utilities

[Utilities]

Collaboration diagram for Binary Data Utilities:



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)
```

Detailed Description

See [im_util.h](#)

Enumeration Type Documentation

```
enum imByteOrder
```

CPU Byte Orders.

Enumerator:

IM_LITTLEENDIAN Little Endian - The most significant byte is on the right end of a word. Used by Intel processors.

IM_BIGENDIAN Big Endian - The most significant byte is on the left end of a word. Used by Motorola processors, also is the network standard byte order.

```
00211 {
00212     IM\_LITTLEENDIAN, /**< Little Endian - The most significant byte is on the right end of a word. Used b
00213     IM\_BIGENDIAN      /**< Big Endian - The most significant byte is on the left end of a word. Used by Mo
00214 };
```

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



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

Complex Numbers

[Utilities]

Collaboration diagram for Complex Numbers:



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)
int operator< (const imcfloat &C1, const imcfloat &C2)
int operator< (const imcfloat &C, const float &F)
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)
```

Detailed Description

See [im_complex.h](#)

Complex numbers operators.



Generated on Thu Oct 1 11:40:04 2009 for IM by 1.6.1

Data Type Utilities

[Utilities]

Collaboration diagram for Data Type Utilities:



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)

```

Detailed Description

See [im_util.h](#)

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]
```



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

Math Utilities

[Utilities]

Collaboration diagram for Math Utilities:



Functions

```

int imRound (float x)
int imResample (int x, float factor)
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 x1, float y1)
template<class T, class TU>
T imBicubicInterpolation (int width, int height, T *map, float x1, float y1, TU Dummy)
template<class T>
void imMinMax (const T *map, int count, T &min, T &max)

```

Detailed Description

When converting between continuous and discrete use:
Continuous = Discrete + 0.5 [Reconstruction/Interpolation]
Discrete = Round(Continuous - 0.5) [Sampling/Quantization]

Notice that must check 0-max limits when converting from Continuous to Discrete.

When converting between discrete and discrete use:
integer src_size, dst_len, src_i, dst_i
real factor = (real)(dst_size)/(real)(src_size)
dst_i = Round(factor*(src_i + 0.5) - 0.5)

See [im_math.h](#)

Function Documentation

```
int imRound (float x ) [inline]
```

Round a real to the nearest integer.

Referenced by [imZeroOrderInterpolation\(\)](#).

```

00055 {
00056     return (int)(x < 0? x-0.5f: x+0.5f);
00057 }

```

```

int imResample (int x,
                float factor
                ) [inline]

```

Converts between two discrete grids. factor is "dst_size/src_size".

```

00067 {
00068     float xr = factor*(x + 0.5f) - 0.5f;
00069     return (int)(xr < 0? xr-0.5f: xr+0.5f); /* Round */
00070 }

```

```

template<class T, class TU>
T imZeroOrderDecimation (int width,
                        int height,
                        T * map,
                        float x1,
                        float y1,
                        float box_width,
                        float box_height,
                        TU Dummy
                        ) [inline]

```

Does Zero Order Decimation (Mean).

```

00076 {
00077     int x0,x1,y0,y1;
00078     (void)Dummy;
00079
00080     x0 = (int)floor(x1 - box_width/2.0 - 0.5) + 1;
00081     y0 = (int)floor(y1 - box_height/2.0 - 0.5) + 1;
00082     x1 = (int)floor(x1 + box_width/2.0 - 0.5);
00083     y1 = (int)floor(y1 + box_height/2.0 - 0.5);
00084
00085     if (x0 == x1) x1++;
00086     if (y0 == y1) y1++;
00087
00088     x0 = x0<0? 0: x0>width-1? width-1: x0;
00089     y0 = y0<0? 0: y0>height-1? height-1: y0;
00090     x1 = x1<0? 0: x1>width-1? width-1: x1;
00091     y1 = y1<0? 0: y1>height-1? height-1: y1;
00092
00093     TU Value;

```

```

00094     int Count = 0;
00095
00096     Value = 0;
00097
00098     for (int y = y0; y <= y1; y++)
00099     {
00100         for (int x = x0; x <= x1; x++)
00101         {
00102             Value += map[y*width+x];
00103             Count++;
00104         }
00105     }
00106
00107     if (Count == 0)
00108     {
00109         Value = 0;
00110         return (T)Value;
00111     }
00112
00113     return (T)(Value/(float)Count);
00114 }

```

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
                        ) [inline]

```

Does Bilinear Decimation.

```

00120 {
00121     int x0,x1,y0,y1;
00122     (void)Dummy;
00123
00124     x0 = (int)floor(xl - box_width/2.0 - 0.5) + 1;
00125     y0 = (int)floor(yl - box_height/2.0 - 0.5) + 1;
00126     x1 = (int)floor(xl + box_width/2.0 - 0.5);
00127     y1 = (int)floor(yl + box_height/2.0 - 0.5);
00128
00129     if (x0 == x1) x1++;
00130     if (y0 == y1) y1++;
00131
00132     x0 = x0<0? 0: x0>width-1? width-1: x0;
00133     y0 = y0<0? 0: y0>height-1? height-1: y0;
00134     x1 = x1<0? 0: x1>width-1? width-1: x1;
00135     y1 = y1<0? 0: y1>height-1? height-1: y1;
00136
00137     TU Value, LineValue;
00138     float LineNorm, Norm, dxr, dyr;
00139
00140     Value = 0;
00141     Norm = 0;
00142
00143     for (int y = y0; y <= y1; y++)
00144     {
00145         dyr = y1 - (y+0.5f);
00146         if (dyr < 0) dyr *= -1;
00147
00148         LineValue = 0;
00149         LineNorm = 0;
00150
00151         for (int x = x0; x <= x1; x++)
00152         {
00153             dxr = x1 - (x+0.5f);
00154             if (dxr < 0) dxr *= -1;
00155
00156             LineValue += map[y*width+x] * dxr;
00157             LineNorm += dxr;
00158         }
00159
00160         Value += LineValue * dyr;
00161         Norm += dyr * LineNorm;
00162     }
00163
00164     if (Norm == 0)
00165     {
00166         Value = 0;
00167         return (T)Value;
00168     }

```

```

00169
00170     return (T)(Value/Norm);
00171 }

```

```

template<class T>
T imZeroOrderInterpolation ( int   width,
                             int   height,
                             T*    map,
                             float xl,
                             float yl
                             )           [inline]

```

Does Zero Order Interpolation (Nearest Neighborhood).

References [imRound\(\)](#).

```

00177 {
00178     int x0 = imRound(xl-0.5f);
00179     int y0 = imRound(yl-0.5f);
00180     x0 = x0<0? 0: x0>width-1? width-1: x0;
00181     y0 = y0<0? 0: y0>height-1? height-1: y0;
00182     return map[y0*width + x0];
00183 }

```

```

template<class T>
T imBilinearInterpolation ( int   width,
                             int   height,
                             T*    map,
                             float xl,
                             float yl
                             )           [inline]

```

Does Bilinear Interpolation.

```

00189 {
00190     int x0, y0, xl, yl;
00191     float t, u;
00192
00193     if (xl < 0.5)
00194     {
00195         xl = x0 = 0;
00196         t = 0;
00197     }
00198     else if (xl > width-0.5)
00199     {
00200         xl = x0 = width-1;
00201         t = 0;
00202     }
00203     else
00204     {
00205         x0 = (int)(xl-0.5f);
00206         xl = x0+1;
00207         t = xl - (x0+0.5f);
00208     }
00209
00210     if (yl < 0.5)
00211     {
00212         yl = y0 = 0;
00213         u = 0;
00214     }
00215     else if (yl > height-0.5)
00216     {
00217         yl = y0 = height-1;
00218         u = 0;
00219     }
00220     else
00221     {
00222         y0 = (int)(yl-0.5f);
00223         yl = y0+1;
00224         u = yl - (y0+0.5f);
00225     }
00226
00227     T fll = map[y0*width + x0];
00228     T fh1 = map[y0*width + xl];
00229     T flh = map[yl*width + x0];
00230     T fhh = map[yl*width + xl];
00231
00232     return (T)((fhh - flh - fh1 + fll) * u * t +
00233               (fh1 - fll) * t +
00234               (flh - fll) * u +
00235               fll);
00236 }

```

```

template<class T, class TU >
T imBicubicInterpolation ( int   width,
                           int   height,
                           T*    map,
                           float xl,
                           float yl,
                           TU    Dummy
                           ) [inline]

```

Does Bicubic Interpolation.

```

00242 {
00243     int X[4], Y[4];
00244     float t, u;
00245     (void)Dummy;
00246
00247     if (xl > width-0.5)
00248     {
00249         X[3] = X[2] = X[1] = width-1;
00250         X[0] = X[1]-1;
00251         t = 0;
00252     }
00253     else
00254     {
00255         X[1] = (int)(xl-0.5f);
00256         if (X[1] < 0) X[1] = 0;
00257
00258         X[0] = X[1]-1;
00259         X[2] = X[1]+1;
00260         X[3] = X[1]+2;
00261
00262         if (X[0] < 0) X[0] = 0;
00263         if (X[3] > width-1) X[3] = width-1;
00264
00265         t = xl - (X[1]+0.5f);
00266     }
00267
00268     if (yl > height-0.5)
00269     {
00270         Y[3] = Y[2] = Y[1] = height-1;
00271         Y[0] = Y[1]-1;
00272         u = 0;
00273     }
00274     else
00275     {
00276         Y[1] = (int)(yl-0.5f);
00277         if (Y[1] < 0) Y[1] = 0;
00278
00279         Y[0] = Y[1]-1;
00280         Y[2] = Y[1]+1;
00281         Y[3] = Y[1]+2;
00282
00283         if (Y[0] < 0) Y[0] = 0;
00284         if (Y[3] > height-1) Y[3] = height-1;
00285
00286         u = yl - (Y[1]+0.5f);
00287     }
00288
00289     float CX[4], CY[4];
00290
00291     // Optimize calculations
00292     {
00293         float c, c2, c3;
00294
00295         #define C0 (-c3 + 2.0f*c2 - c)
00296         #define C1 ( c3 - 2.0f*c2 + 1.0f)
00297         #define C2 (-c3 + c2 + c)
00298         #define C3 ( c3 - c2)
00299
00300         c = t;
00301         c2 = c*c; c3 = c2*c;
00302         CX[0] = C0; CX[1] = C1; CX[2] = C2; CX[3] = C3;
00303
00304         c = u;
00305         c2 = c*c; c3 = c2*c;
00306         CY[0] = C0; CY[1] = C1; CY[2] = C2; CY[3] = C3;
00307
00308         #undef C0
00309         #undef C1
00310         #undef C2
00311         #undef C3
00312     }
00313
00314     TU LineValue, Value;
00315     float LineNorm, Norm;

```

```

00316
00317     Value = 0;
00318     Norm = 0;
00319
00320     for (int y = 0; y < 4; y++)
00321     {
00322         LineValue = 0;
00323         LineNorm = 0;
00324
00325         for (int x = 0; x < 4; x++)
00326         {
00327             LineValue += map[Y[y]*width+X[x]] * CX[x];
00328             LineNorm += CX[x];
00329         }
00330
00331         Value += LineValue * CY[y];
00332         Norm += CY[y] * LineNorm;
00333     }
00334
00335     if (Norm == 0)
00336     {
00337         Value = 0;
00338         return (T)Value;
00339     }
00340
00341     Value = (Value/Norm);
00342
00343     int size = sizeof(T);
00344     if (size == 1)
00345         return (T) (Value<=(TU)0? (TU)0: Value<=(TU)255? Value: (TU)255);
00346     else
00347         return (T) (Value);
00348 }

```

```

template<class T>
void imMinMax ( const T* map,
                int      count,
                T &      min,
                T &      max
                )          [inline]

```

Calculates minimum and maximum values.

```

00354 {
00355     min = *map++;
00356     max = min;
00357     for (int i = 1; i < count; i++)
00358     {
00359         T value = *map++;
00360
00361         if (value > max)
00362             max = value;
00363         else if (value < min)
00364             min = value;
00365     }
00366 }

```



Generated on Thu Oct 1 11:40:06 2009 for IM by 1.6.1

String Utilities

[\[Utilities\]](#)

Collaboration diagram for String Utilities:



Functions

int [imStrEqual](#) (const char *str1, const char *str2)
int [imStrNLen](#) (const char *str, int max_len)
int [imStrCheck](#) (const void *data, int count)

Detailed Description

See [im_util.h](#)

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.



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

Binary File Access

[Utilities]

Collaboration diagram for Binary File Access:



Data Structures

```
struct imBinMemoryFileName
Memory File I/O Filename. More...
```

Typedefs

```
typedef struct 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,...)
int imBinFileReadInteger (imBinFile *handle, int *value)
int imBinFileReadFloat (imBinFile *handle, float *value)
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)
void imBinMemoryRelease (unsigned char *buffer)
```

Detailed Description

These functions are very usefull for reading/writing binary files that have headers or data that have to be converted depending on the current CPU byte order. It can invert 2, 4 or 8 bytes numbers to/from little/big-endian orders.

It will process the data only if the file format is diferent 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](#)

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:

- IM_RAWFILE* System dependent file I/O Rotines.
- IM_STREAM* Standard Ansi C Stream I/O Rotines.
- IM_MEMFILE* Uses a memory buffer (see [imBinMemoryFileName](#)).
- IM_SUBFILE* It is a sub file. FileName is a imBinFile* pointer from any other module.
- IM_FILEHANDLE* System dependent file I/O Rotines, but FileName is a system file handle ("int" in UNIX and "HANDLE" in Windows).
- IM_IOCUSTOM0* Other registered modules starts from here.

```
00112 {
00113     IM\_RAWFILE,    /**< System dependent file I/O Rotines. */
00114     IM\_STREAM,    /**< Standard Ansi C Stream I/O Rotines. */
00115     IM\_MEMFILE,    /**< Uses a memory buffer (see \ref imBinMemoryFileName). */
00116     IM\_SUBFILE,    /**< It is a sub file. FileName is a imBinFile* pointer from any other module. */
00117     IM\_FILEHANDLE, /**< System dependent file I/O Rotines, but FileName is a system file handle ("int" in
00118     IM\_IOCUSTOM0 /**< Other registered modules starts from here. */
00119 };
```

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 * bfile,
                             void *      pValues,
                             unsigned long pCount,
                             int          pSizeOf
                             )
```

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 * bfile,
                             void *      pValues,
                             unsigned long pCount,
                             int          pSizeOf
                             )

```

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 diferent from the CPU byte order.

```

unsigned long imBinFilePrintf ( imBinFile * bfile,
                              char *      format,
                              ...
                              )

```

Writes a string without the NULL terminator. The function uses sprintf to compose the string.
 The internal buffer is fixed at 4096 bytes.

```

int imBinFileReadInteger ( imBinFile * handle,
                          int *       value
                          )

```

Reads an integer number from the current position until found a non integer character. Returns a non zero value if sucessfull.

```

int imBinFileReadFloat ( imBinFile * handle,
                        float *      value
                        )

```

Reads an floating point number from the current position until found a non number character. Returns a non zero value if sucessfull.

```

void imBinFileSeekTo ( imBinFile * bfile,
                     unsigned long pOffset
                     )

```

Moves the file pointer from the begining of the file.
 When writing to a file seeking can go beyond the end of the file.

```

void imBinFileSeekOffset ( imBinFile * bfile,
                          long        pOffset
                          )

```

Moves the file pointer from current position.
 If the offset is a negative value the pointer moves backwards.

```

void imBinFileSeekFrom ( imBinFile * bfile,
                        long        pOffset
                        )

```

Moves the file pointer from the end of the file.
 The offset is usually a negative value.

```

unsigned long imBinFileTell ( imBinFile * bfile )

```

Returns the current offset position.

```

int imBinFileEndOfFile ( imBinFile * bfile )

```

Indicates that the file pointer is at the end of the file.

```

int imBinFileSetCurrentModule ( int pModule )

```

Sets the current I/O module.

Returns:

the previous function set, or -1 if failed. See also [imBinFileModule](#).

```

void imBinMemoryRelease ( unsigned char * buffer )

```

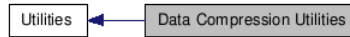
Release the internal memory allocated when writing a Memory File (see [imBinMemoryFileName](#)).



Data Compression Utilities

[Utilities]

Collaboration diagram for Data Compression Utilities:



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)

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 3.5.

<http://software.schmorp.de/pkg/liblzf>

Copyright (C) 2000-2009 Marc Alexander Lehmann See [im_util.h](#)

Function Documentation

```
int imCompressDataZ ( const void * src_data,
                     int          src_size,
                     void *      dst_data,
                     int          dst_size,
                     int          zip_quality
                     )
```

Compresses the data using the ZLIB Deflate compression.
 The destination buffer must be at least 0.1% larger than source_size plus 12 bytes.
 It compresses raw byte data. zip_quality can be 1 to 9.
 Returns the size of the compressed buffer or zero if failed.

```
int imCompressDataUnZ ( const void * src_data,
                       int          src_size,
                       void *      dst_data,
                       int          dst_size
                       )
```

Uncompresses the data compressed with the ZLIB Deflate compression.
 Returns zero if failed.

```
int imCompressDataLZF ( const void * src_data,
                       int          src_size,
                       void *      dst_data,
                       int          dst_size,
                       int          zip_quality
                       )
```

Compresses the data using the libLZF compression.
 Returns the size of the compressed buffer or zero if failed.

```
int imCompressDataUnLZF ( const void * src_data,
                          int          src_size,
                          void *      dst_data,
                          int          dst_size
                          )
```

Uncompresses the data compressed with the libLZF compression. Returns zero if failed.



Counter

[Utilities]

Collaboration diagram for Counter:



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)
    int imCounterIncTo (int counter, int count)
    void imCounterTotal (int counter, int total, const char *message)
```

Detailed Description

Used to notify the application that a step in the loading, saving or processing operation has been performed.

See [im_counter.h](#)

Typedef Documentation

```
typedef int(* imCounterCallback )(int counter, void *user_data, const char *text, int progress)
```

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 *          user_data,
                                         imCounterCallback counter_func
                                         )
```

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.

```
int imCounterIncTo ( int counter,
                    int count
                    )
```

Set a specific 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,
                    const char * message
                  )
```

Sets the total increments of a count.



Generated on Thu Oct 1 11:40:04 2009 for IM by 1.6.1

Windows DIB

[Utilities]

Collaboration diagram for Windows DIB:



Data Structures

struct [imDib](#)
 Windows DIB Structure. [More...](#)

Typedefs

typedef struct
 [imDib](#) imDib
 typedef unsigned int
 (* [imDibLineGetPixel](#)) (unsigned char *line, int col)
 typedef void (* [imDibLineSetPixel](#)) (unsigned char *line, int col, unsigned int 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)

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](#)

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 int(* [imDibLineGetPixel](#))(unsigned char *line, int col)

DIB GetPixel function definition.
the DWORD is a raw copy of the bits, use (unsigned char*)&pixel

typedef void(* [imDibLineSetPixel](#))(unsigned char *line, int col, unsigned int pixel)

DIB SetPixel function definition

Function Documentation

```
imDib* imDibCreate ( int width,
                    int height,
                    int bpp
                    )
```

Creates a new DIB.
use bpp=-16/-32 to allocate space for BITFLIEDS.

```
imDib* imDibCreateCopy ( const imDib * dib )
```

Duplicates the DIB contents in a new DIB.

```
imDib* imDibCreateReference ( BYTE * bmi,
                             BYTE * bits
                             )
```

Creates a DIB using an already allocated memory.
"bmi" must be a pointer to BITMAPINFOHEADER.
"bits" can be NULL if it is inside "bmi" after the palette.

```
imDib* imDibCreateSection ( HDC          hDC,
                           HBITMAP * image,
                           int         width,
                           int         height,
                           int         bpp
                           )
```

Creates a DIB section for drawing porposes.
Returns the image handle also created.

```
void imDibDestroy ( imDib * dib )
```

Destroy the DIB

```
imDibLineGetPixel imDibLineGetPixelFunc ( int bpp )
```

Returns a function to read pixels from a DIB line.

[imDibLineSetPixel](#) imDibLineSetPixelFunc (int *bpp*)

Returns a function to write pixels into a DIB line.

[imDib](#)* imDibFromHBitmap (const HBITMAP *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 *          dib,
                        const unsigned char * map,
                        const long *         palette,
                        int                   palette_count
                    )

```

Converts an indexed image into a DIB. bpp must be <=8. colors is rgb packed (RGBRGBRGB...)

```

void imDibEncodeFromBitmap ( imDib *          dib,
                           const unsigned char * data
                        )

```

Converts a IM_RGB packed image, with or without alpha, into a DIB.

```

void imDibDecodeToBitmap ( const imDib *      dib,
                          unsigned char * data
                        )

```

Converts a DIB into IM_RGB packed image, with or without alpha.

Generated on Thu Oct 1 11:40:04 2009 for IM by



1.6.1

Data Structures

Here are the data structures with brief descriptions:

imBinMemoryFileName	Memory File I/O Filename
imDib	Windows DIB Structure
imFile	Image File Format Base Class (SDK Use Only)
imImage	ImImage Structure Definition
imStats	
imAttribArray	Attributes Table
imAttribTable	Attributes Table
imcfloat	Complex Float Data Type
imFileFormatBase	Image File Format Virtual Class (SDK Use Only)
imFormat	Image File Format Descriptor (SDK Use Only)
imImageFile	C++ Wrapper for the Image File Structure

Generated on Thu Oct 1 11:40:08 2009 for IM by



1.6.1

imAttribTable Class Reference

[Utilities]

Attributes Table. [More...](#)

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

```

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.

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.

```
00030      { ptable = imAttribTableCreate(hash_size); }
```

```
imAttribTable::~imAttribTable ( ) [inline]
```

Destroys the table and all the attributes.

```
00034      { imAttribTableDestroy(ptable); ptable = 0; }
```

Member Function Documentation

```
int imAttribTable::Count ( ) const [inline]
```

Returns the number of elements in the table.

```
00038      { return imAttribTableCount(ptable); }
```

```
void imAttribTable::RemoveAll ( ) [inline]
```

Removes all the attributes in the table

```
00042      { imAttribTableRemoveAll(ptable); }
```

```
void imAttribTable::CopyFrom ( const imAttribTable & table ) [inline]
```

Copies the contents of the given table into this table.

```
00046      { imAttribTableCopyFrom(ptable, table.ptable); }
```

```
void imAttribTable::Set ( const char * name,
                        int          data_type,
                        int          count,
                        const void * data
                        ) [inline]
```

Inserts an attribute into the table.

If *data_type* is BYTE then count can be -1 to indicate a NULL terminated string. Data is duplicated if not NULL, else data is initialized with zeros. See also [imDataType](#).

```
00053      { imAttribTableSet(ptable, name, data_type, count, data); }
```

```
void imAttribTable::UnSet ( const char * name ) [inline]
```

Removes an attribute from the table given its name.

```
00057      { 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. See also [imDataType](#).

```
00063      { 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.

```
00067      { imAttribTableForEach(ptable, user_data, attrib_func); }
```

The documentation for this class was generated from the following file:

- [im_attrb.h](#)



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

_imBinMemoryFileName Struct Reference

[[Binary File Access](#)]

Memory File I/O Filename. [More...](#)

Data Fields

```
unsigned char * buffer
               int size
               float reallocate
```

Detailed Description

Fake file name for the memory I/O module.

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. The buffer is allocated using "malloc", and reallocated using "realloc". Use "free" to release it. To avoid RTL conflicts use the function imBinMemoryRelease.

int [_imBinMemoryFileName::size](#)

Size of the buffer.

float [_imBinMemoryFileName::reallocate](#)

Reallocate factor for the memory buffer when writing (size += reallocate*size). Set reallocate to 0 to disable reallocation, in this case buffer must not be NULL.

The documentation for this struct was generated from the following file:

- [im_binfile.h](#)



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

_imDib Struct Reference

[[Windows DIB](#)]

Windows DIB Structure. [More...](#)

Data Fields

```
HGLOBAL handle
BYTE * dib
      int size
BITMAPINFO * bmi
BITMAPINFOHEADER * bmih
RGBQUAD * bmci
BYTE * bits
      int palette\_count
```

```

int bits\_size
int line\_size
int pad\_size
int is\_reference

```

Detailed Description

Handles a DIB in memory.

The DIB is stored in only one buffer. The secondary members are pointers to the main buffer.

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::bmci](#)

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](#)]

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

Inherited by [imFileFormatBase](#).

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

```

Detailed Description

Base container to hold format independent state variables.

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](#)
-

Generated on Thu Oct 1 11:40:08 2009 for IM by



1.6.1

imFormat Class Reference

[\[File Format SDK\]](#)

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

Public Member Functions

```
virtual imFileFormatBase
    * Create () const =0
virtual int CanWrite (const char *compression, int color_mode, int data_type) const =0
    imFormat (const char *_format, const char *_desc, const char *_ext, const char **_comp, int _comp_count, int
        _can_sequence)
```

Data Fields

```
const char * format
const char * desc
const char * ext
const char ** comp
int comp_count
int can_sequence
```

Detailed Description

All file formats must define these informations. They are stored by [imFormatRegister](#).

The documentation for this class was generated from the following file:

- [im_format.h](#)



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

imcfloat Class Reference

[[Complex Numbers](#)]

Complex Float Data Type. [More...](#)

Public Member Functions

```
imcfloat ()
imcfloat (const float &r, const float &i)
imcfloat (const float &r)
```

Data Fields

```
float real
float imag
```

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.

The documentation for this class was generated from the following file:

- [im_complex.h](#)



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

_imImage Struct Reference

[[imImage](#)]

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

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

```

Detailed Description

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

Field Documentation

int [_imImage::width](#)

Number of columns. image:Width() -> width: number [in Lua 5].

int [_imImage::height](#)

Number of lines. image:Height() -> height: number [in Lua 5].

int [_imImage::color_space](#)

Color space descriptor. See also [imColorSpace](#). image:ColorSpace() -> color_space: number [in Lua 5].

int [_imImage::data_type](#)

Data type descriptor. See also [imDataType](#). image:DataType() -> data_type: number [in Lua 5].

int [_imImage::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 imImageAddAlpha is called, this is done in image load functions.

int [_imImage::depth](#)

Number of planes (ColorSpaceDepth)

int [_imImage::line_size](#)

Number of bytes per line in one plane (width * DataTypeSize)

int [_imImage::plane_size](#)

Number of bytes per plane. (line_size * height)

int [_imImage::size](#)

Number of bytes occupied by the image (plane_size * depth)

int [_imImage::count](#)

Number of pixels (width * height)

void** [_imImage::data](#)

Image data organized as a 2D matrix with several planes.
But plane 0 is also a pointer to the full data.
The remaining planes are: data[i] = data[0] + i*plane_size
In Lua, data indexing is possible using: image[plane][row][column]

long* [_imImage::palette](#)

Color palette. image:GetPalette() -> palette: imPalette [in Lua 5].
Used when depth=1. Otherwise is NULL.

int [_imImage::palette_count](#)

The palette is always 256 colors allocated, but can have less colors used.

void* [_imImage::attrib_table](#)

in fact is an [imAttribTable](#), but we hide this here

The documentation for this struct was generated from the following file:

- [im_image.h](#)



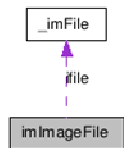
Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

imImageFile Class Reference

[Image Storage]

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

Collaboration diagram for imImageFile:



[\[legend\]](#)

Public Member Functions

```

imImageFile (const char *file_name)
imImageFile (const char *file_name, const char *format)
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)
  
```

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 cheked using the Failed() member.

The documentation for this class was generated from the following file:

- [im_plus.h](#)



Generated on Thu Oct 1 11:40:08 2009 for IM by 1.6.1

File List

Here is a list of all documented files with brief descriptions:

im.h [code]	Main API
im_attrib.h [code]	Attributes Table
im_attrib_flat.h [code]	Attributes Table Flat API. This will simplify the DLL export, and can be used for C applications
im_binfile.h [code]	Binary File Access
im_capture.h [code]	Video Capture
im_color.h [code]	Color Manipulation
im_colorhsi.h [code]	HSI Color Manipulation
im_complex.h [code]	Complex Data Type
im_convert.h [code]	Image Conversion
im_counter.h [code]	Processing Counter
im_dib.h [code]	Windows DIB (Device Independent Bitmap)
im_file.h [code]	File Access

im_format.h [code]	File Format Access
im_format_all.h [code]	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
im_format_avi.h [code]	Register the AVI Format
im_format_ecw.h [code]	Register the ECW Format
im_format_jp2.h [code]	Register the JP2 Format
im_format_raw.h [code]	Initialize the RAW Format Driver Header for internal use only
im_format_wmv.h [code]	Register the WMF Format
im_image.h [code]	Image Manipulation
im_kernel.h [code]	Kernel Generators Creates several known kernels
im_lib.h [code]	Library Management and Main Documentation
im_math.h [code]	Math Utilities
im_math_op.h [code]	Math Operations
im_palette.h [code]	Palette Generators
im_plus.h [code]	C++ Wrapper for File Access
im_process.h [code]	Image Processing
im_process_ana.h [code]	Image Statistics and Analysis
im_process_glo.h [code]	Image Processing - Global Operations
im_process_loc.h [code]	Image Processing - Local Operations
im_process_pon.h [code]	Image Processing - Pontual Operations
im_raw.h [code]	RAW File Format
im_util.h [code]	Utilities
imlua.h [code]	IM Lua 5 Binding
old_im.h [code]	Old API

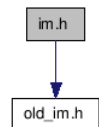


Generated on Thu Oct 1 11:40:09 2009 for IM by 1.6.1

im.h File Reference

Main API. [More...](#)

Include dependency graph for im.h:



This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Typedefs

typedef struct [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 * imFileOpenAs (const char *file_name, const char *format, int *error)
imFile * imFileNew (const char *file_name, const char *format, int *error)
void imFileClose (imFile *ifile)
void * imFileHandle (imFile *ifile, int index)
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 imFormatRegisterInternal (void)
void imFormatRemoveAll (void)
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)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:02 2009 for IM by

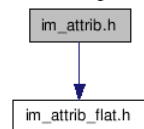


1.6.1

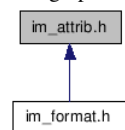
im_attrib.h File Reference

Attributes Table. [More...](#)

Include dependency graph for im_attrib.h:



This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Data Structures

class [imAttribTable](#)
 Attributes Table. [More...](#)
 class [imAttribArray](#)
 Attributes Table. [More...](#)

Detailed Description

See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:02 2009 for IM by

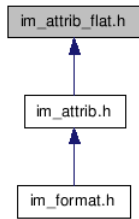


1.6.1

im_attrib_flat.h File Reference

Attributes Table Flat API. This will simplify the DLL export, and can be used for C applications. [More...](#)

This graph shows which files directly or indirectly include this file:



[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)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

Typedef Documentation

```
typedef int(* imAttribTableCallback )(void *user_data, int index, const char *name, int data_type, int count, const void *data)
```

Definition of the callback used in ForEach function.

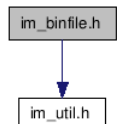


Generated on Thu Oct 1 11:40:02 2009 for IM by 1.6.1

im_binfile.h File Reference

Binary File Access. [More...](#)

Include dependency graph for im_binfile.h:



[Go to the source code of this file.](#)

Data Structures

```
struct \_imBinMemoryFileName
Memory File I/O Filename. More...
```

Typedefs

```

typedef struct _imBinFile imBinFile
typedef struct \_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,...)
int imBinFileReadInteger (imBinFile *handle, int *value)
int imBinFileReadFloat (imBinFile *handle, float *value)
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)
void imBinMemoryRelease (unsigned char *buffer)
```

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:02 2009 for IM by 1.6.1

im_capture.h File Reference

Video Capture. [More...](#)

[Go to the source code of this file.](#)

Typedefs

typedef struct _imVideoCapture **imVideoCapture**

Functions

```
int IM_DECL imVideoCaptureDeviceCount (void)
const char *IM_DECL imVideoCaptureDeviceDesc (int device)
const char *IM_DECL imVideoCaptureDeviceExDesc (int device)
const char *IM_DECL imVideoCaptureDevicePath (int device)
const char *IM_DECL imVideoCaptureDeviceVendorInfo (int device)
int IM_DECL imVideoCaptureReloadDevices (void)
imVideoCapture *IM_DECL imVideoCaptureCreate (void)
void IM_DECL imVideoCaptureDestroy (imVideoCapture *vc)
int IM_DECL imVideoCaptureConnect (imVideoCapture *vc, int device)
void IM_DECL imVideoCaptureDisconnect (imVideoCapture *vc)
int IM_DECL imVideoCaptureDialogCount (imVideoCapture *vc)
int IM_DECL imVideoCaptureShowDialog (imVideoCapture *vc, int dialog, void *parent)
int IM_DECL imVideoCaptureSetInOut (imVideoCapture *vc, int input, int output, int cross)
const char *IM_DECL imVideoCaptureDialogDesc (imVideoCapture *vc, int dialog)
int IM_DECL imVideoCaptureFormatCount (imVideoCapture *vc)
int IM_DECL imVideoCaptureGetFormat (imVideoCapture *vc, int format, int *width, int *height, char *desc)
int IM_DECL imVideoCaptureSetFormat (imVideoCapture *vc, int format)
void IM_DECL imVideoCaptureGetImageSize (imVideoCapture *vc, int *width, int *height)
int IM_DECL imVideoCaptureSetImageSize (imVideoCapture *vc, int width, int height)
int IM_DECL imVideoCaptureFrame (imVideoCapture *vc, unsigned char *data, int color_mode, int timeout)
int IM_DECL imVideoCaptureOneFrame (imVideoCapture *vc, unsigned char *data, int color_mode)
int IM_DECL imVideoCaptureLive (imVideoCapture *vc, int live)
int IM_DECL imVideoCaptureResetAttribute (imVideoCapture *vc, const char *attrib, int fauto)
int IM_DECL imVideoCaptureGetAttribute (imVideoCapture *vc, const char *attrib, float *percent)
int IM_DECL imVideoCaptureSetAttribute (imVideoCapture *vc, const char *attrib, float percent)
const char **IM_DECL imVideoCaptureGetAttributeList (imVideoCapture *vc, int *num_attrib)
```

Detailed Description

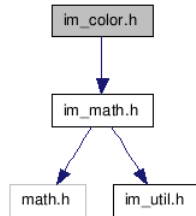
See Copyright Notice in [im.h](#)



im_color.h File Reference

Color Manipulation. [More...](#)

Include dependency graph for im_color.h:



[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)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

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))

```



Generated on Thu Oct 1 11:40:02 2009 for IM by 1.6.1

im_colorhsi.h File Reference

HSI Color Manipulation. [More...](#)

[Go to the source code of this file.](#)

Functions

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)

Detailed Description

See Copyright Notice in [im_lib.h](#)

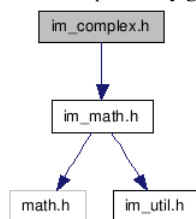


Generated on Thu Oct 1 11:40:02 2009 for IM by 1.6.1

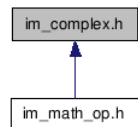
im_complex.h File Reference

Complex Data Type. [More...](#)

Include dependency graph for im_complex.h:



This graph shows which files directly or indirectly include this file:



[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)
 int **operator**< (const [imcfloat](#) &C1, const [imcfloat](#) &C2)
 int **operator**< (const [imcfloat](#) &C, const float &F)
 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)

Detailed Description

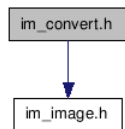
See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:02 2009 for IM by  1.6.1

im_convert.h File Reference

Image Conversion. [More...](#)

Include dependency graph for im_convert.h:



[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
    * imImageGetOpenGLData (const imImage *image, int *glformat)
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)
  
```

Detailed Description

See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:02 2009 for IM by  1.6.1

im_counter.h File Reference

Processing Counter. [More...](#)

[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)
int imCounterIncTo (int counter, int count)
void imCounterTotal (int counter, int total, const char *message)
  
```

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:02 2009 for IM by 1.6.1

im_dib.h File Reference

Windows DIB (Device Independent Bitmap). [More...](#)

[Go to the source code of this file.](#)

Data Structures

struct [imDib](#)
Windows DIB Structure. [More...](#)

Typedefs

typedef struct
[imDib](#) [imDib](#)
typedef unsigned int
([* imDibLineGetPixel](#))(unsigned char *line, int col)
typedef void([* imDibLineSetPixel](#))(unsigned char *line, int col, unsigned int 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)

Detailed Description

See Copyright Notice in [im_lib.h](#)

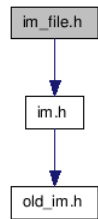


Generated on Thu Oct 1 11:40:02 2009 for IM by 1.6.1

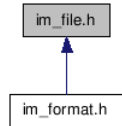
im_file.h File Reference

File Access. [More...](#)

Include dependency graph for im_file.h:



This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Data Structures

struct [imFile](#)

Image File Format Base Class (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)
void imFileSetBaseAttributes (imFile *ifile)
  
```

Detailed Description

See Copyright Notice in [im_lib.h](#)

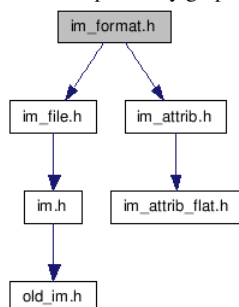


Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_format.h File Reference

File Format Access. [More...](#)

Include dependency graph for im_format.h:



[Go to the source code of this file.](#)

Data Structures

class [imFileFormatBase](#)

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

class [imFormat](#)

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

Functions

```

imFileFormatBase * imFileFormatBaseOpen (const char *file_name, int *error)
imFileFormatBase * imFileFormatBaseOpenAs (const char *file_name, const char *format, int *error)
imFileFormatBase * imFileFormatBaseNew (const char *file_name, const char *format, int *error)
void imFormatRegister (imFormat *ifformat)
  
```

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_format_all.h File Reference

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. [More...](#)

[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)

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_format_avi.h File Reference

Register the AVI Format. [More...](#)

[Go to the source code of this file.](#)

Functions

void **imFormatRegisterAVI** (void)

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_format_jp2.h File Reference

Register the JP2 Format. [More...](#)

[Go to the source code of this file.](#)

Functions

void **imFormatRegisterJP2** (void)

Detailed Description

See Copyright Notice in [im_lib.h](#)



im_format_raw.h File Reference

Initialize the RAW Format Driver Header for internal use only. [More...](#)

[Go to the source code of this file.](#)

Functions

[imFormat](#) * [imFormatInitRAW](#) (void)
void [imFormatFinishRAW](#) (void)

Detailed Description

See Copyright Notice in [im_lib.h](#)



im_format_wmv.h File Reference

Register the WMF Format. [More...](#)

[Go to the source code of this file.](#)

Functions

void [imFormatRegisterWMV](#) (void)

Detailed Description

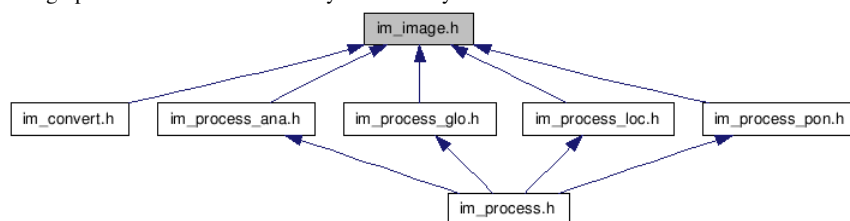
See Copyright Notice in [im_lib.h](#)



im_image.h File Reference

Image Manipulation. [More...](#)

This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Data Structures

struct [imImage](#)
imImage Structure Definition. [More...](#)

Defines

#define [imcdCanvasPutImage](#)(_canvas, _image, _x, _y, _w, _h, _xmin, _xmax, _ymin, _ymax)

Typedefs

typedef struct
[imImage](#) [imImage](#)

Functions

[imImage](#) * [imImageCreate](#) (int width, int height, int color_space, int data_type)
[imImage](#) * [imImageInit](#) (int width, int height, int color_space, int data_type, void *data_buffer, long *palette, int palette_count)
[imImage](#) * [imImageCreateBased](#) (const [imImage](#) *image, int width, int height, int color_space, int data_type)
void [imImageDestroy](#) ([imImage](#) *image)
void [imImageAddAlpha](#) ([imImage](#) *image)

```

void imImageReshape (imImage *image, int width, int height)
void imImageCopy (const imImage *src_image, imImage *dst_image)
void imImageCopyData (const imImage *src_image, imImage *dst_image)
imImage * imImageDuplicate (const imImage *image)
imImage * imImageClone (const imImage *image)
void imImageSetAttribute (const imImage *image, const char *attrib, int data_type, int count, const void *data)
const void * imImageGetAttribute (const imImage *image, const char *attrib, int *data_type, int *count)
void imImageGetAttributeList (const imImage *image, char **attrib, int *attrib_count)
void imImageClear (imImage *image)
int imImageIsBitmap (const imImage *image)
void imImageSetPalette (imImage *image, long *palette, int palette_count)
void imImageCopyAttributes (const imImage *src_image, imImage *dst_image)
int imImageMatchSize (const imImage *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)
void imImageMakeGray (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)
imImage * imFileLoadImageRegion (imFile *ifile, int index, int bitmap, int *error, int xmin, int xmax, int ymin, int ymax, int width, int height)
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)
imImage * imFileImageLoadRegion (const char *file_name, int index, int bitmap, int *error, int xmin, int xmax, int ymin, int ymax, int width, int height)
int imFileImageSave (const char *file_name, const char *format, const imImage *image)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:03 2009 for IM by



1.6.1

im_lib.h File Reference

Library Management and Main Documentation. [More...](#)

[Go to the source code of this file.](#)

Defines

```

#define IM_AUTHOR "Antonio Scuri"
#define IM_COPYRIGHT "Copyright (C) 1994-2009 Tecgraf, PUC-Rio."
#define IM_VERSION "3.5"
#define IM_VERSION_NUMBER 305000
#define IM_VERSION_DATE "2009/10/02"
#define IM_DESCRIPTION "Image Representation, Storage, Capture and Processing"
#define IM_NAME "IM - An Imaging Toolkit"

```

Functions

```

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

```

Detailed Description

See Copyright Notice in this file.

Generated on Thu Oct 1 11:40:03 2009 for IM by

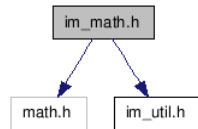


1.6.1

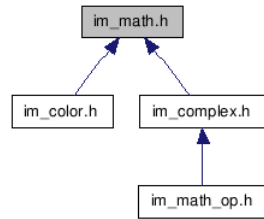
im_math.h File Reference

Math Utilities. [More...](#)

Include dependency graph for im_math.h:



This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Defines

```
#define C0 (-c3 + 2.0f*c2 - c)
#define C1 ( c3 - 2.0f*c2 + 1.0f)
#define C2 (-c3 + c2 + c)
#define C3 ( c3 - c2)
```

Functions

```
int imRound(float x)
int imRound(double x)
int imResample(int x, float factor)
template<class T, class TU>
    T imZeroOrderDecimation(int width, int height, T *map, float x1, float y1, float box_width, float box_height, TU Dummy)
template<class T, class TU>
    T imBilinearDecimation(int width, int height, T *map, float x1, float y1, float box_width, float box_height, TU Dummy)
template<class T>
    T imZeroOrderInterpolation(int width, int height, T *map, float x1, float y1)
template<class T>
    T imBilinearInterpolation(int width, int height, T *map, float x1, float y1)
template<class T, class TU>
    T imBicubicInterpolation(int width, int height, T *map, float x1, float y1, TU Dummy)
template<class T>
    void imMinMax(const T *map, int count, T &min, T &max)
```

Detailed Description

See Copyright Notice in [im_lib.h](#)

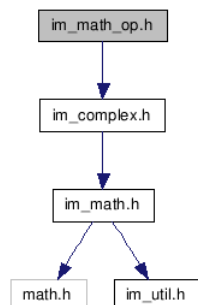


Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_math_op.h File Reference

Math Operations. [More...](#)

Include dependency graph for im_math_op.h:



[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 sqr (const int &C)
template<class T >
    T sqr\_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)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_palette.h File Reference

Palette Generators. [More...](#)

[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)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_plus.h File Reference

C++ Wrapper for File Access. [More...](#)

[Go to the source code of this file.](#)

Data Structures

class [imImageFile](#)

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

Detailed Description

See Copyright Notice in [im_lib.h](#)

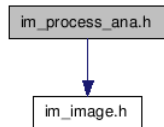


Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

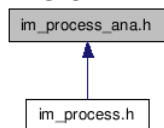
im_process_ana.h File Reference

Image Statistics and Analysis. [More...](#)

Include dependency graph for im_process_ana.h:



This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Data Structures

struct [imStats](#)

Typedefs

typedef struct
 [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 imCalcHistoImageStatistics (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, int region_count)
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, int region_count)
void imProcessPerimeterLine (const imImage *src_image, imImage *dst_image)
void imProcessRemoveByArea (const imImage *src_image, imImage *dst_image, int connect, int start_size, int end_size, int
    inside)
void imProcessFillHoles (const imImage *src_image, imImage *dst_image, int connect)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

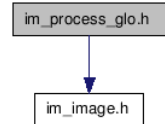


Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

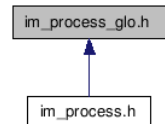
im_process_glo.h File Reference

Image Processing - Global Operations. [More...](#)

Include dependency graph for im_process_glo.h:



This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Functions

```

int imProcessHoughLines (const imImage *src_image, imImage *dst_image)
int imProcessHoughLinesDraw (const imImage *src_image, const imImage *hough, const imImage *hough_points, imImage
    *dst_image)
void imProcessCrossCorrelation (const imImage *src_image1, const imImage *src_image2, imImage *dst_image)
void imProcessAutoCorrelation (const imImage *src_image, imImage *dst_image)
void imProcessDistanceTransform (const imImage *src_image, imImage *dst_image)
void imProcessRegionalMaximum (const imImage *src_image, imImage *dst_image)
void imProcessFFT (const imImage *src_image, imImage *dst_image)
void imProcessIFFT (const imImage *src_image, imImage *dst_image)
void imProcessFFTraw (imImage *image, int inverse, int center, int normalize)
void imProcessSwapQuadrants (imImage *image, int center2origin)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

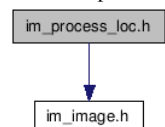


Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

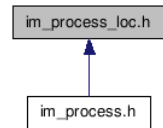
im_process_loc.h File Reference

Image Processing - Local Operations. [More...](#)

Include dependency graph for im_process_loc.h:



This graph shows which files directly or indirectly include this file:



[Go to the source code of this file.](#)

Functions

```

int imProcessReduce (const imImage *src_image, imImage *dst_image, int order)
int imProcessResize (const imImage *src_image, imImage *dst_image, int order)
void imProcessReduceBy4 (const imImage *src_image, imImage *dst_image)
void imProcessCrop (const imImage *src_image, imImage *dst_image, int xmin, int ymin)
void imProcessInsert (const imImage *src_image, const imImage *region_image, imImage *dst_image, int xmin, int ymin)
void imProcessAddMargins (const imImage *src_image, imImage *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 imImage *src_image, imImage *dst_image, double cos0, double sin0, int order)
int imProcessRotateRef (const imImage *src_image, imImage *dst_image, double cos0, double sin0, int x, int y, int to_origin, int order)
void imProcessRotate90 (const imImage *src_image, imImage *dst_image, int dir_clockwise)
void imProcessRotate180 (const imImage *src_image, imImage *dst_image)
void imProcessMirror (const imImage *src_image, imImage *dst_image)
void imProcessFlip (const imImage *src_image, imImage *dst_image)
int imProcessRadial (const imImage *src_image, imImage *dst_image, float k1, int order)
int imProcessSwirl (const imImage *src_image, imImage *dst_image, float k1, int order)
void imProcessInterlaceSplit (const imImage *src_image, imImage *dst_image1, imImage *dst_image2)
int imProcessGrayMorphConvolve (const imImage *src_image, imImage *dst_image, const imImage *kernel, int ismax)
int imProcessGrayMorphErode (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGrayMorphDilate (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGrayMorphOpen (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGrayMorphClose (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGrayMorphTopHat (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGrayMorphWell (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGrayMorphGradient (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessBinMorphConvolve (const imImage *src_image, imImage *dst_image, const imImage *kernel, int hit_white, int iter)
int imProcessBinMorphErode (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphDilate (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphOpen (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphClose (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
int imProcessBinMorphOutline (const imImage *src_image, imImage *dst_image, int kernel_size, int iter)
void imProcessBinMorphThin (const imImage *src_image, imImage *dst_image)
int imProcessMedianConvolve (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessRangeConvolve (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessRankClosestConvolve (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessRankMaxConvolve (const imImage *src_image, imImage *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 imProcessConvolveSep (const imImage *src_image, imImage *dst_image, const imImage *kernel)
int imProcessConvolveDual (const imImage *src_image, imImage *dst_image, const imImage *kernel1, const imImage *kernel2)
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 imProcessLapOfGaussianConvolve (const imImage *src_image, imImage *dst_image, float stddev)
int imProcessMeanConvolve (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessGaussianConvolve (const imImage *src_image, imImage *dst_image, float stddev)
int imProcessBarlettConvolve (const imImage *src_image, imImage *dst_image, int kernel_size)
int imProcessSobelConvolve (const imImage *src_image, imImage *dst_image)
int imProcessPrewittConvolve (const imImage *src_image, imImage *dst_image)
int imProcessSplineEdgeConvolve (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 imGaussianStdDev2KernelSize (float stddev)
float imGaussianKernelSize2StdDev (int kernel_size)
int imProcessUnsharp (const imImage *src_image, imImage *dst_image, float stddev, float amount, float threshold)
int imProcessSharp (const imImage *src_image, imImage *dst_image, float amount, float threshold)
int imProcessSharpKernel (const imImage *src_image, const imImage *kernel, imImage *dst_image, float amount, float threshold)
  
```

Detailed Description

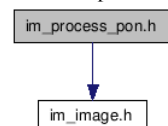
See Copyright Notice in [im_lib.h](#)



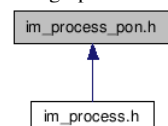
im_process_pon.h File Reference

Image Processing - Pontual Operations. [More...](#)

Include dependency graph for im_process_pon.h:



This graph shows which files directly or indirectly include this file:



[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\_EOL, IM\_UN\_ABS, IM\_UN\_LESS, IM\_UN\_INV,
    IM\_UN\_SQR, IM\_UN\_SORT, 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 imProcessBlendConst (const imImage *src_image1, const imImage *src_image2, imImage *dst_image, float alpha)
void imProcessBlend (const imImage *src_image1, const imImage *src_image2, const imImage *alpha_image, imImage
    *dst_image)
void imProcessSplitComplex (const imImage *src_image, imImage *dst_image1, imImage *dst_image2, int 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 (imImage *image, float *value)
int imProcessRenderWheel (imImage *image, int internal_radius, int external_radius)
int imProcessRenderCone (imImage *image, int radius)
int imProcessRenderTent (imImage *image, int tent_width, int tent_height)
int imProcessRenderRamp (imImage *image, int start, int end, int vert_dir)
int imProcessRenderBox (imImage *image, int box_width, int box_height)
int imProcessRenderSinc (imImage *image, float x_period, float y_period)
int imProcessRenderGaussian (imImage *image, float stddev)
int imProcessRenderLapOfGaussian (imImage *image, float stddev)
int imProcessRenderCosine (imImage *image, float x_period, float y_period)
int imProcessRenderGrid (imImage *image, int x_space, int y_space)
int imProcessRenderChessboard (imImage *image, int x_space, int y_space)
void imProcessToneGamut (const imImage *src_image, imImage *dst_image, int op, float *param)
void imProcessUnNormalize (const imImage *src_image, imImage *dst_image)
void imProcessDirectConv (const imImage *src_image, imImage *dst_image)
void imProcessNegative (const imImage *src_image, imImage *dst_image)
void imProcessThreshold (const imImage *src_image, imImage *dst_image, int level, int value)
void imProcessThresholdByDiff (const imImage *src_image1, const imImage *src_image2, imImage *dst_image)
void imProcessHysteresisThreshold (const imImage *src_image, imImage *dst_image, int low_thres, int high_thres)
void imProcessHysteresisThresEstimate (const imImage *image, int *low_level, int *high_level)
int imProcessUniformErrThreshold (const imImage *src_image, imImage *dst_image)
void imProcessDifusionErrThreshold (const imImage *src_image, imImage *dst_image, int level)
int imProcessPercentThreshold (const imImage *src_image, imImage *dst_image, float percent)
int imProcessOtsuThreshold (const imImage *src_image, imImage *dst_image)
int imProcessMinMaxThreshold (const imImage *src_image, imImage *dst_image)
void imProcessLocalMaxThresEstimate (const imImage *image, int *level)
void imProcessSliceThreshold (const imImage *src_image, imImage *dst_image, int start_level, int end_level)
void imProcessPixelate (const imImage *src_image, imImage *dst_image, int box_size)
void imProcessPosterize (const imImage *src_image, imImage *dst_image, int level)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)



Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_raw.h File Reference

RAW File Format. [More...](#)

[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)

Detailed Description

See Copyright Notice in [im_lib.h](#)

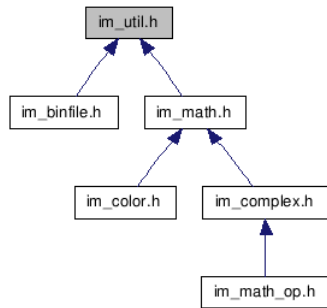


Generated on Thu Oct 1 11:40:03 2009 for IM by 1.6.1

im_util.h File Reference

Utilities. [More...](#)

This graph shows which files directly or indirectly include this file:



[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 imColorModeIsPacked(_cm) (_cm & IM_PACKED)
#define imColorModeIsTopDown(_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 imColorModeIsBitmap (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)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:04 2009 for IM by



1.6.1

imlua.h File Reference

IM Lua 5 Binding. [More...](#)

[Go to the source code of this file.](#)

Functions

```

int imlua\_open (lua_State *L)
int luaopen\_imlua (lua_State *L)
void imlua\_pushimage (lua_State *L, imImage *image)
imImage * imlua\_checkimage (lua_State *L, int param)
int imlua\_open\_capture (lua_State *L)
int luaopen\_imlua\_capture (lua_State *L)
int imlua\_open\_process (lua_State *L)
int luaopen\_imlua\_process (lua_State *L)
int imlua\_open\_fftw (lua_State *L)
int luaopen\_imlua\_fftw (lua_State *L)

```

Detailed Description

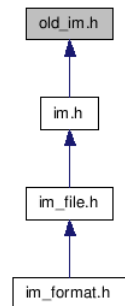
See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:04 2009 for IM by  1.6.1

old_im.h File Reference

Old API. [More...](#)

This graph shows which files directly or indirectly include this file:



[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)

```

Detailed Description

See Copyright Notice in [im_lib.h](#)

Generated on Thu Oct 1 11:40:04 2009 for IM by  1.6.1

- [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](#)

Generated on Thu Oct 1 11:40:09 2009 for IM by  1.6.1

- [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](#)



Generated on Thu Oct 1 11:40:09 2009 for IM by 1.6.1

- [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](#)
- [imImage](#) : [im_image.h](#)
- [imRenderCondFunc](#) : [im_process_pon.h](#)
- [imRenderFunc](#) : [im_process_pon.h](#)
- [imStats](#) : [im_process_ana.h](#)



Generated on Thu Oct 1 11:40:09 2009 for IM by 1.6.1

- [All](#)
- [Functions](#)
- [Typedefs](#)
- [Enumerations](#)
- [Enumerator](#)
- [Defines](#)

- [i](#)

- 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_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](#)