Convert, Edit, and Compose Images

ImageMagick User's Guide Version 6.0.0

John Cristy
Bob Friesenhahn
Glenn Randers-Pehrson

ImageMagick Studio LLC
http://www.imagemagick.org

Copyright

ImageMagick is distributed under the following license:

Copyright 1999-2004 ImageMagick Studio LLC, a non-profit organization dedicated to making software imaging solutions freely available.

1. Definitions.
   ”License” shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.
   ”Licensor” shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.
   ”Legal Entity” shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, ”control” means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50or (iii) beneficial ownership of such entity.
   ”You” (or ”Your”) shall mean an individual or Legal Entity exercising permissions granted by this License.
   ”Source” form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.
   ”Object” form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.
   ”Work” shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).
   ”Derivative Works” shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.
   ”Contribution” shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or

Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, ”submitted” means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as ”Not a Contribution.”
”Contributor” shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
   (d) If the Work includes a ”NOTICE” text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for

informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

(e) Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

5. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
6. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an ”AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
7. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
8. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole re-

sponsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an ”AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

Contents

Preface

About This Book

Acknowledgement

Part 1
Quick Start Guide

1 Introduction

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

1.1 What is ImageMagick

1.1.1 Command-line Utility

1.1.2 Application Programming Interface

1.1.3 Scripting Language

1.1.4 General Purpose Imaging Solution

1.2 Getting Help

1.2.1 Web Site

1.2.2 Mailing List

1.2.3 Defect Tracking System

2 Image Primer

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

2.1 What is an Image

2.2 Image Depth

2.3 Colormapped Images

2.4 Compression

2.4.1 Lossless

2.4.2 Lossy

2.5 Colorspace

2.5.1 RGB

2.5.2 CMYK

2.6 Meta-Information

2.7 Image Formats

3 Image Tools

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

3.0.1 Identify

3.0.2 Convert

3.0.3 Mogrify

3.0.4 Composite

3.0.5 Montage

3.0.6 Display

3.0.7 Animate

3.0.8 Import

3.0.9 Conjure

4 Image Transformations

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

5 Advanced ImageMagick Features

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

5.1 Working with Multi-resolution Images

5.1.1 PCD

5.1.2 PTIF

5.2 Working with an Image Sequence

5.2.1 Animation

5.2.2 Delay

5.2.3 Loop

5.3 Working with a Group of Images

5.4 Working with Raw Images

5.4.1 Size

5.4.2 Depth

5.4.3 Interlace

5.5 Using ImageMagick from a Web Browser

Part 2
Application Programming Interface

6 C Application Programming Interface

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

6.1 Working with Blobs

6.2 Working with Threads

6.2.1 Posix

6.2.2 Windows

7 C++ Application Programming Interface

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

7.1 Working with Blobs

7.2 Working with Threads

7.2.1 Posix

7.2.2 Windows

8 Perl Application Programming Interface

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

8.1 Background

9 PHP Application Programming Interface

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

9.1 Background

10 Other Application Programming Interfaces

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

10.1 Java

10.2 Python

10.3 ImageMagick Integration Project

Part 3
User’s Guide

11 Image Channels

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

11.1 Working with Image Channels

12 Image Painting

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

12.1 Image Painting

12.1.1 Paint Type

Color

Matte

12.1.2 Paint Method

Floodfill

Point

Replace

FillToBorder

Reset

12.1.3 Fuzz Factor

13 Color Profiles

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

13.1 Working with Color Profiles

14 Image Drawing

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

14.1 SVG

14.2 MVG

Part 4
Installation And Administration Guide

15 Installing from Binary

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

15.1 Downloading

15.1.1 web

15.1.2 ftp

15.2 Linux RPM

15.3 Windows

15.4 VMS

15.5 Unix

15.6 Other

16 Installing from Source

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

16.1 Downloading

16.1.1 FTP

16.1.2 CVS

16.2 Unix

16.2.1 Configure

16.2.2 Modules

16.3 Windows

16.3.1 Configure

16.3.2 Modules

16.4 Macintosh

16.5 VMS

17 Customizing ImageMagick

Abstract Please start every chapter with a short summary of what the reader may expect.

To start with we suggest that every heading is followed by at least a short passage of text in order to avoid a simple listing of different hierarchies.

17.1 Image Depth

17.1.1 8-bit

17.1.2 16-bit

17.2 Image Cache

17.2.1 Persistent Cache

17.3 Delegates

17.3.1 Library Delegates

17.3.2 Delegates.mgk

17.4 magic.mgk

17.5 type.mgk

Part 5
Reference Manual

18 Supported Image Formats

Listed here are the various file formats supported by ImageMagick. The Format is the image format identifier and is typically used as the image file extension (e.g. image.png for the PNG image format). The mode shows the type of support: r = read; w = write; + = multi-image files. So for example, a mode of rw+ means ImageMagick can read, write, and save more than one image of a sequence to the same blob or file. Finally the description tells what the image format is in case you cannot tell directly from the format identifier (e.g. 8BIM is the Photoshop resource format).

Table18.1: Supported Image Formats

Supported Image Formats

Format	Mode	Description
8BIM	rw-	Photoshop resource format
AFM	r–	TrueType font
APP1	rw-	Photoshop resource format
ART	r–	PF1: 1st Publisher
AVI	r–	Audio/Visual Interleaved
AVS	rw+	AVS X image
BIE	rw-	Joint Bi-level Image experts Group interchange format
BMP	rw+	Microsoft Windows bitmap image
CAPTION	*r+	Caption (requires separate size info)
CMYK	rw-	Raw cyan, magenta, yellow, and black samples (8 or 16 bits, depending on the image depth)
CMYKA	rw-	Raw cyan, magenta, yellow, black, and matte samples (8 or 16 bits, depending on the image depth)
CUT	r–	DR Halo
DCM	r–	Digital Imaging and Communications in Medicine image

Supported Image Formats (continued)

Format	Mode	Description
DCX	rw+	ZSoft IBM PC multi-page Paintbrush
DIB	rw+	Microsoft Windows bitmap image
DPS	r–	Display Postscript
DPX	r–	Digital Moving Picture Exchange
EPDF	rw-	Encapsulated Portable Document Format
EPI	rw-	Adobe Encapsulated PostScript Interchange format
EPS	rw-	Adobe Encapsulated PostScript
EPS2	-w-	Adobe Level II Encapsulated PostScript
EPS3	-w-	Adobe Level III Encapsulated PostScript
EPSF	rw-	Adobe Encapsulated PostScript
EPSI	rw-	Adobe Encapsulated PostScript Interchange format
EPT	rw-	Adobe Encapsulated PostScript with TIFF preview
FAX	rw+	Group 3 FAX
FILE	r–	Uniform Resource Locator
FITS	rw-	Flexible Image Transport System
FPX	rw-	FlashPix Format
FTP	r–	Uniform Resource Locator
G3	rw-	Group 3 FAX
GIF	rw+	CompuServe graphics interchange format
GIF87	rw-	CompuServe graphics interchange format (version 87a)
GRADIENT	r–	Gradual passing from one shade to another
GRANITE	r–	Granite texture
GRAY	rw+	Raw gray samples (8 or 16 bits, depending on the image depth)
H	rw-	Internal format
HDF		rw+ Hierarchical Data Format
HISTOGRAM	-w-	Histogram of the image
HTM	-w-	Hypertext Markup Language and a client-side image map
HTML	-w-	Hypertext Markup Language and a client-side image map
HTTP	r–	Uniform Resource Locator
ICB	rw+	Truevision Targa image
ICM	rw-	ICC Color Profile
ICO	r–	Microsoft icon
ICON	r–	Microsoft icon
IMPLICIT	—	
IPTC	rw-	IPTC Newsphoto
JBG	rw+	Joint Bi-level Image experts Group interchange format
JBIG	rw+	Joint Bi-level Image experts Group interchange format
JP2	rw-	JPEG-2000 JP2 File Format Syntax
JPC	rw-	JPEG-2000 Code Stream Syntax
JPEG	rw-	Joint Photographic Experts Group JFIF format

Supported Image Formats (continued)

Format	Mode	Description
JPG	rw-	Joint Photographic Experts Group JFIF format
LABEL	r–	Text image format
LOGO	rw-	ImageMagick Logo
M2V	rw+	MPEG-2 Video Stream
MAP	rw-	Colormap intensities (8 or 16 bits, depending on the image depth) and indices (8 or 16 bits, depending on whether $colors \leq 256$).
MAT	-w+	MATLAB image format
MATTE	-w+	MATTE format
MIFF	rw+	Magick image format
MNG	rw+	Multiple-image Network Graphics
MONO	rw-	Bi-level bitmap in least-significant-byte first order
MPC	rw-	Magick Persistent Cache image format
MPEG	rw+	MPEG-1 Video Stream
MPG	rw+	MPEG-1 Video Stream
MPR	r–	Magick Persistent Registry
MSL	r–	Magick Scripting Language
MTV	rw+	MTV Raytracing image format
MVG	rw-	Magick Vector Graphics
NETSCAPE	r–	Netscape 216 color cube
NULL	r–	Constant image of uniform color
OTB	rw-	On-the-air bitmap
P7	rw+	Xv thumbnail format
PAL	rw-	16bit/pixel interleaved YUV
PALM	rw-	PALM Pixmap
PBM	rw+	Portable bitmap format (black and white)
PCD	rw-	Photo CD
PCDS	rw-	Photo CD
PCL	-w-	Page Control Language
PCT	rw-	Apple Macintosh QuickDraw/PICT
PCX	rw-	ZSoft IBM PC Paintbrush
PDB	r–	Pilot Image Format
PDF	rw+	Portable Document Format
PFA	r–	TrueType font
PFB	r–	TrueType font
PFM	r–	TrueType font
PGM	rw+	Portable graymap format (gray scale)
PICON	rw-	Personal Icon
PICT	rw-	Apple Macintosh QuickDraw/PICT
PIX	r–	Alias/Wavefront RLE image format
PLASMA	r–	Plasma fractal image

Supported Image Formats (continued)

Format	Mode	Description
PM	rw-	X Windows system pixmap (color)
PNG	rw-	Portable Network Graphics
PNM	rw+	Portable anymap
PPM	rw+	Portable pixmap format (color)
PREVIEW	-w-	Show a preview an image enhancement, effect, or f/x
PS	rw+	Adobe PostScript
PS2	-w+	Adobe Level II PostScript
PS3	-w+	Adobe Level III PostScript
PSD	rw-	Adobe Photoshop bitmap
PTIF	rw-	Pyramid encoded TIFF
PWP	r–	Seattle Film Works
RAS	rw+	SUN Rasterfile
RGB	rw+	Raw red, green, and blue samples (8 or 16 bits, depending on the image depth)
RGBA	rw+	Raw red, green, blue, and matte samples (8 or 16 bits, depending on the image depth)
RLA	r–	Alias/Wavefront image
RLE	r–	Utah Run length encoded image
ROSE	*rw-	70x46 Truecolor test image
SCT	r–	Scitex HandShake
SFW	r–	Seattle Film Works
SGI	rw+	Irix RGB image
SHTML	-w-	Hypertext Markup Language and a client-side image map
STEGANO	r–	Steganographic image
SUN	rw+	SUN Rasterfile
SVG	rw+	Scalable Vector Gaphics
TEXT	rw+	Raw text
TGA	rw+	Truevision Targa image
TIF	rw+	Tagged Image File Format
TIFF	rw+	Tagged Image File Format
TILE	r–	Tile image with a texture
TIM	r–	PSX TIM
TTF	r–	TrueType font
TXT	rw+	Raw text
UIL	-w-	X-Motif UIL table
UYVY	rw-	16bit/pixel interleaved YUV
VDA	rw+	Truevision Targa image
VICAR	rw-	VICAR rasterfile format
VID	rw+	Visual Image Directory
VIFF	rw+	Khoros Visualization image
VST	rw+	Truevision Targa image

Supported Image Formats (continued)

Format	Mode	Description
WBMP	rw-	Wireless Bitmap (level 0) image
WMF	r–	Windows Metafile
WPG	r–	Word Perfect Graphics
X	rw-	X Image
XBM	rw-	X Windows system bitmap (black and white)
XC	r–	Constant image uniform color
XCF	r–	GIMP image
XML	r–	Scalable Vector Gaphics
XPM	rw-	X Windows system pixmap (color)
XV	rw+	Khoros Visualization image
XWD	rw-	X Windows system window dump (color)
YUV	rw-	CCIR 601 4:1:1

Your installation might not support all of the formats in the list. To get an up-to-date listing of the formats supported by your particular configuration, run ”`convert -list format`”.

19 Commandline Options

This is a combined list of the commandline options used by the ImageMagick utilities (*animate*, *composite*, *convert*, *display*, *identify*, *import*, *mogrify* and *montage*).

In this document, angle brackets ("<>") enclose variables and curly brackets ("{}") enclose optional parameters. For example, "**-fuzz** <**distance**>{**%**}" means you can use the option `"-fuzz 10"` or `"-fuzz 2%"`.

-adjoin join images into a single multi-image file

By default, all images of an image sequence are stored in the same file. However, some formats (e.g. JPEG) do not support more than one image and are saved to separate files. Use **+adjoin** to force this behavior.

-affine <***matrix***> drawing transformation matrix

This option provides a transformation matrix {`sx,rx,ry,sy,tx,ty`} for use by subsequent **-draw** or **-transform** options.

The transformation matrix has 3x3 elements, but three of them are omitted from the input because they are constant. The complete matrix is

```
sx rx  0
ry sy  0
tx ty  1
```

Scaling by the factor `s` is accomplished with the matrix

```
{s,0,0,s,0,0}
```

Translation by a displacement {`dx,dy`} is accomplished with the matrix

```
{1,0,0,1,dx,dy}
```

Rotation counterclockwise about the origin by an angle `a` is accomplished with the matrix

```
{cos(a),sin(a),-sin(a),cos(a),0,0}
```

A series of operations can be accomplished by using a matrix that is the multiple of the matrices for each operation.

-antialias remove pixel aliasing

By default antialiasing algorithms are used when drawing objects (e.g. lines) or rendering vector formats (e.g. WMF and Postscript). Use +antialias to disable use of antialiasing algorithms. Reasons to disable antialiasing

include avoiding increasing colors in the image, or improving rendering speed.

-append append a set of images

This option creates a single image where the images in the original set are stacked top-to-bottom. If they are not of the same width, any narrow images will be expanded to fit using the background color. Use **+append** to stack images left-to-right. The set of images is terminated by the appearance of any option. If the **-append** option appears after all of the input images, all images are appended.

-authenticate *<string>* decrypt image with this password

Use this option to supply a password for decrypting an image or an image sequence, if it is being read from a format such as PDF that supports encryption. Encrypting images being written is not supported.

-average average a set of images

The set of images is terminated by the appearance of any option. If the **-average** option appears after all of the input images, all images are averaged.

-backdrop *<color>* display the image centered on a backdrop.

This backdrop covers the entire workstation screen and is useful for hiding other X window activity while viewing the image. The color of the backdrop is specified as the background color. The color is specified using the format described under the **-fill** option. Refer to “X Resources” in the manual page for *display* for details.

-background *<color>* the background color

The color is specified using the format described under the **-fill** option.

-blue-primary *<**x**>,<**y**>* blue chromaticity primary point

-blur *<**radius**>{**x**<**sigma**>}* blur the image with a Gaussian operator

Blur with the given radius and standard deviation (sigma).

-border *<**width**>**x**<**height**>* surround the image with a border of color

See **-geometry** for details about the geometry specification.

-bordercolor *<**color**>* the border color

The color is specified using the format described under the **-fill** option.

-borderwidth *<**geometry**>* the border width

-box *<**color**>* set the color of the annotation bounding box

The color is specified using the format described under the **-fill** option.

See **-draw** for further details.

-channel *<**type**>* the type of channel

Choose from: **Red**, **Green**, **Blue**, **Alpha**, **Cyan**, **Magenta**, **Yellow**, **Black**, or All.

Use this option to apply an image-processing option to a particular *channel* from the image.

-charcoal *<**factor**>* simulate a charcoal drawing

-chop *<**width**>**x**<**height**>{**+-**}<**x**>{**+-**}<**y**>{**%**}* remove pixels from the interior of an image

Width and *height* give the number of columns and rows to remove, and *x* and *y* are offsets that give the location of the leftmost column and topmost row to remove.

The *x* offset normally specifies the leftmost column to remove. If the **-gravity** option is present with *NorthEast*, *East*, or *SouthEast* gravity, it gives the distance leftward from the right edge of the image to the rightmost column to remove. Similarly, the *y* offset normally specifies the topmost row to remove, but if the **-gravity** option is present with *SouthWest*, *South*, or *SouthEast* gravity, it specifies

the distance upward from the bottom edge of the image to the bottom row to remove.

The **-chop** option removes entire rows and columns, and moves the remaining corner blocks leftward and upward to close the gaps.

-clip apply the clipping path, if one is present

If a clipping path is present, it will be applied to subsequent operations.

For example, if you type the following command:

```
convert -clip -negate cockatoo.tif negated.tif
```

only the pixels within the clipping path are negated.

The **-clip** feature requires the XML library. If the XML library is not present, the option is ignored.

-coalesce merge a sequence of images

Each image N in the sequence after Image 0 is replaced with the image created by flattening images 0 through N.

The set of images is terminated by the appearance of any option. If the **-coalesce** option appears after all of the input images, all images are coalesced.

-colorize <***value***> colorize the image with the pen color

Specify the amount of colorization as a percentage. You can apply separate colorization values to the red, green, and blue channels of the image with a colorization value list delimited with slashes (e.g. 0/0/50).

-colormap <***type***> define the colormap type

Choose between **shared** or **private**.

This option only applies when the default X server visual is *PseudoColor* or *GRAYScale*. Refer to **-visual** for more details. By default, a shared colormap is allocated. The image shares colors with other X clients. Some image colors could be approximated, therefore your image may look very different than intended. Choose **Private** and the image colors appear exactly as they are defined. However, other clients may go *technicolor* when the image colormap is installed.

-colors <***value***> preferred number of colors in the image

The actual number of colors in the image may be less than your request, but never more. Note, this is a color reduction option. Images with less unique colors

than specified with this option will have any duplicate or unused colors removed. Refer to quantize for more details.

Note, options **-dither**, **-colorspace**, and **-treedepth** affect the color reduction algorithm.

-colorspace <***value***> the type of colorspace

Choices are: **GRAY**, **OHTA**, **RGB**, **Transparent**, **XYZ**, **YCbCr**, **YIQ**, **YPbPr**, **YUV**, or **CMYK**.

Color reduction, by default, takes place in the RGB color space. Empirical evidence suggests that distances in color spaces such as YUV or YIQ correspond to perceptual color differences more closely than do distances in RGB space. These color spaces may give better results when color reducing an image. Refer to quantize for more details.

The **Transparent** color space behaves uniquely in that it preserves the matte channel of the image if it exists.

The **-colors** or **-monochrome** option is required for this option to take effect.

-comment <***string***> annotate an image with a comment

Use this option to assign a specific comment to the image, when writing to an image format that supports comments. You can include the image filename, type, width, height, or other image attribute by embedding special format characters listed under the **-format** option. The comment is not drawn on the image, but is embedded in the image datastream via a "Comment" tag or similar mechanism. If you want the comment to be visible on the image itself, use the **-draw** option.

For example,

```
-comment "%m:%f %wx%h"
```

produces an image comment of **MIFF:bird.miff 512x480** for an image titled **bird.miff** and whose width is 512 and height is 480.

If the first character of *string* is *@*, the image comment is read from a file titled by the remaining characters in the string.

-compose <***operator***> the type of image composition

By default, each of the composite image pixels are replaced by the corresponding image tile pixel. You can choose an alternate composite operation:

```
Over
In
```

```
Out
Atop
Xor
Plus
Minus
Add
Subtract
Difference
Multiply
Bumpmap
Copy
CopyRed
CopyGreen
CopyBlue
CopyOpacity
```

How each operator behaves is described below.

Over
: The result will be the union of the two image shapes, with opaque areas of *composite image* obscuring *image* in the region of overlap.

In
: The result is simply *composite image* cut by the shape of *image*. None of the image data of *image* will be in the result.

Out
: The resulting image is *composite image* with the shape of *image* cut out.

Atop
: The result is the same shape as image *image*, with *composite image* obscuring *image* where the image shapes overlap. Note this differs from **over** because the portion of *composite image* outside *image*'s shape does not appear in the result.

Xor
: The result is the image data from both *composite image* and *image* that is outside the overlap region. The overlap region will be blank.

Plus
: The result is just the sum of the image data. Output values are cropped to MaxRGB (no overflow).

Minus
: The result of *composite image* - *image*, with underflow cropped to zero.

Add
: The result of *composite image* + *image*, with overflow wrapping around (*mod* (MaxRGB+1)).

Subtract
: The result of *composite image* - *image*, with underflow wrapping around (*mod* (MaxRGB+1)). The **add** and **subtract** operators can be used to perform reversible transformations.

Difference
: The result of abs(*composite image* - *image*). This is useful for comparing two very similar images.

Multiply
: The result of *composite image* * *image*. This is useful for the creation of drop-shadows.

Bumpmap
: The result *image* shaded by *composite image*.

Copy
: The resulting image is *image* replaced with *composite image*. Here the matte information is ignored.

CopyRed
: The resulting image is the red layer in *image* replaced with the red layer in *composite image*. The other layers are copied untouched.

CopyGreen
: The resulting image is the green layer in *image* replaced with the green layer in *composite image*. The other layers are copied untouched.

CopyBlue
: The resulting image is the blue layer in *image* replaced with the blue layer in *composite image*. The other layers are copied untouched.

CopyOpacity
: The resulting image is the matte layer in *image* replaced with the matte layer in *composite image*. The other layers are copied untouched.

The image compositor requires a matte, or alpha channel in the image for some operations. This extra channel usually defines a mask which represents a sort of a cookie-cutter for the image. This is the case when matte is opaque (full coverage) for pixels inside the shape, zero outside, and between 0 and MaxRGB on the boundary. For certain operations, if *image* does not have a matte channel, it is initialized with 0 for any pixel matching in color to pixel location (0,0), otherwise MaxRGB (to work properly **borderwidth** must be 0).

-compress *<**type**>* the type of image compression

Choices are: *None*, *BZip*, *Fax*, *Group4*, *JPEG*, *Lossless*, *LZW*, *RLE* or *Zip*.

Specify **+compress** to store the binary image in an uncompressed format. The default is the compression type of the specified image file.

If *LZW* compression is specified but LZW compression has not been enabled, the image data will be written in an uncompressed LZW format that can be read by LZW decoders. This may result in larger-than-expected GIF files.

"Lossless" refers to lossless JPEG, which is only available if the JPEG library has been patched to support it.

Use the **-quality** option to set the compression level to be used by JPEG, PNG, MIFF, and MPEG encoders. Use the **-sampling-factor** option to set the sampling factor to be used by JPEG, MPEG, and YUV encoders for downsampling the chroma channels.

-contrast enhance or reduce the image contrast

This option enhances the intensity differences between the lighter and darker elements of the image. Use **-contrast** to enhance the image or **+contrast** to reduce the image contrast.

For a more pronounced effect you can repeat the option:

```
convert rose: -contrast -contrast rose_c2.png
```

-convolve <***kernel***> convolve image with the specified convolution kernel

The kernel is specified as a comma-separated list of integers, ordered left-to right, starting with the top row. The order of the kernel is determined by the square root of the number of entries. Presently only square kernels are supported.

-crop <***width***>***x***<***height***>{**+-**}<***x***>{**+-**}<***y***>{***%***} preferred size and location of the cropped image

See **-geometry** for details about the geometry specification.

The width and height give the size of the image that remains after cropping, and *x* and *y* are offsets that give the location of the top left corner of the cropped image with respect to the original image. To specify the amount to be removed, use **-shave** instead.

If the *x* and *y* offsets are present, a single image is generated, consisting of the pixels from the cropping region. The offsets specify the location of the upper left corner of the cropping region measured downward and rightward with respect to the upper left corner of the image. If the **-gravity** option is present with *NorthEast, East,* or *SouthEast* gravity, it gives the distance leftward from the right edge of the image to the right edge of the cropping region. Similarly, if the **-gravity** option is present with *SouthWest, South*, or *SouthEast* gravity, the distance is measured upward between the bottom edges.

If the *x* and *y* offsets are omitted, a set of tiles of the specified geometry, covering the entire input image, is generated. The rightmost tiles and the bottom tiles are smaller if the specified geometry extends beyond the dimensions of the input image.

-cycle <***amount***> displace image colormap by amount

Amount defines the number of positions each colormap entry is shifted.

-debug <***events***> enable debug printout

The `events` parameter specifies which events are to be logged. It can be either `None`, `All`, or a comma-separated list consisting of one or more of the following domains: `Annotate`, `Blob`, `Cache`, `Coder`, `Configure`, `Locale`,

`Render`, `Resource`, `Transform`, `X11`, or `User`. For example, to log cache and blob events, use

```
convert -debug "Cache,Blob" rose: rose.png
```

The “User” domain is normally empty, but developers can log ”User” events in their private copy of ImageMagick.

Use the **-log** option to specify the format for debugging output.

Use **+debug** to turn off all logging.

-delete *<**index**>* delete image from the image sequence.

-deconstruct break down an image sequence into constituent parts

This option compares each image with the next in a sequence and returns the maximum bounding region of any pixel differences it discovers. This method can undo a coalesced sequence returned by the **-coalesce** option, and is useful for removing redundant information from a GIF or MNG animation.

The sequence of images is terminated by the appearance of any option. If the **-deconstruct** option appears after all of the input images, all images are deconstructed.

-delay *<**1/100ths of a second**>* display the next image after pausing

This option is useful for regulating the animation of image sequences *Delay/100* seconds must expire before the display of the next image. The default is no delay between each showing of the image sequence. The maximum delay is 65535.

You can specify a delay range (e.g. *-delay 10-500*) which sets the minimum and maximum delay.

-density *<**width**>**x**<**height**>* vertical and horizontal resolution in pixels of the image

This option specifies an image density when decoding a *PostScript* or Portable Document page. The default is 72 dots per inch in the horizontal and vertical direction. This option is used in concert with **-page**.

-depth *<**value**>* depth of the image

This is the number of bits in a color sample within a pixel. The only acceptable values are 8 or 16. Use this option to specify the depth of raw images whose depth is unknown such as GRAY, RGB, or CMYK, or to change the depth of any image after it has been read.

-descend obtain image by descending window hierarchy

-despeckle reduce the speckles within an image

-displace <***horizontal scale***>***x***<***vertical scale***> shift image pixels as defined by a displacement map

With this option, *composite image* is used as a displacement map. Black, within the displacement map, is a maximum positive displacement. White is a maximum negative displacement and middle gray is neutral. The displacement is scaled to determine the pixel shift. By default, the displacement applies in both the horizontal and vertical directions. However, if you specify *mask*, *composite image* is the horizontal X displacement and *mask* the vertical Y displacement.

-display <***host:display[.screen]***> specifies the X server to contact

This option is used with convert for obtaining image or font from this X server. See *X(1)*.

-dispose <***method***> GIF disposal method

The Disposal Method indicates the way in which the graphic is to be treated after being displayed.

Here are the valid methods:

```
Undefined       No disposal specified.
None            Do not dispose between frames.
Background      Overwrite the image area with
                the background color.
Previous        Overwrite the image area with
                what was there prior to rendering
                the image.
```

-dissolve <***percent***> dissolve an image into another by the given percent

The opacity of the composite image is multiplied by the given percent, then it is composited over the main image.

-dither apply Floyd/Steinberg error diffusion to the image

The basic strategy of dithering is to trade intensity resolution for spatial resolution by averaging the intensities of several neighboring pixels. Images which suffer from severe contouring when reducing colors can be improved with this option.

The **-colors** or **-monochrome** option is required for this option to take effect.

Use **+dither** to turn off dithering and to render PostScript without text or graphic aliasing.

-draw *<**string**>* annotate an image with one or more graphic primitives

Use this option to annotate an image with one or more graphic primitives. The primitives include shapes, text, transformations, and pixel operations. The shape primitives are

```
point           x,y
line            x0,y0 x1,y1
rectangle       x0,y0 x1,y1
roundRectangle  x0,y0 x1,y1 wc,hc
arc             x0,y0 x1,y1 a0,a1
ellipse         x0,y0 rx,ry a0,a1
circle          x0,y0 x1,y1
polyline        x0,y0  ...  xn,yn
polygon         x0,y0  ...  xn,yn
Bezier          x0,y0  ...  xn,yn
path            path specification
image           operator x0,y0 w,h filename
```

The text primitive is

```
text            x0,y0 string
```

The text gravity primitive is

```
gravity         NorthWest, North, NorthEast, West, Center,
                East, SouthWest, South, or SouthEast
```

The text gravity primitive only affects the placement of text and does not interact with the other primitives. It is equivalent to using the **-gravity** commandline option, except that it is limited in scope to the **-draw** option in which it appears.

The transformation primitives are

```
rotate          degrees
translate       dx,dy
scale           sx,sy
skewX           degrees
skewY           degrees
```

The pixel operation primitives are

```
color           x0,y0 method
matte           x0,y0 method
```

The shape primitives are drawn in the color specified in the preceding **-stroke** option. Except for the **line** and **point** primitives, they are filled with the color specified in the preceding **-fill** option. For unfilled shapes, use `-fill none`.

Point requires a single coordinate.

Line requires a start and end coordinate.

Rectangle expects an upper left and lower right coordinate.

RoundRectangle has the upper left and lower right coordinates and the width and height of the corners.

Circle has a center coordinate and a coordinate for the outer edge.

Use **Arc** to inscribe an elliptical arc within a rectangle. Arcs require a start and end point as well as the degree of rotation (e.g. 130,30 200,100 45,90).

Use **Ellipse** to draw a partial ellipse centered at the given point with the x-axis and y-axis radius and start and end of arc in degrees (e.g. 100,100 100,150 0,360).

Finally, **polyline** and **polygon** require three or more coordinates to define its boundaries. Coordinates are integers separated by an optional comma. For example, to define a circle centered at 100,100 that extends to 150,150 use:

```
-draw 'circle 100,100 150,150'
```

Paths (See Paths) represent an outline of an object which is defined in terms of moveto (set a new current point), lineto (draw a straight line), curveto (draw a curve using a cubic Bezier), arc (elliptical or circular arc) and closepath (close the current shape by drawing a line to the last moveto) elements. Compound paths (i.e., a path with subpaths, each consisting of a single moveto followed by one or more line or curve operations) are possible to allow effects such as “donut holes” in objects.

Use **image** to composite an image with another image. Follow the image keyword with the composite operator, image location, image size, and filename:

```
-draw 'image Over 100,100 225,225 image.jpg'
```

You can use 0,0 for the image size, which means to use the actual dimensions found in the image header. Otherwise, it will be scaled to the given dimensions. See **-compose** for a description of the composite operators.

Use **text** to annotate an image with text. Follow the text coordinates with a string. If the string has embedded spaces, enclose it in double quotes. Optionally you can include the image filename, type, width, height, or other image attribute by embedding special format character. See **-comment** for details.

For example,

```
-draw 'text 100,100 "%m:%f %wx%h"'
```

annotates the image with `MIFF:bird.miff 512x480` for an image titled `bird.miff` and whose width is 512 and height is 480.

If the first character of *string* is @, the text is read from a file titled by the remaining characters in the string.

Rotate rotates subsequent shape primitives and text primitives about the origen of the main image. If the **-region** option precedes the **-draw** option, the origen for transformations is the upper left corner of the region.

Translate translates them.

Scale scales them.

SkewX and **SkewY** skew them with respect to the origen of the main image or the region.

The transformations modify the current affine matrix, which is initialized from the initial affine matrix defined by the **-affine** option. Transformations are cumulative within the **-draw** option. The initial affine matrix is not affected; that matrix is only changed by the appearance of another **-affine** option. If another **-draw** option appears, the current affine matrix is reinitialized from the initial affine matrix.

Use **color** to change the color of a pixel to the fill color (see **-fill**). Follow the pixel coordinate with a method:

```
point
replace
floodfill
filltoborder
reset
```

Consider the target pixel as that specified by your coordinate. The **point** method recolors the target pixel. The **replace** method recolors any pixel that matches the color of the target pixel. **Floodfill** recolors any pixel that matches the color of the target pixel and is a neighbor, whereas **filltoborder** recolors any neighbor pixel that is not the border color. Finally, **reset** recolors all pixels.

Use **matte** to the change the pixel matte value to transparent. Follow the pixel coordinate with a method (see the **color** primitive for a description of methods). The **point** method changes the matte value of the target pixel. The **replace** method changes the matte value of any pixel that matches the color of the target pixel. **Floodfill** changes the matte value of any pixel that matches the color of the target pixel and is a neighbor, whereas **filltoborder** changes the matte value of any neighbor pixel that is not the border color (**-bordercolor**). Finally **reset** changes the matte value of all pixels.

You can set the primitive color, font, and font bounding box color with **-fill**, **-font**, and **-box** respectively. Options are processed in command line order so be sure to use these options *before* the **-draw** option.

-edge <***radius***> detect edges within an image

-emboss <***radius***> emboss an image

-encoding <***type***> specify the text encoding

Choose from *AdobeCustom, AdobeExpert, AdobeStandard, AppleRoman, BIG5, GB2312, Latin 2, None, SJIScode, Symbol, Unicode, Wansung.*

-endian <***type***> specify endianness (MSB or LSB) of output image

Use **+endian** to revert to unspecified endianness.

-enhance apply a digital filter to enhance a noisy image

-equalize perform histogram equalization to the image

-extract <***width***>***x***<***height***>{**+-**}<***x***>{**+-**}<***y***>{***%***}{@} {***!***}{<}{>} extract an area from the image while decoding

-fill <***color***> color to use when filling a graphic primitive

Colors are represented in ImageMagick in the same form used by SVG:

```
name                 ("convert -list color" to see names)
#RGB                 (R,G,B are hex numbers, 4 bits each)
#RRGGBB              (8 bits each)
#RRRGGGBBB           (12 bits each)
#RRRRGGGGBBBB        (16 bits each)
#RGBA                (4 bits each)
#RRGGBBAA            (8 bits each)
#RRRGGGBBBAAA        (12 bits each)
#RRRRGGGGBBBBAAAA    (16 bits each)
rgb(r,g,b)           (r,g,b are decimal numbers)
rgba(r,g,b,a)        (r,g,b,a are decimal numbers)
```

Enclose the color specification in quotation marks to prevent the "#" or the parentheses from being interpreted by your shell.

For example,

```
convert -fill blue ...
convert -fill "#ddddff" ...
convert -fill "rgb(65000,65000,65535)" ...
```

The shorter forms are scaled up, if necessary by replication. For example, #3af, #33aaff, and #3333aaaaffff are all equivalent.

See **-draw** for further details.

-filter *<type>* use this type of filter when resizing an image

Use this option to affect the resizing operation of an image (see **-geometry**). Choose from these filters:

```
Point
Box
Triangle
Hermite
Hanning
Hamming
Blackman
Gaussian
Quadratic
Cubic
Catrom
Mitchell
Lanczos
Bessel
Sinc
```

The default filter is **Lanczos**

-flatten flatten a sequence of images

The sequence of images is replaced by a single image created by composing each image after the first over the first image.

The sequence of images is terminated by the appearance of any option. If the **-flatten** option appears after all of the input images, all images are flattened.

-flip create a "mirror image"

reflect the scanlines in the vertical direction.

-flop create a “mirror image”

reflect the scanlines in the horizontal direction.

-font *<name>* use this font when annotating the image with text

You can tag a font to specify whether it is a PostScript, TrueType, or OPTION1 font. For example, `Arial.ttf` is a TrueType font, `ps:helvetica` is PostScript, and `x:fixed` is OPTION1.

-foreground *<color>* define the foreground color

The color is specified using the format described under the **-fill** option.

-format *<type>* the image format type

When used with the **mogrify** utility, this option will convert any image to the image format you specify. See *ImageMagick(1)* for a list of image format types supported by **ImageMagick**.

By default the file is written to its original name. However, if the filename extension matches a supported format, the extension is replaced with the image format type specified with **-format**. For example, if you specify *tiff* as the format type and the input image filename is *image.gif*, the output image filename becomes *image.tiff*.

-format *<string>* output formatted image characteristics

When used with the **identify** utility, use this option to print information about the image in a format of your choosing. You can include the image filename, type, width, height, Exif data, or other image attributes by embedding special format characters:

```
%b   file size
%c   comment
%d   directory
%e   filename extension
%f   filename
%g   page geometry
%h   height
%i   input filename
%k   number of unique colors
%l   label
%m   magick
%n   number of scenes
%o   output filename
```

```
%p   page number
%q   quantum depth
%s   scene number
%t   top of filename
%u   unique temporary filename
%w   width
%x   x resolution
%y   y resolution
%z   image depth
%#   signature
\n   newline
\r   carriage return
```

For example,

```
-format "%m:%f %wx%h"
```

displays **MIFF:bird.miff 512x480** for an image titled **bird.miff** and whose width is 512 and height is 480.

If the first character of *string* is **@**, the format is read from a file titled by the remaining characters in the string.

You can also use the following special formatting syntax to print Exif information contained in the file:

```
%[EXIF:<tag>]
```

Where "<tag>" can be one of the following:

```
*  (print all Exif tags, in keyword=data format)
!  (print all Exif tags, in tag_number data format)
#hhhh (print data for Exif tag #hhhh)
ImageWidth
ImageLength
BitsPerSample
Compression
PhotometricInterpretation
FillOrder
DocumentName
ImageDescription
Make
Model
StripOffsets
Orientation
SamplesPerPixel
RowsPerStrip
```

```
StripByteCounts
XResolution
YResolution
PlanarConfiguration
ResolutionUnit
TransferFunction
Software
DateTime
Artist
WhitePoint
PrimaryChromaticities
TransferRange
JPEGProc
JPEGInterchangeFormat
JPEGInterchangeFormatLength
YCbCrCoefficients
YCbCrSubSampling
YCbCrPositioning
ReferenceBlackWhite
CFARepeatPatternDim
CFAPattern
BatteryLevel
Copyright
ExposureTime
FNumber
IPTC/NAA
ExifOffset
InterColorProfile
ExposureProgram
SpectralSensitivity
GPSInfo
ISOSpeedRatings
OECF
ExifVersion
DateTimeOriginal
DateTimeDigitized
ComponentsConfiguration
CompressedBitsPerPixel
ShutterSpeedValue
ApertureValue
BrightnessValue
ExposureBiasValue
MaxApertureValue
SubjectDistance
MeteringMode
LightSource
```

```
Flash
FocalLength
MakerNote
UserComment
SubSecTime
SubSecTimeOriginal
SubSecTimeDigitized
FlashPixVersion
ColorSpace
ExifImageWidth
ExifImageLength
InteroperabilityOffset
FlashEnergy
SpatialFrequencyResponse
FocalPlaneXResolution
FocalPlaneYResolution
FocalPlaneResolutionUnit
SubjectLocation
ExposureIndex
SensingMethod
FileSource
SceneType
```

Surround the format specification with quotation marks to prevent your shell from misinterpreting any spaces and square brackets.

-frame <width>x<height>+<outer bevel width>+<inner bevel width>

-fx <expression> apply the mathematical expression an image or image channels.

For example, to extract the matte channel of the image (this is the negative to what is commonly thought of as the alpha channel mask of the image), use:

convert drawn.png -fx 'a' +matte matte.png

Mathematic operators include

constants MaxRGB, Opaque, Transparent, Pi standard operators: +, -, *, etc. math function name: abs(), acos(), asin(), atan(), cos(), exp(), log(), ln(), max(), min(), rand(), sin(), sqrt(), tan() symbols: u = first image in sequence v = second image in sequence i = the current column j = the current row p = pixel to use (absolute or relative to current pixel) w = width of this image h = height of this image r = red value (from RGBA), of a specific or current pixel g = green " b = blue " a = alpha " c = cyan value of CMYK color of pixel y = yellow " m = magenta " k = black " intensity = grayscale value

Specify the image source using an image index represented by 'u', starting at zero for the first image, (eg: 'u[3]' is the fourth image in the image sequence). A

negative image index counts images from the end of the current image sequence, therefore ‘u[-1]’ refers to the last image in the sequence.

Without an index ‘u’ or ‘v’ represent the first and second image of the sequence. If no image is specified, the ‘u’ image is used.

For example to reduce the intensity of the red channel by 50

convert image.png -channel red -fx ’u/2.0’ image.jpg

The pixels are processed one at a time, but a different pixel of a image can be specified with a pixel index represented by ‘p’. For example,

p[-1].g Green value of pixel to the immediate left of current

p[-1,-1].r Red value, diagonally left and up from current pixel

To specify an absolute position, use braces, rather than brackets

p12,34.b is the blue pixel at image location 12,34

The other symbols specify the value you wish to retrieve.

A pixel outside the boundary of the image has a value dictated by the -virtual-pixel option setting.

The -channel setting can be used to specify the output channel of the result. If no output channel is given the result is set over all RGBA channels. For example, suppose you want to replace the red channel of alpha.png with the average of the green channels from the images alpha.png and beta.png, use:

convert alpha.png beta.png -channel red
-fx ’(u.g+v.g)/2’ gamma.png

Note that all the original images in the current image sequence are replaced by the updated ‘alpha.png’ image. surround the image with an ornamental border

See **-geometry** for details about the geometry specification. The **-frame** option is not affected by the **-gravity** option.

The color of the border is specified with the **-mattecolor** command line option.

-frame include the X window frame in the imported image

-fuzz *<**distance**>{**%**}* colors within this distance are considered equal

A number of algorithms search for a target color. By default the color must be exact. Use this option to match colors that are close to the target color in RGB space. For example, if you want to automatically trim the edges of an image with **-trim** but the image was scanned and the target background color may differ by a small amount. This option can account for these differences.

The *distance* can be in absolute intensity units or, by appending *“%”*, as a percentage of the maximum possible intensity (255 or 65535).

-fx *<**fx-image**>* *<**expression**>* Rapplies a mathematical expression to the specified image channel(s).

-gamma *<**value**>* level of gamma correction

The same color image displayed on two different workstations may look different due to differences in the display monitor. Use gamma correction to adjust for this color difference. Reasonable values extend from **0.8** to **2.3**. Gamma less than 1.0 darkens the image and gamma greater than 1.0 lightens it.

You can apply separate gamma values to the red, green, and blue channels of the image with a gamma value list delimited with slashes (e.g., **1.7**/**2.3**/**1.2**).

Use **+gamma** *value* to set the image gamma level without actually adjusting the image pixels. This option is useful if the image is of a known gamma but not set as an image attribute (e.g. PNG images).

-Gaussian *<**radius**>*{***x****<**sigma**>*} blur the image with a Gaussian operator

Use the given radius and standard deviation (sigma).

-geometry *<**width**>**x**<**height**>*{**+-**}*<**x**>*{**+-**}*<**y**>*{**%**}{@} {***!***}{<}{>} preferred size and location of the Image window.

By default, the window size is the image size and the location is chosen by you when it is mapped.

By default, the width and height are maximum values. That is, the image is expanded or contracted to fit the width and height value while maintaining the aspect ratio of the image. *Append an exclamation point to the geometry to force the image size to exactly the size you specify*. For example, if you specify `640x480!` the image width is set to 640 pixels and height to 480.

If only the width is specified, the width assumes the value and the height is chosen to maintain the aspect ratio of the image. Similarly, if only the height is specified (e.g., `-geometry x256`), the width is chosen to maintain the aspect ratio.

To specify a percentage width or height instead, append %. The image size is multiplied by the width and height percentages to obtain the final image dimensions. To increase the size of an image, use a value greater than 100 (e.g. 125%). To decrease an image's size, use a percentage less than 100.

Use @ to specify the maximum area in pixels of an image.

Use > to change the dimensions of the image *only* if its width or height exceeds the geometry specification. < resizes the image *only* if both of its dimensions are less than the geometry specification. For example, if you specify `'640x480>'` and the image size is 256x256, the image size does not change. However, if the

image is 512x512 or 1024x1024, it is resized to 480x480. Enclose the geometry specification in quotation marks to prevent the < or > from being interpreted by your shell as a file redirection.

When used with *animate* and *display*, offsets are handled in the same manner as in *X(1)* and the **-gravity** option is not used. If the *x* is negative, the offset is measured leftward from the right edge of the screen to the right edge of the image being displayed. Similarly, negative *y* is measured between the bottom edges. The offsets are not affected by "%"; they are always measured in pixels.

When used as a *composite* option, **-geometry** gives the dimensions of the image and its location with respect to the composite image. If the **-gravity** option is present with *NorthEast*, *East*, or *SouthEast* gravity, the *x* represents the distance from the right edge of the image to the right edge of the composite image. Similarly, if the **-gravity** option is present with *SouthWest*, *South*, or *SouthEast* gravity, *y* is measured between the bottom edges. Accordingly, a positive offset will never point in the direction outside of the image. The offsets are not affected by "%"; they are always measured in pixels. To specify the dimensions of the composite image, use the **-resize** option.

When used as a *convert*, *import* or *mogrify* option, **-geometry** is synonymous with **-resize** and specifies the size of the output image. The offsets, if present, are ignored.

When used as a *montage* option, **-geometry** specifies the image size and border size for each tile; default is 256x256+0+0. Negative offsets (border dimensions) are meaningless. The **-gravity** option affects the placement of the image within the tile; the default gravity for this purpose is *Center*. If the "%" sign appears in the geometry specification, the tile size is the specified percentage of the original dimensions of the first tile. To specify the dimensions of the montage, use the **-resize** option.

-gravity <type> direction primitive gravitates to when annotating the image.

Choices are: NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast.

The direction you choose specifies where to position the text when annotating the image. For example *Center* gravity forces the text to be centered within the image. By default, the image gravity is *NorthWest*. See **-draw** for more details about graphic primitives. Only the text primitive is affected by the **-gravity** option.

The **-gravity** option is also used in concert with the **-geometry** option and other options that take <**geometry**> as a parameter, such as the **-crop** option. See **-geometry** for details of how the **-gravity** option interacts with the <**x**> and <**y**> parameters of a geometry specification.

When used as an option to *composite*, **-gravity** gives the direction that the image gravitates within the composite.

When used as an option to *montage*, **-gravity** gives the direction that an image gravitates within a tile. The default gravity is *Center* for this purpose.

-green-primary *<x>,<y>* green chromaticity primary point

-help print usage instructions

-iconGeometry *<geometry>* specify the icon geometry

Offsets, if present in the geometry specification, are handled in the same manner as the **-geometry** option, using X11 style to handle negative offsets.

-iconic iconic animation

-immutable make image immutable

-implode *<factor>* implode image pixels about the center

-intent *<type>* use this type of rendering intent when managing the image color

Use this option to affect the the color management operation of an image (see **-profile**). Choose from these intents: **Absolute, Perceptual, Relative, Saturation**

The default intent is undefined.

-interlace *<type>* the type of interlacing scheme

Choices are: **None, Line, Plane,** or **Partition**. The default is **None**.

This option is used to specify the type of interlacing scheme for raw image formats such as **RGB** or **YUV**.

None means do not interlace (RGBRGBRGBRGBRGBRGB...),

Line uses scanline interlacing (RRR...GGG...BBB...RRR...GGG...BBB...), and

Plane uses plane interlacing (RRRRRR...GGGGGG...BBBBBB...).

Partition is like plane except the different planes are saved to individual files (e.g. image.R, image.G, and image.B).

Use **Line** or **Plane** to create an **interlaced PNG** or **GIF** or **progressive JPEG** image.

-label *<**name**>* assign a label to an image

Use this option to assign a specific label to the image, when writing to an image format that supports labels, such as TIFF, PNG, MIFF, or PostScript. You can include the the image filename, type, width, height, or other image attribute by embedding special format character. A label is not drawn on the image, but is embedded in the image datastream via a "Label" tag or similar mechanism. If you want the label to be visible on the image itself, use the **-draw** option. See **-comment** for details.

For example,

```
-label "%m:%f %wx%h"
```

produces an image label of **MIFF:bird.miff 512x480** for an image titled **bird.miff** and whose width is 512 and height is 480.

If the first character of *string* is @, the image label is read from a file titled by the remaining characters in the string.

When converting to *PostScript*, use this option to specify a header string to print above the image. Specify the label font with **-font**.

When creating a montage, by default the label associated with an image is displayed with the corresponding tile in the montage. Use the **+label** option to suppress this behavior.

-lat *<**width**>**x**<**height**>*{**+-**}*<**offset**>*{**%**} perform local adaptive thresholding

Perform local adaptive thresholding using the specified width, height, and offset. The offset is a distance in sample space from the mean, as an absolute integer ranging from 0 to the maximum sample value or as a percentage.

-level *<**black_point**>*{**,***<**white_point**>*}{**%**}{**,***<**gamma**>*} adjust the level of image contrast

Give one, two or three values delimited with commas: black, white, and gamma (e.g. 10,65000,1.0 or 2%,98%,0.5). The black and white points range from 0 to MaxRGB or from 0 to 100%; if the white point is omitted it is set to MaxRGB-black_point. If a "%" sign is present anywhere in the string, the black and white points are percentages of MaxRGB. Gamma is an exponent that ranges from 0.1 to 10.; if it is omitted, the default of 1.0 (no gamma correction) is assumed.

-limit *<**type**>* *<**value**>* Area, Disk, Map, or Memory resource limit

The value for Area is in number of Megabytes and the values for the other resources are in Megabytes. By default the limits are 64 Megabytes area, 512MB

memory, 1024MB map, and unlimited disk, but these are adjusted at startup time on platforms that can provide information about available resources. When the limit is reached, ImageMagick will fail in some fashion, or take compensating actions if possible. For example, `-limit memory 32 -limit map 64` limits memory When the pixel cache reaches the memory limit it uses memory mapping. When that limit is reached it goes to disk. If disk has a hard limit, the program will fail.

You can use the option `-list resource` to find out the limits. This will also show the number of files available, which is not changeable via the -limit option.

-linewidth the line width for subsequent draw operations

-list *<type>* the type of list

Choices are: **Delegate**, **Format**, **Magic**, **Module**, **Resource**, or **Type**.

This option lists information about the ImageMagick configuration.

-log *<string>* This option specifies the format for the log printed when the **-debug** option is active.

You can display the following components by embedding special format characters:

```
%d   domain
%e   event
%f   function
%l   line
%m   module
%p   process ID
%r   real CPU time
%t   wall clock time
%u   user CPU time
%%   percent sign
\n   newline
\r   carriage return
```

For example:

```
convert -debug coders -log "%u %m:%l %e" in.gif out.png
```

The default behavior is to print all of the components.

-loop *<iterations>* add Netscape loop extension to your GIF animation

A value other than zero forces the animation to repeat itself up to *iterations* times.

-magnify <***factor***> magnify the image

-map <***filename***> choose a particular set of colors from this image

[*convert* or *mogrify*]

By default, color reduction chooses an optimal set of colors that best represent the original image. Alternatively, you can choose a particular set of colors from an image file with this option.

Use **+map** to reduce all images in the image sequence that follows to a single optimal set of colors that best represent all the images. The sequence of images is terminated by the appearance of any option. If the **+map** option appears after all of the input images, all images are mapped.

-map <***type***> display image using this type.

[*animate* or *display*]

Choose from these *Standard Colormap* types:

```
best
default
gray
red
green
blue
```

The *X server* must support the *Standard Colormap* you choose, otherwise an error occurs. Use **list** as the type and **display** searches the list of colormap types in **top-to-bottom** order until one is located. See *xstdcmap(1)* for one way of creating Standard Colormaps.

-mask <***filename***> Specify a clipping mask

The image read from the file is used as a clipping mask. It must have the same dimensions as the image being masked.

If the mask image contains an alpha channel, the opacity of each pixel is used to define the mask. Otherwise, the intensity (gray level) of each pixel is used.

Use **+mask** to remove the clipping mask.

It is not necessary to use **-clip** to activate the mask; **-clip** is implied by **-mask**.

-matte store matte channel if the image has one

If the image does not have a matte channel, create an opaque one.

Use **+matte** to ignore the matte channel and to avoid writing a matte channel in the output file.

-mattecolor *<color>* specify the color to be used with the **-frame** option

The color is specified using the format described under the **-fill** option.

-median *<radius>* apply a median filter to the image

-mode *<value>* mode of operation

-modulate *<value>* vary the brightness, saturation, and hue of an image

Specify the percent change in brightness, the color saturation, and the hue separated by commas. For example, to increase the color brightness by 20% and decrease the color saturation by 10% and leave the hue unchanged, use: **-modulate 120,90**.

-monochrome transform the image to black and white

-morph *<frames>* morphs an image sequence

Both the image pixels and size are linearly interpolated to give the appearance of a meta-morphosis from one image to the next.

The sequence of images is terminated by the appearance of any option. If the **-morph** option appears after all of the input images, all images are morphed.

-mosaic create a mosaic from an image or an image sequence

The **-page** option can be used to establish the dimensions of the mosaic and to locate the images within the mosaic.

The sequence of images is terminated by the appearance of any option. If the **-mosaic** option appears after all of the input images, all images are included in the mosaic.

-name name an image

-negate replace every pixel with its complementary color

The red, green, and blue intensities of an image are negated. White becomes black, yellow becomes blue, etc. Use **+negate** to only negate the grayscale pixels of the image.

-noise <***radius***|***type***> add or reduce noise in an image

The principal function of noise peak elimination filter is to smooth the objects within an image without losing edge information and without creating undesired structures. The central idea of the algorithm is to replace a pixel with its next neighbor in value within a pixel window, if this pixel has been found to be noise. A pixel is defined as noise if and only if this pixel is a maximum or minimum within the pixel window.

Use **radius** to specify the width of the neighborhood.

Use **+noise** followed by a noise type to add noise to an image. Choose from these noise types:

```
Uniform
Gaussian
Multiplicative
Impulse
Laplacian
Poisson
```

-noop NOOP (no option)

The **-noop** option can be used to terminate a group of images and reset all options to their default values, when no other option is desired.

-normalize transform image to span the full range of color values

This is a contrast enhancement technique.

-opaque <***color***> change this color to the pen color within the image

The color is specified using the format described under the **-fill** option.

See **-fill** for more details.

-page <***width***>***x***<***height***>{***+-***}<***x***>{***+-***}<***y***>{***%***}{***!***}{<}{>} size and location of an image canvas

Use this option to specify the dimensions of the *PostScript* page in dots per inch or a TEXT page in pixels. The choices for a PostScript page are:

```
11x17          792  1224
Ledger        1224   792
Legal          612  1008
Letter         612   792
LetterSmall    612   792
ArchE         2592  3456
ArchD         1728  2592
ArchC         1296  1728
ArchB          864  1296
ArchA          648   864
A0            2380  3368
A1            1684  2380
A2            1190  1684
A3             842  1190
A4             595   842
A4Small        595   842
A5             421   595
A6             297   421
A7             210   297
A8             148   210
A9             105   148
A10             74   105
B0            2836  4008
B1            2004  2836
B2            1418  2004
B3            1002  1418
B4             709  1002
B5             501   709
C0            2600  3677
C1            1837  2600
C2            1298  1837
C3             918  1298
C4             649   918
C5             459   649
C6             323   459
Flsa           612   936
Flse           612   936
HalfLetter     396   612
```

For convenience you can specify the page size by media (e.g. A4, Ledger, etc.). Otherwise, **-page** behaves much like **-geometry** (e.g. `-page letter+43+43`>).

This option is also used to place subimages when writing to a multi-image format that supports offsets, such as GIF89 and MNG. When used for this purpose the offsets are always measured from the top left corner of the canvas and are not affected by the **-gravity** option. To position a GIF or MNG image, use **-page**{+-}<*x*>{+-}<*y*> (e.g. -page +100+200). When writing to a MNG file, a **-page**

option appearing ahead of the first image in the sequence with nonzero width and height defines the width and height values that are written in the **MHDR** chunk. Otherwise, the MNG width and height are computed from the bounding box that contains all images in the sequence. When writing a GIF89 file, only the bounding box method is used to determine its dimensions.

For a PostScript page, the image is sized as in **-geometry** and positioned relative to the lower left hand corner of the page by {+-}<**x***offset*>{+-}<**y** *offset*>. Use `-page 612x792>`, for example, to center the image within the page. If the image size exceeds the PostScript page, it is reduced to fit the page. The default gravity for the **-page** option is *NorthWest*, i.e., positive **x** and **y** *offset* are measured rightward and downward from the top left corner of the page, unless the **-gravity** option is present with a value other than *NorthWest*.

The default page dimensions for a TEXT image is 612x792.

This option is used in concert with **-density**.

Use **+page** to remove the page settings for an image.

-paint <***radius***> simulate an oil painting

Each pixel is replaced by the most frequent color in a circular neighborhood whose width is specified with *radius*.

-pause <***seconds***> pause between animation loops [animate]

Pause for the specified number of seconds before repeating the animation.

-pause <***seconds***> pause between snapshots [import]

Pause for the specified number of seconds before taking the next snapshot.

-pen <***color***> (This option has been replaced by the -fill option)

-ping efficiently determine image characteristics

-pointsize <***value***> pointsize of the PostScript, OPTION1, or TrueType font

-preview <***type***> image preview type

Use this option to affect the preview operation of an image (e.g. `convert file.png -preview Gamma Preview:gamma.png`). Choose from these previews:

```
Rotate
Shear
Roll
Hue
Saturation
Brightness
Gamma
Spiff
Dull
Grayscale
Quantize
Despeckle
ReduceNoise
Add Noise
Sharpen
Blur
Threshold
EdgeDetect
Spread
Shade
Raise
Segment
Solarize
Swirl
Implode
Wave
OilPaint
CharcoalDrawing
JPEG
```

The default preview is **JPEG**.

-process *<**command**>* process a sequence of images using a process module

The command argument has the form `module=arg1,arg2,arg3,...,argN` where `module`¿ is the name of the module to invoke (e.g. ”analyze”) and arg1,arg2,arg3,...,argN are an arbitrary number of arguments to pass to the process module.

The sequence of images is terminated by the appearance of any option.

If the **-process** option appears after all of the input images, all images are processed.

-profile *<**filename**>* add ICM, IPTC, or generic profile to image

`-profile filename` adds an ICM (ICC color management), IPTC (newswire information), or a generic profile to the image.

Use `+profile icm`, `+profile iptc`, or `+profile profile_name` to remove the respective profile. Use `identify -verbose` to find out what profiles are in the image file. Use `+profile "*"` to remove all profiles.

To extract a profile, the **-profile** option is not used. Instead, simply write the file to an image format such as *APP1, 8BIM, ICM,* or *IPTC*.

For example, to extract the Exif data (which is stored in JPEG files in the *APP1* profile), use

```
convert cockatoo.jpg exifdata.app1
```

-quality *<**value**>* JPEG/MIFF/PNG compression level

For the JPEG and MPEG image formats, quality is 0 (lowest image quality and highest compression) to 100 (best quality but least effective compression). The default quality is 75. Use the **-sampling-factor** option to specify the factors for chroma downsampling.

For the MIFF image format, quality/10 is the zlib compression level, which is 0 (worst but fastest compression) to 9 (best but slowest). It has no effect on the image appearance, since the compression is always lossless.

For the MNG and PNG image formats, the quality value sets the zlib compression level (quality / 10) and filter-type (quality % 10). Compression levels range from 0 (fastest compression) to 100 (best but slowest). For compression level 0, the Huffman-only strategy is used, which is fastest but not necessarily the worst compression.

If filter-type is 4 or less, the specified filter-type is used for all scanlines:

```
0: none
1: sub
2: up
3: average
4: Paeth
```

If filter-type is 5, adaptive filtering is used when quality is greater than 50 and the image does not have a color map, otherwise no filtering is used.

If filter-type is 6, adaptive filtering with *minimum-sum-of-absolute-values* is used.

Only if the output is MNG, if filter-type is 7, the LOCO color transformation and adaptive filtering with *minimum-sum-of-absolute-values* are used.

The default is quality is 75, which means nearly the best compression with adaptive filtering. The quality setting has no effect on the appearance of PNG and MNG images, since the compression is always lossless.

For further information, see the PNG specification.

When writing a JNG image with transparency, two quality values are required, one for the main image and one for the grayscale image that conveys the alpha channel. These are written as a single integer equal to the main image quality plus 1000 times the opacity quality. For example, if you want to use quality 75 for the main image and quality 90 to compress the opacity data, use `-quality 90075`.

-raise *<width>x<height>* lighten or darken image edges

This will create a 3-D effect. See **-geometry** for details details about the geometry specification. Offsets are not used.

Use **-raise** to create a raised effect, otherwise use **+raise**.

-red-primary *<x>,<y>* red chromaticity primary point

-region *<width>x<height>{+-}<x>{+-}<y>* apply options to a portion of the image

The *x* and *y* offsets are treated in the same manner as in **-crop**.

-remote perform a remote operation

The only command recognized at this time is the name of an image file to load.

-render render vector operations

Use **+render** to turn off rendering vector operations.

-resize *<width>x<height>{%}{@}{!}{<}{>}* resize an image

This is an alias for the **-geometry** option and it behaves in the same manner. If the **-filter** option precedes the **-resize** option, the specified filter is used.

There are some exceptions:

When used as a *composite* option, **-resize** conveys the preferred size of the output image, while **-geometry** conveys the size and placement of the *composite image* within the main image.

When used as a *montage* option, **-resize** conveys the preferred size of the montage, while **-geometry** conveys information about the tiles.

-roll *{+-}<x>{+-}<y>* roll an image vertically or horizontally

See **-geometry** for details the geometry specification. The *x* and *y* offsets are not affected by the **-gravity** option.

A negative *x* offset rolls the image left-to-right. A negative *y* offset rolls the image top-to-bottom.

-rotate *<**degrees**>{<}{>}* apply Paeth image rotation to the image

Use > to rotate the image only if its width exceeds the height. < rotates the image *only* if its width is less than the height. For example, if you specify `-rotate "-90>"` and the image size is 480x640, the image is not rotated. However, if the image is 640x480, it is rotated by -90 degrees. If you use > or <, enclose it in quotation marks to prevent it from being misinterpreted as a file redirection.

Empty triangles left over from rotating the image are filled with the color defined as **background** (class **backgroundColor**). The color is specified using the format described under the **-fill** option.

-sample *<**geometry**>* scale image with pixel sampling

See **-geometry** for details about the geometry specification. **-sample** ignores the **-filter** selection if the **-filter** option is present. Offsets, if present in the geometry string, are ignored, and the **-gravity** option has no effect.

-sampling-factor *<**horizontal_factor**>**x**<**vertical_factor**>* sampling factors used by JPEG or MPEG-2 encoder and YUV decoder/encoder.

This option specifies the sampling factors to be used by the JPEG encoder for chroma downsampling. If this option is omitted, the JPEG library will use its own default values. When reading or writing the YUV format and when writing the M2V (MPEG-2) format, use **-sampling-factor 2x1** to specify the 4:2:2 downsampling method.

-scale *<**geometry**>* scale the image.

See **-geometry** for details about the geometry specification. **-scale** uses a simpler, faster algorithm, and it ignores the **-filter** selection if the **-filter** option is present. Offsets, if present in the geometry string, are ignored, and the **-gravity** option has no effect.

-scene *<**value**>* set scene number

This option sets the scene number of an image or the first image in an image sequence.

-scenes *<**value-value**>* range of image scene numbers to read

Each image in the range is read with the filename followed by a period (**.**) and the decimal scene number. You can change this behavior by embedding a **%d,**

%0Nd, %o, %0No, %x, or %0Nx printf format specification in the file name. For example,

```
montage -scenes 5-7 image.miff
```

makes a montage of files image.miff.5, image.miff.6, and image.miff.7, and

```
animate -scenes 0-12 image%02d.miff
```

animates files image00.miff, image01.miff, through image12.miff.

-screen specify the screen to capture

This option indicates that the GetImage request used to obtain the image should be done on the root window, rather than directly on the specified window. In this way, you can obtain pieces of other windows that overlap the specified window, and more importantly, you can capture menus or other popups that are independent windows but appear over the specified window.

-seed *<value>* pseudo-random number generator seed value

The value can be any integer in the range 1 to 2**31-1. Successive runs with a particular seed will generate the same sequence of pseudo-random numbers. If the **-seed** option is not present, ImageMagick will generate a random seed from system timers, clocks, etc., so that successive runs will generate different sequences. The pseudo-random numbers are used by options such as **-noise**, **-spread**, and the **plasma** format.

-segment *<cluster threshold>x<smoothing threshold>* segment an image

Segment an image by analyzing the histograms of the color components and identifying units that are homogeneous with the fuzzy c-means technique.

Specify *cluster threshold* as the number of pixels in each cluster must exceed the the cluster threshold to be considered valid. *Smoothing threshold* eliminates noise in the second derivative of the histogram. As the value is increased, you can expect a smoother second derivative. The default is 1.5. See “Image Segmentation” in the manual page for *display* for details.

-shade *<azimuth>x<elevation>* shade the image using a distant light source

Specify *azimuth* and *elevation* as the position of the light source. Use **+shade** to return the shading results as a grayscale image.

-shadow shadow the montage

-shared-memory use shared memory

This option specifies whether the utility should attempt use shared memory for pixmaps. ImageMagick must be compiled with shared memory support, and the display must support the *MIT-SHM* extension. Otherwise, this option is ignored. The default is **True**.

-sharpen *<radius>*{***x***<*sigma*>} sharpen the image

Use a Gaussian operator of the given radius and standard deviation (sigma).

-shave *<width>*x*<height>*{***%***} shave pixels from the image edges

Specify the width of the region to be removed from both sides of the image and the height of the regions to be removed from top and bottom.

-shear *<x degrees>*x*<y degrees>* shear the image along the X or Y axis

Use the specified positive or negative shear angle.

Shearing slides one edge of an image along the X or Y axis, creating a parallelogram. An X direction shear slides an edge along the X axis, while a Y direction shear slides an edge along the Y axis. The amount of the shear is controlled by a shear angle. For X direction shears, *x degrees* is measured relative to the Y axis, and similarly, for Y direction shears *y degrees* is measured relative to the X axis.

Empty triangles left over from shearing the image are filled with the color defined as **background** (class **backgroundColor**). The color is specified using the format described under the **-fill** option.

-silent operate silently

-size *<width>*x*<height>*{***+offset***} width and height of the image

Use this option to specify the width and height of raw images whose dimensions are unknown such as **GRAY**, **RGB**, or **CMYK**. In addition to width and height, use **-size** with an offset to skip any header information in the image or tell the number of colors in a **MAP** image file, (e.g. -size 640x512+256).

For Photo CD images, choose from these sizes:

```
192x128
384x256
768x512
1536x1024
3072x2048
```

Finally, use this option to choose a particular resolution layer of a JBIG or JPEG image (e.g. -size 1024x768).

-snaps *<value>* number of screen snapshots

Use this option to grab more than one image from the X server screen, to create an animation sequence.

-solarize *<factor>* negate all pixels above the threshold level

Specify *factor* as the percent threshold of the intensity (0 - 99.9%).

This option produces a *solarization* effect seen when exposing a photographic film to light during the development process.

-spread *<amount>* displace image pixels by a random amount

Amount defines the size of the neighborhood around each pixel to choose a candidate pixel to swap.

-stegano *<offset>* hide watermark within an image

Use an offset to start the image hiding some number of pixels from the beginning of the image. Note this offset and the image size. You will need this information to recover the steganographic image (e.g. display -size 320x256+35 stegano:image.png).

-stereo composite two images to create a stereo anaglyph

The left side of the stereo pair is saved as the red channel of the output image. The right side is saved as the green channel. Red-green stereo glasses are required to properly view the stereo image.

-stroke *<color>* color to use when stroking a graphic primitive

The color is specified using the format described under the **-fill** option.

See **-draw** for further details.

-strokewidth *<value>* set the stroke width

See **-draw** for further details.

-swap *<index,index>* swap two images in the image sequence.

-swirl *<**degrees**>* swirl image pixels about the center

Degrees defines the tightness of the swirl.

-text-font *<**name**>* font for writing fixed-width text

Specifies the name of the preferred font to use in fixed (typewriter style) formatted text. The default is 14 point *Courier*.

You can tag a font to specify whether it is a PostScript, TrueType, or OPTION1 font. For example, `Courier.ttf` is a TrueType font and `x:fixed` is OPTION1.

-texture *<**filename**>* name of texture to tile onto the image background

-threshold *<**value**>*{*<**green**>*,*<**blue**>*,*<**opacity**>*}{***%***} threshold the image

Create an image such that any pixel sample that is equal or exceeds the threshold is reassigned the maximum intensity otherwise the minimum intensity.

If the green or blue value is omitted, these channels use the same value as the first one provided. If all three color values are the same, the result is a bi-level image. If the opacity threshold is omitted, OpaqueOpacity will be used and any partially transparent pixel will become fully transparent. If only a single 0 is provided, auto-thresholding will be performed.

To generate an all-black or all-white image with the same dimensions as the input image, you can use

```
convert -threshold 100% in.png black.png
convert -threshold -1 in.png white.png
```

-tile *<**filename**>* tile image when filling a graphic primitive

-tile *<**geometry**>* layout of images [*montage*]

-title *<**string**>* assign title to displayed image [*animate, display, montage*]

Use this option to assign a specific title to the image. This is assigned to the image window and is typically displayed in the window title bar. Optionally you can include the image filename, type, width, height, Exif data, or other image attribute by embedding special format characters described under the **-format** option.

For example,

```
-title "%m:%f %wx%h"
```

produces an image title of `MIFF:bird.miff 512x480` for an image titled `bird.miff` and whose width is 512 and height is 480.

-swap *<index,index>* swap two images in the image sequence.

-transform transform the image

This option applies the transformation matrix from a previous **-affine** option.

```
convert -affine 2,2,-2,2,0,0 -transform bird.ppm bird.jpg
```

-transparent *<color>* make this color transparent within the image

The color is specified using the format described under the **-fill** option.

-treedepth *<value>* tree depth for the color reduction algorithm

Normally, this integer value is zero or one. A zero or one tells display to choose an optimal tree depth for the color reduction algorithm

An optimal depth generally allows the best representation of the source image with the fastest computational speed and the least amount of memory. However, the default depth is inappropriate for some images. To assure the best representation, try values between 2 and 8 for this parameter. Refer to quantize for more details.

The **-colors** or **-monochrome** option is required for this option to take effect.

-trim trim an image

This option removes any edges that are exactly the same color as the corner pixels. Use **-fuzz** to make **-trim** remove edges that are nearly the same color as the corner pixels.

-type *<type>* the image type

Choose from: **Bilevel**, **Grayscale**, **Palette**, **PaletteMatte**, **TrueColor**, **TrueColorMatte**, **ColorSeparation**, **ColorSeparationMatte**, or **Optimize**.

Normally, when a format supports different subformats such as grayscale and truecolor, the encoder will try to choose an efficient subformat. The **-type** option can be used to overrride this behavior. For example, to prevent a JPEG from being written in grayscale format even though only gray pixels are present, use

```
convert bird.pgm -type TrueColor bird.jpg
```

Similarly, using `-type TrueColorMatte` will force the encoder to write an alpha channel even though the image is opaque, if the output format supports transparency.

-update <***seconds***> detect when image file is modified and redisplay.

Suppose that while you are displaying an image the file that is currently displayed is over-written. **display** will automatically detect that the input file has been changed and update the displayed image accordingly.

-units <***type***> the type of image resolution

Choose from: **Undefined**, **PixelsPerInch**, or **PixelsPerCentimeter**.

-unsharp <***radius***>{***x***<***sigma***>}{**+**<***amount***>}{**+**<***threshold***>} sharpen the image with an unsharp mask operator

The **-unsharp** option sharpens an image. We convolve the image with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, radius should be larger than sigma. Use a radius of 0 to have the method select a suitable radius.

The parameters are:

```
radius:    The radius of the Gaussian, in pixels,  not
           counting the center pixel (default 0).
sigma:     The standard deviation of the Gaussian, in
           pixels (default 1.0).
amount:    The percentage of the difference between the
           original and the blur image that is added back
           into the original (default 1.0).
threshold: The threshold, as a fraction of MaxRGB, needed
           to apply the difference amount (default 0.05).
```

-use-pixmap use the pixmap

-verbose print detailed information about the image

This information is printed: image scene number; image name; image size; the image class (*DirectClass* or *PseudoClass*); the total number of unique colors; and the number of seconds to read and transform the image. Refer to miff for a description of the image class.

If **-colors** is also specified, the total unique colors in the image and color reduction error values are printed. Refer to quantize for a description of these values.

-version print ImageMagick version string

-view <***string***> FlashPix viewing parameters

-virtual-pixel <***method***> specify contents of “virtual pixels”

This option defines “virtual pixels” for use in operations that can access pixels outside the boundaries of an image.

Choose from these methods:

```
Constant:  Use the image background color.
Edge:      Extend the edge pixel toward infinity (default).
Mirror:    Mirror the image.
Tile:      Tile the image.
```

This option affects operations that use virtual pixels such as **-blur**, **-sharpen**, **-wave**, etc.

-visual <***type***> animate images using this X visual type

Choose from these visual classes:

```
StaticGray
GrayScale
StaticColor
PseudoColor
TrueColor
DirectColor
default
visual id
```

The X server must support the visual you choose, otherwise an error occurs. If a visual is not specified, the visual class that can display the most simultaneous colors on the default screen is chosen.

-watermark <***brightness***>***x***<***saturation***> percent brightness and saturation of a watermark

-wave <***amplitude***>***x***<***wavelength***> alter an image along a sine wave

Specify *amplitude* and *wavelength* of the wave.

-white-point *<x>,<y>* chromaticity white point

-window *<id>* make image the background of a window

id can be a window id or name. Specify **root** to select X's root window as the target window.

By default the image is tiled onto the background of the target window. If **backdrop** or **-geometry** are specified, the image is surrounded by the background color. Refer to **X RESOURCES** for details.

The image will not display on the root window if the image has more unique colors than the target window colormap allows. Use **-colors** to reduce the number of colors.

-window-group specify the window group

-write *<filename>* write an image sequence [*convert*, *composite*]

The image sequence following the **-write** *filename*option is written out, and then processing continues with the same image in its current state if there are additional options. To restore the image to its original state after writing it, use the **+write** *filename* option.

-write *<filename>* write the image to a file [*display*]

If *filename* already exists, you will be prompted as to whether it should be overwritten.

By default, the image is written in the format that it was read in as. To specify a particular image format, prefix *filename* with the image type and a colon (e.g., ps:image) or specify the image type as the filename suffix (e.g., image.ps). See convert(1) for a list of valid image formats. Specify file as - for standard output. If file has the extension **.Z** or **.gz**, the file size is **compressed** using compress or **gzip** respectively. Precede the image file name with | to pipe to a system command.

Use **-compress** to specify the type of image compression.

The equivalent X resource for this option is **writeFilename** (class **WriteFilename**). See "X Resources" in the manual page for *display* for details.

20 API Structures and Enumerations

20.1 API Structures

AffineMatrix The members of the AffineMatrix structure are shown in the following table:

Table20.1: AffineMatrix Structure

AffineMatrix Structure

Member	Type	Description
sx	*double*	x scale.
sy	*double*	y scale.
rx	*double*	x rotate.
ry	*double*	y rotate.
tx	*double*	x translate.
ty	*double*	y translate.

ChromaticityInfo The members of the ChromaticityInfo structure are shown in the following table. The structure can contain either (x,y) or, if Z is nonzero, CIE (X,Y,Z) points.

Table20.2: ChromaticityInfo Structure

ChromaticityInfo Structure

Member	Type	Description
red_primary	*PrimaryInfo*	x,y or X,Y,Z of red primary.
green_primary	*PrimaryInfo*	x,y or X,Y,Z of green primary.
blue_primary	*PrimaryInfo*	x,y or X,Y,Z of blue primary.
white_point	*PrimaryInfo*	x,y or X,Y,Z of white point.

DrawInfo The DrawInfo structure is used to support annotating an image using drawing commands.

The members of the DrawInfo structure are shown in the following table. The structure is initialized to reasonable defaults by first initializing the equivalent members of ImageInfo, and then initializing the entire structure using GetDrawInfo().

Table20.3: DrawInfo Structure

DrawInfo Structure

Member	Type	Description
affine	*AffineMatrix*	Coordinate transformation (rotation, scaling, and translation).
align	*AlignType*	Alignment type.
border_color	*PixelPacket*	Border color.
bounds	*SegmentInfo*	Bounds.
box	*PixelPacket*	Text solid background color.
compose	*CompositeOperator*	Composite operator.
clip_path	*char **	Clipping path.
clip_units	*ClipPathUnits*	Clipping path units.
dash_offset	*double*	Dash offset.
dash_pattern	*double*	Dash pattern.
decorate	*DecorationType*	Text decoration type.
density	*char **	Text rendering density in DPI (effects scaling font according to pointsize). E.g. “72x72”.

DrawInfo Structure (continued)

Member	Type	Description
element_reference	*ElementReference*	Element reference.
encoding	*char* *	Text encoding.
family	*char* *	Font family to use when rendering text.
fill	*PixelPacket*	Object internal fill (within outline) color.
fill_pattern	*Image* *	Image to use as fill pattern.
fill_rule	*FillRule*	Fill rule.
font	*char* *	Font to use when rendering text.
geometry	*char* *	Text scaling and location.
gradient	*GradientInfo*	Gradient information.
gravity	*GravityType*	Text placement preference (e.g. NorthWestGravity).
linecap	*LineCap*	Line cap style.
linejoin	*LineJoin*	Line joining style.
miterlimit	*unsigned long*	Miter limit.
opacity	*Quantum*	Opacity.
pointsize	*double*	Font size (also see density).
primitive	*char* *	Space or new-line delimited list of text drawing primitives (e.g "text 100, 100 Cockatoo"). See the table Drawing Primitives for the available drawing primitives.
render	*unsigned int*	Render flag.
server_name	*char* *	Server name.
signature	*unsigned long*	Internal signature.
stretch	*StretchType*	Font stretch type.
stroke	*PixelPacket*	Object stroke (outline) color.
stroke_antialias	*unsigned int*	Set to True (non-zero) to obtain anti-aliased stroke rendering.
stroke_pattern	*Image* *	Image to use as stroke pattern.
stroke_width	*double*	Stroke (outline) drawing width in pixels.
style	*StyleType*	Font style.
text	*char* *	Text to use for annotation.
text_antialias	*unsigned int*	Set to True (non-zero) to obtain anti-aliased text rendering.
tile	*Image* *	Image texture to draw with. Use an image containing a single color (e.g. a 1x1 image) to draw in a solid color.
undercolor	*PixelPacket*	Under color.
weight	*unsigned long*	Font weight.

ExceptionInfo The members of the ExceptionInfo structure are shown in the following table:

Table20.4: ExceptionInfo Structure

ExceptionInfo Structure

Member	Type	Description
description	*char* *	warning or error description.
error_number	*int*	system errno at time exception was thrown.
reason	*char* *	warning or error message.
severity	*ExceptionType*	warning or error severity.
signature	*unsigned long*	internal signature.

FrameInfo The FrameInfo structure is used to represent dimensioning information for image frames in ImageMagick.

The members of the FrameInfo structure are shown in the following table:

Table20.5: FrameInfo Structure

FrameInfo Structure

Member	Type	Description
width	*unsigned long*	width.
height	*unsigned long*	height.
x	*long*	x.
y	*long*	y.
inner_bevel	*long*	Inner bevel thickness.
outer_bevel	*long*	Outer bevel thickness.

Image The *Image* structure represents an ImageMagick image. It is initially allocated by AllocateImage() and deallocated by DestroyImage(). The functions ReadImage(), ReadImages(), BlobToImage() and CreateImage() return a new image. Use CloneImage() to copy an image. An image consists of a structure containing image attributes as well as the image pixels.

The image pixels are represented by the structure PixelPacket and are cached in-memory, or on disk, depending on the cache threshold setting. This cache is known as the “pixel cache”. Pixels in the cache may not be edited directly. They must first be made visible from the cache via a pixel view. A pixel view is a rectangular view of the pixels as defined by a starting coordinate, and a number of rows and columns. When considering the varying abilities of multiple platforms, the most reliably efficient pixel view is comprized of part, or all, of one image row.

There are three means of accessing pixel views. When using the default view, the pixels are made visible and accessable by using the AcquireImagePixels() method which provides access to a specified region of the image. If you intend to change any of the pixel values, use GetImagePixels(). After the view has been updated, the pixels may be saved back to the cache in their original positions via SyncImagePixels(). In order to create an image with new contents, or to blindly overwrite existing contents, the method SetImagePixels() is used to reserve a pixel view corresponding to a region in the pixel cache. Once the pixel view has been updated, it may be written to the cache via SyncImagePixels(). The function GetIndexes() provides access to the image colormap, represented as an array of type IndexPacket.

A more flexible interface to the image pixels is via the CacheView interface. This interface supports multiple pixel cache views (limited by the number of image rows), each of which are identified by a handle (of type ViewInfo*). Use OpenCacheView() to obtain a new cache view, CloseCacheView() to discard a cache view, GetCacheView() to access an existing pixel region, SetCacheView() to define a new pixel region, and SyncCacheView() to save the updated pixel region. The function GetCacheViewIndexes() provides access to the colormap indexes associated with the pixel view.

When writing encoders and decoders for new image formats, it is convenient to have a high-level interface available which supports converting between external pixel representations and ImageMagick’s own representation. Pixel components (red, green, blue, opacity, RGB, or RGBA) may be transferred from a user-supplied buffer into the default view by using PushImagePixels(). Pixel components may be transferred from the default view into a user-supplied buffer by using PopImagePixels(). Use of this high-level interface helps protect image coders from changes to ImageMagick’s pixel representation and simplifies the implementation.

The members of the Image structure are shown in the following table:

Table20.6: Image Structure

Image Structure

Member	Type	Description
attributes	*ImageAttribute* *	Image attribute list. Consists of a doubly-linked-list of ImageAttribute structures, each of which has an associated key and value. Access/update list via SetImageAttribute() and GetImageAttribute(). Key-strings used by ImageMagick include “Comment” (image comment), “Label” (image label), and “Signature” (image signature). Key-strings used internally by ImageMagick are enclosed in square brackets.
background_color	*PixelPacket*	Image background color.
blob	*BlobInfo* *	A BlobInfo structure whose ”data” member is a blob from which image data is read or to which it is written.
blur	*double*	Blur factor to apply to the image when zooming.
border_color	*PixelPacket*	Image border color.
cache	*void* *	Image cache.
chromaticity	*ChromaticityInfo*	Red, green, blue, and white-point chromaticity values.
client_data	*void* *	Data used by the encoder or decoder.
clip_mask	*Image* *	Image used as a clipping mask.
color_profile	*ProfileInfo*	ICC color profile. Specifications are available from the International Color Consortium for the format of ICC color profiles.
colormap	*PixelPacket*	PseudoColor palette array.
colors	*unsigned long*	The desired number of colors. Used by QuantizeImage().
colorspace	*ColorspaceType*	Image pixel interpretation.If the colorspace is RGB the pixels are red, green, blue. If matte is true, then red, green, blue, and index. If it is CMYK, the pixels are cyan, yellow, magenta, black. Otherwise the colorspace is ignored.

Image Structure (continued)

Member	Type	Description
columns	*unsigned long*	Image width.
comments	*char **	Image comments.
compose	*CompositeOperator*	Composite operator.
compression	*CompressionType*	Image compression type. The default is the compression type of the specified image file.
delay	*unsigned long*	Time in 1/100ths of a second (0 to 65535) which must expire before displaying the next image in an animated sequence. This option is useful for regulating the animation of a sequence of GIF images within Netscape.
depth	*unsigned long*	Image depth (8 or 16).
directory	*char **	Tile names from within an image montage. Only valid after calling MontageImages() or reading a MIFF file which contains a directory.
dispose	*unsigned long*	GIF disposal method. This option is used to control how successive frames are rendered (how the preceding frame is disposed of) when creating a GIF animation.
exception	*ExceptionInfo*	Record of any error which occurred when updating image.
exempt	*unsigned int*	Specifies whether image's file is exempt from being closed by CloseBlob().
endian	*EndianType*	Specifies the endianness of the output image.
filename	*char[MaxTextExtent]*	Image file name to read or write.
filesize	*long int*	Number of bytes of the encoded file.
filter	*FilterTypes*	Filter to use when resizing image. The reduction filter employed has a significant effect on the time required to resize an image and the resulting quality. The default filter is Lanczos which has been shown to produce high quality results when reducing most images.

Image Structure (continued)

Member	Type	Description
fuzz	*double*	Colors within this distance are considered equal. A number of algorithms search for a target color. By default the color must be exact. Use this option to match colors that are close to the target color in RGB space.
gamma	*double*	Gamma level of the image. The same color image displayed on two different workstations may look different due to differences in the display monitor. Use gamma correction to adjust for this color difference.
generic_profiles	*unsigned long*	Number of generic profiles.
generic_profile	*ProfileInfo* *	List of generic profiles.
geometry	*char* *	Preferred size and location of the image when encoding. Positive offsets are measured downward and to the right of the upper left corner. Negative offsets are measured leftward or upward from the right edge or bottom edge.
gravity	*GravityType*	Image gravity.
interlace	*InterlaceType*	The type of interlacing scheme (default NoInterlace). This option is used to specify the type of interlacing scheme for raw image formats such as RGB or YUV. NoInterlace means do not interlace, LineInterlace uses scanline interlacing, and PlaneInterlace uses plane interlacing. PartitionInterlace is like PlaneInterlace except the different planes are saved to individual files (e.g. image.R, image.G, and image.B). Use LineInterlace or PlaneInterlace to create an interlaced GIF or progressive JPEG image.
iptc_profile	*ProfileInfo*	IPTC profile. Specifications are available from the International Press Telecommunications Council for IPTC profiles.
iterations	*unsigned long*	Number of iterations to loop an animation (e.g. Netscape loop extension) for.

Image Structure (continued)

Member	Type	Description
list	*Image* *	Undo image list (used only by 'display')
magick	*char[MaxTextExtent]*	Image encoding format (e.g. "GIF").
magick_columns	*unsigned long*	Base image width (before transformations)
magick_filename	*char[MaxTextExtent]*	Base image filename (before transformations)
magick_rows	*unsigned long*	Base image height (before transformations)
matte	*unsigned int*	If non-zero, then the index member of pixels represents the alpha channel.
matte_color	*PixelPacket*	Image matte (transparent) color
mean_error_ _per_pixel	*double*	The mean error per pixel computed when an image is color reduced. This parameter is only valid if *verbose* is set to *True* and the image has just been quantized.
montage	*char* *	Tile size and offset within an image montage. Only valid for montage images.
next	*Image* *	Next image frame in sequence
normalized_ _maximum_error	*double*	The normalized max error per pixel computed when an image is color reduced. This parameter is only valid if *verbose* is set to true and the image has just been quantized.
normalized_ _mean_error	*double*	The normalized mean error per pixel computed when an image is color reduced. This parameter is only valid if *verbose* is set to *True* and the image has just been quantized.
offset	*long*	Number of initial bytes to skip over when reading raw image.
orphan		[Deprecated].
page	*RectangleInfo*	size of Postscript page and offsets. Offsets are measured from the upper left corner of the page, regardless of their sign.

Image Structure (continued)

Member	Type	Description
pipet	*unsigned int*	Set to *True* if image is read/written from/to a POSIX pipe. To read from (or write to) an open pipe, set this member to True, set the file member to a stdio stream representing the pipe (obtained from popen()), and invoke ReadImage(), WriteImage(). The pipe is automatically closed via pclose() when the operation completes.
pixels	*PixelPacket*	Image pixels retrieved via GetPixelCache() or initialized via SetPixelCache().
previous	*Image **	Previous image frame in sequence.
reference_count	*long*	Reference count.
rendering_intent	*RenderingIntent*	The type of rendering intent.
rows	*unsigned long*	Image height.
scene	*unsigned long*	Image frame scene number.
semaphore	*SemaphoreInfo*	Semaphore.
signature	*unsigned long*	Internal signature used for checking integrity. Note: this is different from the SHA signature reported by "identify".
start_loop	*ClassType*	Marks first image to be displayed in a loop.
status	*unsigned int*	Return code.
storage_class	*ClassType*	Image storage class. If DirectClass then the image packets contain valid RGB or CMYK colors. If PseudoClass then the image has a colormap referenced by pixel's index member.
taint	*int*	Set to non-zero (True) if the image pixels have been modified.
temporary	*unsigned int*	True if image is temporary?.
tile_info	*RectangleInfo*	Describes a tile within an image. For example, if your images is 640x480 you may only want 320x256 with an offset of +128+64. It is used for raw formats such as RGB and CMYK as well as for TIFF.
timer	*TimerInfo*	Support for measuring actual (user + system) and elapsed execution time.

Image Structure (continued)

Member	Type	Description
total_colors	*unsigned long*	The number of colors in the image after QuantizeImage(), or QuantizeImages() if the verbose flag was set before the call. Calculated by GetNumberColors().
units	*ResolutionType*	Units of image resolution
x_resolution	*double*	Horizontal resolution of the image.
y_resolution	*double*	Vertical resolution of the image

ImageAttribute The ImageAttribute structure is used to add arbitary textual attributes to an image. Each attribute has an associated key and value. Add new attributes, or update an existing attribute, via SetImageAttribute() and obtain the value of an existing attribute via GetImageAttribute(). Key-strings used by ImageMagick include "Comment" (image comment), "Label" (image label), and "Signature" (image signature).

The members of the ImageAttribute structure are shown in the following table:

Table20.7: ImageAttribute Structure

ImageAttribute Structure

Member	Type	Description
key	*char* *	key.
value	*char* *	value.
compression	*unsigned int*	compression.
next	*ImageAttribute* *	next attribute in list.
previous	*ImageAttribute* *	previous attribute in list.

ImageInfo The *ImageInfo* structure is used to supply option information to the methods AllocateImage(), AnimateImages(), BlobToImage(), CloneAnnotateInfo(), DisplayImages(), GetAnnotateInfo(), ImageToBlob(), PingImage(), ReadImage(), ReadImages(), and WriteImage(). These methods update information in ImageInfo to reflect attributes of the current image.

Use CloneImageInfo() to duplicate an existing ImageInfo structure or allocate a new one. Use DestroyImageInfo() to deallocate memory associated with an ImageInfo structure. Use GetImageInfo() to initialize an existing ImageInfo struc-

ture. Use SetImageInfo() to set image type information in the ImageInfo structure based on an existing image.

The members of the ImageInfo structure are shown in the following table:

Table20.8: ImageInfo Structure

ImageInfo Structure

Member	**Type**	**Description**
adjoin	*unsigned int*	Join images into a single multi-image file.
affirm	*unsigned int*	Affirm flag.
antialias	*unsigned int*	Control antialiasing of rendered graphic primitives and text fonts.
attributes	*Image* *	Image attributes.
authenticate	*char* *	Password for encrypted input images.
background_color	*PixelPacket*	Image background color.
blob	*void* *	A blob containing an image datastream.
border_color	*PixelPacket*	Image border color.
box	*char* *	Base color that annotation text is rendered on.
cache	*void* *	Cache.
client_data	*void* *	Client data.
colorspace	*ColorspaceType*	Image pixel interpretation. If the colorspace is RGB the pixels are red, green, blue. If matte is true, then red, green, blue, and index. If it is CMYK, the pixels are cyan, yellow, magenta, black. Otherwise the colorspace is ignored.
compression	*CompressionType*	Image compression type. The default is the compression type of the specified image file.
density	*char* *	Vertical and horizontal resolution in pixels of the image. This option specifies an image density when decoding a Postscript or Portable Document page. Often used with page.
depth	*unsigned long*	Image depth (8 or 16).

ImageInfo Structure (continued)

Member	Type	Description
dither	*unsigned int*	Apply Floyd/Steinberg error diffusion to the image. The basic strategy of dithering is to trade intensity resolution for spatial resolution by averaging the intensities of several neighboring pixels. Images which suffer from severe contouring when reducing colors can be improved with this option. The colors or monochrome option must be set for this option to take effect.
endian	*EndianType*	Specify the endianness of the output image.
file	*FILE* *	Stdio stream to read image from or write image to. If set, ImageMagick will read from or write to the stream rather than opening a file. Used by ReadImage() and WriteImage(). The stream is closed when the operation completes.
filename	*char[MaxTextExtent]*	Image file name to read or write.
font	*char* *	Text rendering font. If the font is a fully qualified X server font name, the font is obtained from an X server. To use a TrueType font, precede the TrueType filename with an @. Otherwise, specify a Postscript font name (e.g. “helvetica”).
fuzz	*double*	Colors within this distance are considered equal. A number of algorithms search for a target color. By default the color must be exact. Use this option to match colors that are close to the target color in RGB space.
group	*long*	Group number.

ImageInfo Structure (continued)

Member	Type	Description
interlace	*InterlaceType*	The type of interlacing scheme (default NoInterlace). This option is used to specify the type of interlacing scheme for raw image formats such as RGB or YUV. NoInterlace means do not interlace, LineInterlace uses scanline interlacing, and PlaneInterlace uses plane interlacing. PartitionInterlace is like PlaneInterlace except the different planes are saved to individual files (e.g. image.R, image.G, and image.B). Use LineInterlace or PlaneInterlace to create an interlaced GIF or progressive JPEG image.
length	*size_t*	Length of the ImageInfo blob.
magick	*char[MaxTextExtent]*	Image encoding format (e.g. "GIF").
matte_color	*PixelPacket*	Image matte (transparent) color.
monochrome	*unsigned int*	Transform the image to black and white.
page	*char **	Equivalent size of Postscript page.
pen	*PixelPacket*	Pen color.
ping	*unsigned int*	Set to True to read enough of the image to determine the image columns, rows, and filesize. The columns, rows, and size attributes are valid after invoking ReadImage() while ping is set. The image data is not valid after calling ReadImage() if ping is set.
pointsize	*double*	Text rendering font point size.
preview_type	*PreviewType*	Image manipulation preview option. Used by 'display'.
quality	*unsigned long*	JPEG/MIFF/MNG/PNG compression level (default 75).
sampling_factor	*char **	Sampling factor for the chroma channels in JPEG, MPEG-2, or YUV datastreams.
server_name	*char **	X11 display to display to obtain fonts from, or to capture image from.
signature	*unsigned long*	Signature used internally by ImageMagick to determine integrity of the image_info structure.

ImageInfo Structure (continued)

Member	Type	Description
size	*char **	Width and height of a raw image (an image which does not support width and height information). Size may also be used to affect the image size read from a multi-resolution format (e.g. Photo CD, JBIG, or JPEG.
stream	*StreamHandler*	Stream handler.
subimage	*unsigned long*	Subimage of an image sequence.
subrange	*unsigned long*	Number of images relative to the base image.
temporary	*unsigned int*	Temporary flag.
texture	*char **	Image filename to use as background texture.
tile	*char **	Tile name.
type	*ImageType*	Image type.
unique	*char[MaxTextExtent]*	Unique string.
units	*ResolutionType*	Units of image resolution.
verbose	*unsigned int*	Print detailed information about the image if True.
view	*char **	FlashPix viewing parameters.
zero	*char[MaxTextExtent]*	Zero byte string.

MagickInfo The MagickInfo structure is used by ImageMagick to register support for an Image format. The MagickInfo structure is allocated with default parameters by calling SetMagickInfo(). Image formats are registered by calling RegisterMagickInfo() which adds the initial structure to a linked list (at which point it is owned by the list). A pointer to the structure describing a format may be obtained by calling GetMagickInfo(). Pass the argument NULL to obtain the first member of this list. A human-readable list of registered image formats may be printed to a file descriptor by calling ListMagickInfo().

Support for formats may be provided as a module which is part of the ImageMagick library, provided by a module which is loaded dynamically at runtime, or directly by the linked program. Users of ImageMagick will normally want to create a loadable-module, or support encode/decode of an image format directly from within their program.

Table20.9: MagickInfo Structure

MagickInfo Structure

Member	Type	Description
adjoin	*unsigned int*	Set to non-zero (*True*) if this file format supports multi-frame images.
blob_support	*unsigned int*	Set to non-zero (*True*) if the encoder and decoder for this format supports operating arbitrary BLOBs (rather than only disk files).
client_data	*void **	User specified data. A way to pass any sort of data structure to the endoder/decoder. To set this, GetMagickInfo() must be called to first obtain a pointer to the registered structure since it can not be set via a RegisterMagickInfo() parameter.
decoder	*Image **	*(*decoder)(const ImageInfo *)* Pointer to a function to decode image data and return ImageMagick Image.
description	*const char **	Long form image format description (e.g. "CompuServe graphics interchange format").
encoder	*unsigned int*	*(*encoder)(const ImageInfo, Image *)* Pointer to a function to encode image data with options passed via ImageInfo and image represented by Image.
magick	*const char **	*(const unsigned char *,const size_t)* Pointer to a function that returns *True* if it recognizes this format in the supplied string, otherwise *False*.
module	*const char **	Name of module (e.g. "GIF") which registered this format. Set to NULL if format is not registered by a module.
name	*const char **	Name (e.g. "GIF") of this format.
next	*MagickInfo*	Next MagickInfo struct in linked-list. NULL if none.
previous	*MagickInfo*	Previous MagickInfo struct in linked-list. NULL if none.
raw	*unsigned int*	Image format does not contain size (must be specified in ImageInfo)
signature	*unsigned long*	Signature (*0xabacadab*) used internally by ImageMagick to determine integrity of the image structure.
stealth	*unsigned int*	Image format does not get listed.

MagickInfo Structure (continued)

Member	Type	Description
thread_support	*unsigned int*	Set to non-zero (*True*) if the encoder and decoder are thread safe.
version	*const char* *	Version of the module used to process this image format.

MontageInfo Montage info.

Table20.10: MontageInfo Structure

MontageInfo Structure

Member	Type	Description
background_color	*PixelPacket*	background color.
border_color	*PixelPacket*	border color.
border_width	*unsigned long*	border width.
filename[MaxTextExtent]	*char*	filename.
fill	*PixelPacket*	fill color.
frame	*char* *	geometry of frame.
font	*char* *	font.
geometry	*char* *	geometry of each tile.
gravity	*GravityType*	gravity of tiles.
matte_color	*PixelPacket*	matte color.
pointsize	*double*	point size for text.
shadow	*unsigned int*	shadow (*True* or *False*)
signature	*unsigned long*	internal signature.
stroke	*PixelPacket*	stroke color for text.
texture	*char* *	texture.
tile	*char* *	geometry of tile layout.
title	*char* *	title.

PixelPacket The PixelPacket structure is used to represent DirectClass color pixels in ImageMagick. If the image is indicated as a PseudoClass image, its DirectClass representation is only valid immediately after calling SyncImage(). If an image is set as PseudoClass and the DirectClass representation is modified, the image should then be set as DirectClass. Use QuantizeImage() to restore the PseudoClass colormap if the DirectClass representation is modified.

The members of the PixelPacket structure are shown in the following table:

Table20.11: PixelPacket Structure

PixelPacket Structure

Member	Type	Description
red	*Quantum*	red.
green	*Quantum*	green.
blue	*Quantum*	blue.
opacity	*Quantum*	opacity (0 is fully opaque).

PrimaryInfo The PrimaryInfo structure is used to represent chromaticity points, using (x,y), or for temporary use in converting chromaticity from CIE (X,Y,Z).

The members of the PrimaryInfo structure are shown in the following table:

Table20.12: PrimaryInfo Structure

PrimaryInfo Structure

Member	Type	Description
x	*double*	x.
y	*double*	y.
z	*double*	Z (temporary use only).

ProfileInfo The ProfileInfo structure is used to represent ICC, IPCT, and generic profiles in ImageMagick (stored as an opaque BLOB).

The members of the ProfileInfo structure are shown in the following table:

Table20.13: ProfileInfo Structure

ProfileInfo Structure

Member	Type	Description
length	*unsigned int*	length.
info	*unsigned char* *	data.
name	*char* *	profile name.

RectangleInfo The RectangleInfo structure is used to represent positioning information in ImageMagick.

The members of the RectangleInfo structure are shown in the following table:

Table20.14: RectangleInfo Structure

RectangleInfo Structure

Member	Type	Description
width	*unsigned long*	width.
height	*unsigned long*	height.
x	*long*	x.
y	*long*	y.

SegmentInfo Segment info.

Table20.15: SegmentInfo Structure

SegmentInfo Structure

Member	Type	Description
x1	*double*	x1.
y1	*double*	y1.

SegmentInfo Structure (continued)

Member	Type	Description
x2	*double*	x2.
y2	*double*	y2.

Timer Timer data.

Table20.16: Timer Structure

Timer Structure

Member	Type	Description
start	*double*	start time.
stop	*double*	stop time.
total	*double*	total time.

TimerInfo Timer info.

Table20.17: TimerInfo Structure

TimerInfo Structure

Member	Type	Description
user	*Timer*	user time.
elapsed	*Timer*	elapsed time.
state	*TimerState*	timer state.
signature	*unsigned long*	internal signature.

20.2 API Enumerations

AlignType The type of text alignment.

Table20.18: AlignType Enumeration

AlignType Enumeration

Enumeration	Description
UndefinedAlign	Undefined alignment.
LeftAlign	Left alignment.
RightAlign	Right alignment.
CenterAlign	Center alignment.

CacheType The cache type.

Table20.19: CacheType Enumeration

CacheType Enumeration

Enumeration	Description
UndefinedCache	Undefined cache type.
MemoryCache	Memory cache type.
DiskCache	Disk cache type.
MemoryMappedCache	Memory mapped cache type.

ChannelType ChannelType is used as an argument when doing color separations. Use ChannelType when extracting a layer from an image. MatteChannel is useful for extracting the opacity values from an image. Note that an image may be represented in RGB, RGBA, CMYK, or CMYKA, pixel formats and a channel may only be extracted if it is valid for the current pixel format.

Table20.20: ChannelType Enumeration

ChannelType Enumeration

Enumeration	Description
UndefinedChannel	Unset value.
RedChannel	Extract red channel (RGB images only).
GreenChannel	Extract green channel (RGB images only).
BlueChannel	Extract blue channel (RGB images only).
CyanChannel	Extract cyan channel (CMYK images only).
MagentaChannel	Extract magenta channel (CMYK images only).
YellowChannel	Extract yellow channel (CMYK images only).
BlackChannel	Extract black channel (CMYK images only).
OpacityChannel	Extract opacity channel (CMYKA images only).
MatteChannel	Extract matte (opacity values) channel (RGB images only).

ClassType ClassType specifies the image storage class.

Table20.21: ClassType Enumeration

ClassType Enumeration

Enumeration	Description
UndefinedClass	Unset value.
DirectClass	Image is composed of pixels which represent literal color values.
PseudoClass	Image is composed of pixels which specify an index in a color palette.

ClipPathUnits ClassType specifies the units used in clipping paths.

Table20.22: ClipPathUnits Enumeration

ClipPathUnits Enumeration

Enumeration	Description
UserSpace	User space.
UserSpaceOnUse	User space on use.
ObjectBoundingBox	Object bounding box.

ColorspaceType The ColorspaceType enumeration is used to specify the colorspace that quantization (color reduction and mapping) is done under or to specify the colorspace when encoding an output image. Colorspaces are ways of describing colors to fit the requirements of a particular application (e.g. Television, offset printing, color monitors). Color reduction, by default, takes place in the RGB-Colorspace. Empirical evidence suggests that distances in color spaces such as YUVColorspace or YIQColorspace correspond to perceptual color differences more closely han do distances in RGB space. These color spaces may give better results when color reducing an image. Refer to quantize for more details.

When encoding an output image, the colorspaces RGBColorspace, CMYKColorspace, and GRAYColorspace may be specified. The CMYKColorspace option is only applicable when writing TIFF, JPEG, and Adobe Photoshop bitmap (PSD) files.

Table20.23: ColorspaceType Enumeration

ColorspaceType Enumeration

Enumeration	Description
UndefinedColorspace	Unset value.
RGBColorspace	Red-Green-Blue colorspace.
GRAYColorspace	
TransparentColorspace	The Transparent color space behaves uniquely in that it preserves the matte channel of the image if it exists.
OHTAColorspace	
XYZColorspace	
YCbCrColorspace	
YCCColorspace	

ColorspaceType Enumeration (continued)

Enumeration	Description
YIQColorspace	
YPbPrColorspace	
YUVColorspace	Y-signal, U-signal, and V-signal colorspace. YUV is most widely used to encode color for use in television transmission.
CMYKColorspace	Cyan-Magenta-Yellow-Black colorspace. CYMK is a subtractive color system used by printers and photographers for the rendering of colors with ink or emulsion, normally on a white surface.
sRGBColorspace	

ComplianceType ComplianceType specifies the system used for relating color names to values.

Table20.24: ComplianceType Enumeration

ComplianceType Enumeration

Enumeration	Description
UndefinedCompliance	Undefine compliance.
SVGCompliance	SVG compliance.
X11Compliance	X11 compliance.
XPMCompliance	XPM compliance.
AllCompliance	All compliance.

CompositeOperator CompositeOperator is used to select the image composition algorithm used to compose a composite image with an image. By default, each of the composite mage pixels are replaced by the corresponding image tile pixel. Specify CompositeOperator to select a different algorithm.

Table20.25: CompositeOperator Enumeration

CompositeOperator Enumeration

Enumeration	Description
UndefinedCompositeOp	Unset value.
OverCompositeOp	The result is the union of the the two image shapes with the composite image obscuring image in the region of overlap.
InCompositeOp	The result is a simply composite image cut by the shape of image. None of the image data of image is included in the result.
OutCompositeOp	The resulting image is composite image with the shape of image cut out.
AtopCompositeOp	The result is the same shape as image image, with composite image obscuring image there the image shapes overlap. Note that this differs from OverCompositeOp because the portion of composite image outside of image's shape does not appear in the result.
XorCompositeOp	The result is the image data from both composite image and image that is outside the overlap region. The overlap region will be blank.
PlusCompositeOp	The result is just the sum of the image data. Output values are cropped to MaxRGB (no overflow). This operation is independent of the matte channels.
MinusCompositeOp	The result of composite image - image, with overflow cropped to zero.
AddCompositeOp	The result of composite image + image, with overflow wrapping around (mod (MaxRGB+1)).
SubtractCompositeOp	The result of composite image - image, with underflow wrapping around (mod (MaxRGB+1)). The add and subtract operators can be used to perform reverible transformations.
DifferenceCompositeOp	The result of abs (composite image - image). This is useful for comparing two very similar images.
MultiplyCompositeOp	The result of image multiplied by composite image.

CompositeOperator Enumeration (continued)

Enumeration	Description
BumpmapCompositeOp	The result of image shaded by composite image.
CopyCompositeOp	The resulting image is image replaced with composite image. Here the matte information is ignored.
CopyRedCompositeOp	The resulting image is the red channel in image replaced with the red channel in composite image. The other channels are copied untouched.
CopyGreenCompositeOp	The resulting image is the green channel in image replaced with the green channel in composite image. The other channels are copied untouched.
CopyBlueCompositeOp	The resulting image is the blue channel in image replaced with the blue channel in composite image. The other channels are copied untouched.
CopyOpacityCompositeOp	The resulting image is the opacity channel in image replaced with the opacity channel in composite image. The other channels are copied untouched. The image compositor requires a matte, or opacity channel in the image for some operations. This extra channel usually defines a mask which represents a sort of a cookie-cutter for the image. This is the case when matte is opaque (full coverage) for pixels inside the shape, zero outside, and between 0 and MaxRGB on the boundary. For certain operations, if image does not have a matte channel, it is initialized with 0 for any pixel matching in color to pixel location (0, 0), otherwise MaxRGB (to work properly borderWidth must be 0).
ClearCompositeOp	Clear Op
DissolveCompositeOp	Dissolve Op
DisplaceCompositeOp	Displace Op
ModulateCompositeOp	Modulate Op
ThresholdCompositeOp	Threshold Op
NoCompositeOp	No Op
DarkenCompositeOp	Darken Op
LightenCompositeOp	Lighten Op
HueCompositeOp	Hue Op

CompositeOperator Enumeration (continued)

Enumeration	Description
SaturateCompositeOp	Saturate Op
ColorizeCompositeOp	Colorize Op
LuminizeCompositeOp	Luminize Op
ScreenCompositeOp	Screen Op
OverlayCompositeOp	overlay Op

CompressionType CompressionType is used to express the desired compression type when encoding an image. Be aware that most image types only support a sub-set of the available compression types. If the compression type specified is incompatable with the image, ImageMagick selects a compression type compatable with the image type.

Table20.26: CompressionType Enumeration

CompressionType Enumeration

Enumeration	Description
UndefinedCompression	Unset value.
NoCompression	No compression.
BZipCompression	BZip (Burrows-Wheeler block-sorting text compression algorithm and Huffman coding) as used by bzip2 utilities.
FaxCompression	CCITT Group 3 FAX compression.
Group4Compression	CCITT Group 4 FAX compression (used only for TIFF).
JPEGCompression	JPEG compression.
LosslessJPEGCompression	Lossless JPEG compression.
LZWCompression	Lempel-Ziv-Welch (LZW) compression.
RunlengthEncodedCompression	Run-Length encoded (RLE) compression.
ZipCompression	Lempel-Ziv compression (LZ77) as used in PKZIP and GNU gzip.

DecorationType Types of text decoration.

Table20.27: DecorationType Enumeration

DecorationType Enumeration

Enumeration	Description
NoDecoration	No decoration.
UnderlineDecoration	Underline decoration.
OverlineDecoration	Overline decoration.
LineThroughDecoration	LineThrough decoration.

DisposeType DisposeType specifies the GIF disposal method for an image.

Table20.28: DisposeType Enumeration

DisposeType Enumeration

Enumeration	Description
UndefinedDispose	Disposal method is unspecified.
NoneDispose	Do not dispose of the image.
BackgroundDispose	Overwrite the image area with the background color.
PreviousDispose	Overwrite the image area with what was there previously.

EndianType EndianType specifies the “endianness” of the output file, when the format supports different endian types.

Table20.29: EndianType Enumeration

EndianType Enumeration

Enumeration	Description
UndefinedEndian	Unset value.
LSBEndian	LSB First (Little Endian)
MSBEndian	MSB First (Big endian)

ExceptionType Exception types (Warnings, Errors, and Fatal Errors).

Table20.30: ExceptionType Enumeration

ExceptionType Enumeration

Enumeration	Description
UndefinedException	Undefined exception.
WarningException	Warning exception.
ResourceLimitWarning	Resource limit warning.
TypeWarning	Type warning.
OptionWarning	Option warning.
DelegateWarning	Delegate warning.
MissingDelegateWarning	Missing delegate warning.
CorruptImageWarning	Corrupt image warning.
FileOpenWarning	File open warning.
BlobWarning	Blob warning.
StreamWarning	Stream warning.
CacheWarning	Cache warning.
CoderWarning	Coder warning.
ModuleWarning	Module warning.
DrawWarning	Draw warning.
ImageWarning	Image warning.
XServerWarning	X server warning.
MonitorWarning	Monitor warning.
RegistryWarning	Registry warning.
ConfigureWarning	Configuration warning.
ErrorException	Error exception.
FatalException	Fatal exception.
ResourceLimitError	Resource limit error.
TypeError	Type error.
OptionError	Option error.
DelegateError	Delegate error.
MissingDelegateError	Missing delegate error.
CorruptImageError	Corrupt image error.
FileOpenError	File open error.
BlobError	Blob error.
StreamError	Stream error.
CacheError	Cache error.
CoderError	Coder error.
ModuleError	Module error.
DrawError	Draw error.

ExceptionType Enumeration (continued)

Enumeration	Description
ImageError	Image error.
XServerError	X server error.
MonitorError	Monitor error.
RegistryError	Registry error.
ConfigureError	Configuration error.
FatalErrorException	Fatal error exception.
ResourceLimitFatalError	Resource limit fatal error.
TypeFatalError	Type fatal error.
OptionFatalError Option fatal error.	
DelegateFatalError	Delegate fatal error.
MissingDelegateFatalError	Missing delegate fatal error.
CorruptImageFatalError	Corrupt Image fatal error.
FileOpenFatalError	File open fatal error.
BlobFatalError	Blob fatal error.
StreamFatalError	Stream fatal error.
CacheFatalError	Cache fatal error.
CoderFatalError	Coder fatal error.
ModuleFatalError	Module fatal error.
DrawFatalError	Draw fatal error.
ImageFatalError	Image fatal error.
XServerFatalError	X server fatal error.
MonitorFatalError	Monitor fatal error.
RegistryFatalError	Registry fatal error.
ConfigureFatalError	Configure fatal error.

FillRule Types of fill rules.

Table20.31: FillRule Enumeration

FillRule Enumeration

Enumeration	Description
UndefinedRule	Undefined fill rule.
EvenOddRule	Even-odd fill rule.
NonZeroRule	Nonzero fill rule.

FilterTypes FilterTypes is used to adjust the filter algorithm used when resizing images. Different filters experience varying degrees of success with various images and can take signicantly different amounts of processing time. ImageMagick uses the Lanczos filter by default since this filter has been shown to provide the best results for most images in a reasonable amount of time. Other filter types (e.g. TriangleFilter) may execute much faster but may show artifacts when the image is re-sized or around diagonal lines. The only way to be sure is to test the filter with sample images.

Table20.32: FilterTypes Enumeration

FilterTypes Enumeration

Enumeration	Description
UndefinedFilter	Unset value.
PointFilter	Point Filter
BoxFilter	Box Filter
TriangleFilter	Triangle Filter
HermiteFilter	Hermite Filter
HanningFilter	Hanning Filter
HammingFilter	Hamming Filter
BlackmanFilter	Blackman Filter
GaussianFilter	Gaussian Filter
QuadraticFilter	Quadratic Filter
CubicFilter	Cubic Filter
CatromFilter	Catrom Filter
MitchellFilter	Mitchell Filter
LanczosFilter	Lanczos Filter
BesselFilter	Bessel Filter
SincFilter	Sinc Filter

GeometryFlags Flags that are set depending on what is found while parsing a geometry string.

Table20.33: GeometryFlags Enumeration

GeometryFlags Enumeration

Enumeration	Description
NoValue	No value was found.
XValue	An “x” value was found.
YValue	A “y” value was found.
WidthValue	A “width” value was found.
HeightValue	A “height” value was found.
AllValues	All four values were found.
XNegative	A negative “x” value was found.
YNegative	A negative “y” value was found.
PercentValue	A percent sign was found.
AspectValue	An exclamation point was not found.
LessValue	A “<” symbol was found.
GreaterValue	A “>” symbol was found.
AreaValue	An “@” symbol was found.

GravityType GravityType specifies positioning of an object (e.g. text, image) within a bounding region (e.g. an image). Gravity provides a convenient way to locate objects irrespective of the size of the bounding region, in other words, you don’t need to provide absolute coordinates in order to position an object. A common default for gravity is NorthWestGravity.

Table20.34: GravityType Enumeration

GravityType Enumeration

Enumeration	Description
ForgetGravity	Don’t use gravity.
NorthWestGravity	Position object at top-left of region.
NorthGravity	Postiion object at top-center of region.
NorthEastGravity	Position object at top-right of region.
WestGravity	Position object at left-center of region.
CenterGravity	Position object at center of region.
EastGravity	Position object at right-center of region.
SouthWestGravity	Position object at left-bottom of region.

GravityType Enumeration (continued)

Enumeration	Description
SouthGravity	Position object at bottom-center of region.
SouthEastGravity	Position object at bottom-right of region.

ImageType ImageType indicates the type classification of the image.

Table20.35: ImageType Enumeration

ImageType Enumeration

Enumeration	Description
UndefinedType	Unset value.
BilevelType	Monochrome image.
GrayscaleType	Grayscale image.
PaletteType	Indexed color (palette) image.
PaletteMatteType	Indexed color (palette) image with opacity.
TrueColorType	Truecolor image.
TrueColorMatteType	Truecolor image with opacity.
ColorSeparationType	Cyan/Yellow/Magenta/Black (CYMK) image.

InterlaceType InterlaceType specifies the ordering of the red, green, and blue pixel information in the image. Interlacing is usually used to make image information available to the user faster by taking advantage of the space vs time tradeoff. For example, interlacing allows images on the Web to be recognizable sooner and satellite images to accumulate/render with image resolution increasing over time.

Use LineInterlace or PlaneInterlace to create an interlaced GIF or progressive JPEG image.

Table20.36: InterlaceType Enumeration

InterlaceType Enumeration

Enumeration	Description
UndefinedInterlace	Unset value.
NoInterlace	RGBRGBRGBRGBRGBRGB... (Don't interlace image).
LineInterlace	RRR...GGG...BBB...RRR...GGG...BBB... (Use scanline interlacing).
PlaneInterlace	RRRRRR...GGGGGG...BBBBBB... (Use plane interlacing).
PartitionInterlace	Similar to plane interlacing except that the different planes are saved to individual files (e.g. image.R, image.G, and image.B).

LineCap Types of line caps.

Table20.37: LineCap Enumeration

LineCap Enumeration

Enumeration	Description
UndefinedCap	Undefined cap.
ButtCap	Butt cap.
RoundCap	Round cap.
SquareCap	Square cap.

LineJoin Types of line joining.

Table20.38: LineJoin Enumeration

LineJoin Enumeration

Enumeration	Description
UndefinedJoin	Undefined line join method.
MiterJoin	Miter line join method.
RoundJoin	Round line join method.
BevelJoin	Bevel line join method.

LogEventType Magic methods.

Table20.39: LogEventType Enumeration

LogEventType Enumeration

Enumeration	Description
UndefinedMagicMethod	Undefined magic method.
NoEvents	Do not log any events.
ConfigureEvent	Log configure events.
AnnotateEvent	Log annotate events.
DrawEvent	Log draw events.
LocaleEvent	Log locale events.
CoderEvent	Log coder events.
TransformEvent	transform events.
X11Event	Log X11 events.
CacheEvent	Log cache events.
BlobEvent	Log blob events.
DeprecateEvent	Log deprecated events.
UserEvents	Log user events.
AllEvents	Log all events.

MagicMethod Magic methods.

Table20.40: MagicMethod Enumeration

MagicMethod Enumeration

Enumeration	Description
UndefinedMagicMethod	Undefined magic method.
StringMagicMethod	String magic method.

MapMode Map modes.

Table20.41: MapMode Enumeration

MapMode Enumeration

Enumeration	Description
ReadMode	Read map mode.
WriteMode	Write map mode.
IOMod	I/O map mode.

MontageMode Montage modes.

Table20.42: MontageMode Enumeration

MontageMode Enumeration

Enumeration	Description
UndefinedMode	Undefined montage mode.
FrameMode	Frame montage mode.
UnframeMode	Unframe montage mode.
ConcatenateMode	Concatenate montage mode.

NoiseType NoiseType is used as an argument to select the type of noise to be added to the image.

Table20.43: NoiseType Enumeration

NoiseType Enumeration

Enumeration	Description
UniformNoise	Uniform noise.
GaussianNoise	Gaussian noise.
MultiplicativeGaussianNoise	Multiplicative Gaussian noise.
ImpulseNoise	Impulse noise.
LaplacianNoise	Laplacian noise.
PoissonNoise	Poisson noise.

PaintMethod PaintMethod specifies how pixel colors are to be replaced in the image. It is used to select the pixel-filling algorithm employed.

Table20.44: PaintMethod Enumeration

PaintMethod Enumeration

Enumeration	Description
PointMethod	Replace pixel color at point.
ReplaceMethod	Replace color for all image pixels matching color at point.
FloodfillMethod	Replace color for pixels surrounding point until encountering pixel that fails to match color at point.
FillToBorderMethod	Replace color for pixels surrounding point until encountering pixels matching border color.
ResetMethod	Replace colors for all pixels in image with pen color.

PreviewType Preview types.

Table20.45: PreviewType Enumeration

PreviewType Enumeration

Enumeration	Description
UndefinedPreview	Undefined Preview.
RotatePreview	Preview of Rotate effect
ShearPreview	Preview of Shear effect.
RollPreview	Preview of Roll effect.
HuePreview	Preview of Hue effect.
SaturationPreview	Preview of Saturation effect.
BrightnessPreview	Preview of Brightness effect.
GammaPreview	Preview of Gamma effect.
SpiffPreview	Preview of Spiff effect.
DullPreview	Preview of Dull effect.
GrayscalePreview	Preview of Grayscale effect.
QuantizePreview	Preview of Quantize effect.
DespecklePreview	Preview of Despeckle effect.
ReduceNoisePreview	Preview of ReduceNoise effect.
AddNoisePreview	Preview of AddNoise effect.
SharpenPreview	Preview of Sharpen effect.
BlurPreview	Preview of Blur effect.
ThresholdPreview	Preview of Threshold effect.
EdgeDetectPreview	Preview of EdgeDetect effect.
SpreadPreview	Preview of Spread effect.
SolarizePreview	Preview of Solarize effect.
ShadePreview	Preview of Shade effect.
RaisePreview	Preview of Raise effect.
SegmentPreview	Preview of Segment effect.
SwirlPreview	Preview of Swirl effect.
ImplodePreview	Preview of Implode effect.
WavePreview	Preview of Wave effect.
OilPaintPreview	Preview of OilPaint effect.
CharcoalDrawingPreview	Preview of CharcoalDrawing effect.
JPEGPreview	Preview of JPEG compression.

PrimitiveType Primitives used in drawing operations.

Table20.46: PrimitiveType Enumeration

PrimitiveType Enumeration

Enumeration	Description
UndefinedPrimitive	Undefined Primitive.
PointPrimitive	Point Primitive.
LinePrimitive	Line Primitive.
RectanglePrimitive	Rectangle Primitive.
RoundRectanglePrimitive	Round Rectangle Primitive.
ArcPrimitive	Arc Primitive.
EllipsePrimitive	Ellipse Primitive.
CirclePrimitive	Circle Primitive.
PolylinePrimitive	Polyline Primitive.
PolygonPrimitive	Polygon Primitive.
BezierPrimitive	Bezier Primitive.
ColorPrimitive	Color Primitive.
MattePrimitive	Matte Primitive.
TextPrimitive	Text Primitive.
ImagePrimitive	Image Primitive.
PathPrimitive	Path Primitive.

ProfileType Profiles can be embedded in an image file by digital cameras and by image processing software. ImageMagick recognizes the profiles listed here, and also stores other profiles found in images as “generic” profiles.

Table20.47: ProfileType Enumeration

ProfileType Enumeration

Enumeration	Description
UndefinedProfile	Unset value.
ICMProfile	ICC Color Profile.
IPTCProfile	IPTC Newswire Profile.

RenderingIntent Rendering intent is a concept defined by ICC Spec ICC.1:1998-09, "File Format for Color Profiles". ImageMagick uses RenderingIntent in order to support ICC Color Profiles.

From the specification: "Rendering intent specifies the style of reproduction to be used during the evaluation of this profile in a sequence of profiles. It applies specifically to that profile in the sequence and not to the entire sequence. Typically, the user or application will set the rendering intent dynamically at runtime or embedding time."

Table20.48: RenderingIntent Enumeration

RenderingIntent Enumeration

Enumeration	Description
UndefinedIntent	Unset value.
SaturationIntent	A rendering intent that specifies that the saturation of the pixels in the image is preserved perhaps at the expense of accuracy in hue and lightness.
PerceptualIntent	A rendering intent that specifies that the full gamut of the image is compressed or expanded to fill the gamut of the destination device. Gray balance is preserved but colorimetric accuracy might not be preserved.
AbsoluteIntent	Absolute colorimetric.
RelativeIntent	Relative colorimetric.

ResolutionType By default, ImageMagick defines resolutions in pixels per inch. ResolutionType provides a means to adjust this.

Table20.49: ResolutionType Enumeration

ResolutionType Enumeration

Enumeration	Description
UndefinedResolution	Unset value.
PixelsPerInchResolution	Density specifications are specified in units of pixels per inch (english units).
PixelsPerCentimeterResolution	Density specifications are specified in units of pixels per centimeter (metric units).

StretchType Stretch types used in rendering text.

Table20.50: StretchType Enumeration

StretchType Enumeration

Enumeration	Description
NormalStretch	Normal stretch style.
UltraCondensedStretch	Ultra condensed stretch style.
ExtraCondensedStretch	Extra condensed stretch style.
CondensedStretch	Condensed stretch style.
SemiCondensedStretch	Semicondensed stretch style.
SemiExpandedStretch	Semi expanded stretch style.
ExpandedStretch	Expanded stretch style.
ExtraExpandedStretch	Extra expanded stretch style.
UltraExpandedStretch	Ultra expanded stretch style.
AnyStretch	Any stretch style.

StyleType Style types used in rendering text.

Table20.51: StyleType Enumeration

StyleType Enumeration

Enumeration	Description
NormalStyle	Normal style.
ItalicStyle	Italic style.
ObliqueStyle	Oblique style.
AnyStyle	Any style.

TimerState Timer states.

Table20.52: TimerState Enumeration

TimerState Enumeration

Enumeration	Description
UndefinedTimerState	Undefined timer state.
StoppedTimerState	Stopped timer state.
RunningTimerState	Running timer state.

VirtualPixelMethod Virtual Pixel methods used in operations that require an off-image pixel.

Table20.53: VirtualPixelMethod Enumeration

VirtualPixelMethod Enumeration

Enumeration	Description
NormalStyle	Normal style.
UndefinedVirtualPixelMethod	Undefined method.
ConstantVirtualPixelMethod	Use the background color.
EdgeVirtualPixelMethod	Extend the edge color.
MirrorVirtualPixelMethod	Mirror the image.
TileVirtualPixelMethod	Tile the image.

21 C API Methods

21.1 Methods to Constitute an Image

ConstituteImage() create an image from pixel data.

Image *ConstituteImage (const unsigned long width, const unsigned long height, const char *map, const StorageType type, const void *pixels, ExceptionInfo *exception)

ConstituteImage() returns an image from the pixel data you supply. The pixel data must be in scanline order top-to-bottom. The data can be of type *char*, *short int*, *int*, *long*, *float*, or *double*. *Float* and *double* require the pixels to be normalized [0..1] otherwise [0..MaxRGB]. For example, to create a 640 x 480 image from unsigned red-green-blue character data, use

```
image = ConstituteImage(640, 480, "RGB", CharPixel, pixels,
        exception);
```

A description of each parameter follows:

width Width in pixels of the image.
height Height in pixels of the image.
map This string reflects the expected ordering of the pixel array. It can be any combination or order of R = red, G = green, B = blue, A = alpha, C = cyan, Y = yellow, M = magenta, K = black, or I = intensity (for grayscale).
type Define the data type of the pixels. Float and double types are expected to be normalized [0..1] otherwise [0..MaxRGB]. Choose from these types:

CharPixel	ShortPixel	IntegerPixel
LongPixel	FloatPixel	DoublePixel

pixels This array of values contain the pixel components as defined by `map` and `type`. The expected length of the array varies depending on the values of `width`, `height`, `map`, and `type`.
exception Return any errors or warnings in this structure.

DispatchImage() extract pixel data from an image.

unsigned int DispatchImage(Image *image, const long x, const long y, const unsigned long columns, const unsigned long rows, const char *map, const StorageType type, void *pixels, ExceptionInfo *exception)

DispatchImage() extracts pixel data from an image and returns it to you. The method returns False on success otherwise True if an error is encountered. The data is returned as *char*, *short int*, *int*, *long*, *float*, or *double* in the order specified by `map`.

Suppose we want want to extract the first scanline of a 640x480 image as character data in red-green-blue order:

```
status = DispatchImage(image, 0, 0, 640, 1, "RGB", 0, pixels,
         exception);
```

A description of each parameter follows:

image The image.

x, y, columns, rows These values define the perimeter of a region of pixels you want to extract.

map This string reflects the expected ordering of the pixel array. It can be any combination or order of R = red, G = green, B = blue, A = alpha, C = cyan, Y = yellow, M = magenta, K = black, or I = intensity (for grayscale).

type Define the data type of the pixels. Float and double types are normalized to [0..1] otherwise [0..MaxRGB]. Choose from these types:

CharPixel	ShortPixel	IntegerPixel
LongPixel	FloatPixel	DoublePixel

pixels This array of values contain the pixel components as defined by `map` and `type`. You must preallocate this array where the expected length varies depending on the values of `width`, `height`, `map`, and `type`.

exception Return any errors or warnings in this structure.

PingImage() get information about an image.

Image *PingImage(const ImageInfo *image_info, ExceptionInfo *exception)

PingImage() returns all the attributes of an image or image sequence except for the pixels. It is much faster and consumes far less memory than ReadImage(). On failure, a NULL image is returned and `exception` describes the reason for the failure.

A description of each parameter follows:

image_info Ping the image defined by the `file` or `filename` members of this structure.
exception Return any errors or warnings in this structure.

ReadImage() read one or more image files.

Image *ReadImage(const ImageInfo *image_info, ExceptionInfo *exception)

ReadImage() reads an image or image sequence from a file or file handle. On failure, a NULL image is returned and `exception` describes the reason for the failure.

A description of each parameter follows:

image_info Read the image defined by the `file` or `filename` members of this structure.
exception Return any errors or warnings in this structure.

WriteImage() write one or more image files.

unsigned int WriteImage(const ImageInfo *image_info, Image *image)

Use Write() to write an image or an image sequence to a file or filehandle. If writing to a file on disk, the name is defined by the filename member of the image structure. Write() returns 0 is there is a memory shortage or if the image cannot be written. Check the `exception` member of `image` to determine the cause for any failure.

A description of each parameter follows:

image_info The image info.
image The image.

21.2 ImageMagick Image Methods

AllocateImage() allocate an image.

Image *AllocateImage(const ImageInfo *image_info)

AllocateImage() returns a pointer to an image structure initialized to default values.

A description of each parameter follows:

image_info Many of the image default values are set from this structure. For example, filename, compression, depth, background color, and others.

AllocateImageColormap() allocate an image colormap.

unsigned int AllocateImageColormap(Image *image, const unsigned long colors)

AllocateImageColormap() allocates an image colormap and initializes it to a linear gray colorspace. If the image already has a colormap, it is replaced. AllocateImageColormap() returns True if successful, otherwise False if there is not enough memory.

A description of each parameter follows:

image The image.
colors The number of colors in the image colormap.

AllocateNextImage() allocate the next image in a sequence.

void AllocateNextImage(const ImageInfo *image_info, Image *image)

Use AllocateNextImage() to initialize the next image in a sequence to default values. The `next` member of `image` points to the newly allocated image. If there is a memory shortage, `next` is assigned NULL.

A description of each parameter follows:

image_info Many of the image default values are set from this structure. For example, filename, compression, depth, background color, and others.
image The image.

AnimateImages() animate an image sequence.

unsigned int AnimateImages(const ImageInfo *image_info, Image *image)

AnimateImages() repeatedly displays an image sequence to any X window screen. It returns a value other than 0 if successful. Check the `exception` member of `image` to determine the cause for any failure.

A description of each parameter follows:

image_info The image info.
image The image.

AppendImages() append a set of images.

Image *AppendImages (Image *image, const unsigned int stack, ExceptionInfo *exception)

The Append() method takes a set of images and appends them to each other. Each image in the set must have the same width or height (or both). Append() returns a single image where each image in the original set is side-by-side if all the heights the same or stacked on top of each other if all widths are the same. On failure, a NULL image is returned and `exception` describes the reason for the failure.

A description of each parameter follows:

image The image sequence.
stack An unsigned value other than stacks rectangular image top-to-bottom otherwise left-to-right.
exception Return any errors or warnings in this structure.

AverageImages() average a set of images.

Image *AverageImages (const Image *image, ExceptionInfo *exception)

The Average() method takes a set of images and averages them together. Each image in the set must have the same width and height. Average() returns a single image with each corresponding pixel component of each image averaged. On failure, a NULL image is returned and `exception` describes the reason for the failure.

A description of each parameter follows:

image The image sequence.
exception Return any errors or warnings in this structure.

ChannelImage() extract a channel from the image.

unsigned int ChannelImage (Image *image, const ChannelType channel)

Extract a channel from the image. A channel is a particular color component of each pixel in the image. Choose from these components:

A description of each parameter follows:

image The image.

channel Identify which channel to extract:

Red
Cyan
Green
Magenta
Blue
Yellow
Opacity
Black

CloneImage() create a new copy of an image.

Image *CloneImage(Image *image, const unsigned long columns, const unsigned long rows, const unsigned int orphan, ExceptionInfo *exception)

CloneImage() copies an image and returns the copy as a new image object. If the specified columns and rows is 0, an exact copy of the image is returned, otherwise the pixel data is undefined and must be initialized with the SetImagePixels() and SyncImagePixels() methods. On failure, a NULL image is returned and `exception` describes the reason for the failure.

A description of each parameter follows:

image The image.
columns The number of columns in the cloned image.
rows The number of rows in the cloned image.
orphan With a value other than 0, the cloned image is an orphan. An orphan is a stand-alone image that is not assocated with an image list. In effect, the `next` and `previous` members of the cloned image is set to NULL.
exception Return any errors or warnings in this structure.

CloneImageInfo() clone an image info structure.

ImageInfo *CloneImageInfo(const ImageInfo *image_info)

CloneImageInfo() makes a copy of the given image info structure. If NULL is specified, a new image info structure is created initialized to default values.

A description of each parameter follows:

image_info The image info.

CompositeImage() composite one image to another.

unsigned int CompositeImage(Image *image, const CompositeOperator compose, const Image *composite_image, const long x_offset, const long y_offset)

CompositeImage() returns the second image composited onto the first at the specified offsets.

A description of each parameter follows:

image The image.
compose This operator affects how the composite is applied to the image. The default is Over. Choose from these operators:

OverCompositeOP	DifferenceCompositeOP	XorCompositeOP
AtopCompositeOP	DisplaceCompositeOP	PlusCompositeOP
MinusCompositeOP	SubtractCompositeOP	AddCompositeOP
InCompositeOP	BumpmapCompositeOP	CopyCompositeOP
OutCompositeOP		

composite_image The composite image.
x_offset The column offset of the composited image. If the offset is negative, it is measured between the right edges of the images.
y_offset The row offset of the composited image. If it is negative, it is measured between the bottom edges of the images.

CycleColormapImage() displace a colormap.

CycleColormapImage(Image *image, const int amount)

CycleColormap() displaces an image's colormap by a given number of positions. If you cycle the colormap a number of times you can produce a psychodelic effect.

A description of each parameter follows:

image The image.
amount Offset the colormap this much.

DescribeImage() describe an image.

void DescribeImage (Image *image, FILE *file, const unsigned int verbose)

DescribeImage() describes an image by printing its attributes to the file. Attributes include the image width, height, size, and others.

A description of each parameter follows:

image The image.
file The file, typically stdout.
verbose A value other than zero prints additional detailed information about the image.

DestroyImage() destroy an image.

void DestroyImage(Image *image)

DestroyImage() dereferences an image, deallocating memory associated with the image if the reference count becomes zero.

A description of each parameter follows:

image The image.

DestroyImageInfo() destroy image info.

void DestroyImageInfo(ImageInfo *image_info)

DestroyImageInfo() deallocates memory associated with `image_Info`.

A description of each parameter follows:

image_info The image info.

DisplayImages() display an image sequence.

unsigned int DisplayImages(const ImageInfo *image_info, Image *image)

DisplayImages() displays an image sequence to any X window screen. It returns a value other than 0 if successful. Check the `exception` member of `image` to determine the reason for any failure.

A description of each parameter follows:

image_info The image info.
image The image.

GetImageDepth() get image depth.

unsigned int GetImageDepth(Image *image)

GetImageDepth() returns the depth of the image, either 8 or 16 bits. By default, pixel components are stored as 16-bit two byte unsigned short integers that range in value from 0 to 65535. However, if all the pixels have lower-order bytes that are identical to their higher-order bytes, the image depth is 8-bit.

A description of each parameter follows:

image The image.

GetImageInfo() get image info.

void GetImageInfo(ImageInfo *image_info)

GetImageInfo() initializes `image_info` to default values.

A description of each parameter follows:

image_info The image info.

GetImageType() get image type.

ImageType GetImageType(const Image *image,ExceptionInfo *exception)

GetImageType() returns the type of image:

Bilevel	Grayscale	GrayscaleMatte
Palette	PaletteMatte	TrueColor
TrueColorMatte	ColorSeparation	ColorSeparationMatte
Optimize		

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

IsImagesEqual() measure the pixel differences between two images.

unsigned int IsImagesEqual(Image *image, Image *reference)

IsImagesEqual() measures the difference between colors at each pixel location of two images. A value other than 0 means the colors match exactly. Otherwise an error measure is computed by summing over all pixels in an image the distance squared in RGB space between each image pixel and its corresponding pixel in the reference image. The error measure is assigned to these image members:

mean_error_per_pixel The mean error for any single pixel in the image.
normalized_mean_error The normalized mean quantization error for any single pixel in the image. This distance measure is normalized to a range between 0 and 1. It is independent of the range of red, green, and blue values in the image.
normalized_maximum_error The normalized maximum quantization error for any single pixel in the image. This distance measure is normalized to a range between 0 and 1. It is independent of the range of red, green, and blue values in your image.

Accessed as `image->normalized_mean_error`, a small normalized mean square error, suggests the images are very similiar in spatial layout and color.

A description of each parameter follows:

image The image.
reference The reference image.

IsTaintImage() tell if an image has been altered.

unsigned int IsTaintImage(const Image *image)

IsTaintImage() returns a value other than 0 if any pixel in the image has been altered since it was first constituted.

A description of each parameter follows:

image The image.

ProfileImage() add or remove a profile.

unsigned int ProfileImage(Image *image, const char *profile_name, const char *filename)

ProfileImage() adds or removes a ICM, IPTC, or generic profile from an image. If the profile name is defined it is deleted from the image. If a filename is given, one or more profiles are read and added to the image. ProfileImage() returns a value other than 0 if the profile is successfully added or removed from the image.

A description of each parameter follows:

image The image.
profile_name The type of profile to add or remove.
filename The filename of the ICM, IPTC, or generic profile.

SetImage() set image pixels to the background color.

void SetImage(Image *image, const Quantum opacity)

SetImage() sets the red, green, and blue components of each pixel to the image background color and the opacity component to the specified level of transparency. The background color is defined by the `background_color` member of the image.

A description of each parameter follows:

image The image.
opacity Set each pixel to this level of transparency.

SetImageClipMask()

unsigned int SetImageClipMask(Image *image,Image *clip_mask)

SetImageClipMask() associates a clip mask with the image. The clip mask must be the same dimensions as the image.

A description of each parameter follows:

image The image.
clip_mask The clip mask.

SetImageDepth()

unsigned int SetImageDepth(Image *image,const unsigned long depth)

SetImageDepth() sets the depth of the image, either 8 or 16. Some image formats support both 8 and 16-bits per color component (e.g. PNG). Use SetImageDepth() to specify your preference. A value other than 0 is returned if the depth is set. Check the `exception` member of `image` to determine the cause for any failure.

A description of each parameter follows:

image The image.
depth The image depth.

SetImageOpacity() set image pixels transparency level.

void SetImageOpacity(Image *image, const unsigned long opacity)

SetImageOpacity() attenuates the opacity channel of an image. If the image pixels are opaque, they are set to the specified opacity level. Otherwise, the pixel opacity values are blended with the supplied transparency value.

A description of each parameter follows:

image The image.
opacity The level of transparency: 0 is fully opaque and MaxRGB is fully transparent.

SetImageType() set image type.

void SetImageType(Image *image, const ImageType image_type)

SetImageType() sets the type of image. Choose from these types:

Bilevel	Grayscale	GrayscaleMatte
Palette	PaletteMatte	TrueColor
TrueColorMatte	ColorSeparation	ColorSeparationMatte

A description of each parameter follows:

image The image.
image_type Image type.

TextureImage() tile a texture on image background.

void TextureImage(Image *image, Image *texture)

TextureImage() repeatedly tiles the texture image across and down the image canvas.

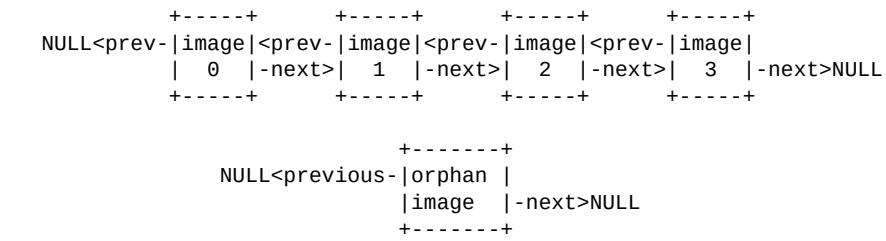
A description of each parameter follows:

image The image.
texture This image is the texture to layer on the background.

21.3 Working With Image Lists

In the ImageMagick API, image lists and sequences are managed by using the "next" and "previous" pointers in the Image structure.

Every image is a member of a doubly-linked image list, as illustrated below:



If the "previous" and "next" pointers are both are NULL, the image is called an "orphan". Each "orphan" is in effect a single-image list. Applications can maintain any number of image lists. Each image belongs to only one image list.

An **image sequence** is that part of an **image list** beginning with a specific image, plus the remainder of the **image list** pointed to by its **next** pointer. The image pointed to by the specific image's "previous" pointer and other images in the list prior to the specific image in the **image list** do not form a part of the **image sequence**.

Each image, image sequence, and image list is referenced by pointing to an image structure of type `Image *`. In the illustration above, a reference to the structure for Image 2 refers to image 2 itself, to the image sequence consisting of images 2 and 3, and to the image list consisting of all images 0 through 3. In the C API, functions that operate on an image list contain the words "ImageList" as a part of the function name, and are described in this section. In general, functions that operate on an image sequence contain the word "Images", although for legacy reasons some, such as ReadImage(), WriteImage(), and PingImage(), do not. In general, functions that contain the word "Image" work on a single image.

CloneImageList() duplicate an image list.

```
Image *CloneImageList(const Image *images, ExceptionInfo *exception)
```

CloneImageList() returns a duplicate of the specified image list.

A description of each parameter follows:

images The image list.
exception Return any errors or warnings in this structure.

DeleteImageFromList() delete an image from the list.

unsigned int DeleteImageFromList(Image *images, const long offset)

DeleteImageFromList() deletes an image at the specified position in the list..

A description of each parameter follows:

images The image list.
offset The position within the list.

DestroyImageList() destroy an image list.

DestroyImageList(Image *images)

DestroyImageList() destroys an image list.

A description of each parameter follows:

images The image list.

GetImageFromList() get an image from an image list.

Image *GetImageFromList(const Image *images, const long offset, ExceptionInfo *exception)

GetImageFromList() returns a clone of the image at the specified position in the image list. The clone is an “orphan”, not linked to the list.

A description of each parameter follows:

images The image list.
offset The position in the image list.
exception Return any errors or warnings in this structure.

GetImageIndexInList() the position in the list of the specified image.

unsigned long *GetImageIndexInList(const Image *images)

GetImageIndexInList() returns the position of the specified image in the image list.

A description of each parameter follows:

images The image list.

GetImageListLength() the number of images in the image list.

unsigned long GetImageListLength(const Image *images)

GetImageListLength() returns the number of images in the image list.

A description of each parameter follows:

images The image list.

GetPreviousImageInList() get the previous image in an image list.

Image *GetPreviousImageInList(Image *images)

GetPreviousImageInList() returns a pointer to the previous image in an image list after the image pointed to by *images.

A description of each parameter follows:

images The image list.

GetNextImageInList() get the next image in an image list.

Image *GetNextImageInList(Image *images)

GetNextImageInList() returns a pointer to the next image in an image list after the image pointed to by *images.

A description of each parameter follows:

images The image list.

ImageListToArray() convert an image list to an array.

Image **ImageListToArray(const Image *images, ExceptionInfo *exception)

ImageListToArray() is a convenience method that converts a linked list of images to a sequential array. For example,

```
Image **group;
group = ImageListToArray(images, exception);
n = GetImageListLength(images);
for (i=0; i < n; i++)
  puts(group[i]->filename);
LiberateMemory((void **) &group);
```

A description of each parameter follows:

image The image list.
exception Return any errors or warnings in this structure.

NewImageList() create an empty image list.

Image *NewImageList(void)

NewImageList() creates an empty image list.

RemoveLastImageFromList() remove the last image from an image list.

Image *RemoveLastImageFromList(Image **images)

RemoveLastImageFromList() removes the last image in the list and returns it.

A description of each parameter follows:

images The image list.

AppendImageToList() adds an image list to the end of an image list.

unsigned int *AppendImageToList(Image **images, const Image *image, ExceptionInfo *exception)

AppendImageToList() adds the image list to the end of the image list.

A description of each parameter follows:

images The image list.
image The image list to be added.
exception Return any errors or warnings in this structure.

ReverseImageList() reverse an image list.

Image *ReverseImageList(Image *images, ExceptionInfo *exception)

ReverseImageList() returns a new list with the order of images reversed from those in the specified image list.

A description of each parameter follows:

images The image list.
exception Return any errors or warnings in this structure.

InsertImageInList() adds an image to the end of an image list.

unsigned int InsertImageInList(Image **images,const Image *image, const long offset,ExceptionInfo *exception)

InsertImageInList() inserts an image into the list at the specified position.

A description of each parameter follows:

images The image list.
image The image.
offset The position within the list.
exception Return any errors or warnings in this structure.

RemoveFirstImageFromList() remove and return the first image in the list.

Image *RemoveFirstImageFromList(Image **images)

RemoveFirstImageFromList() removes an image from the beginning of the specified image list.

A description of each parameter follows:

images The image list.

SpliceImageIntoList() splice an image list.

Image *SpliceImageIntoList(Image *images, const long offset, const unsigned long length, const Image *splices, ExceptionInfo *exception)

SpliceImageIntoList() removes the images designated by offset and length from the list and replaces them with the specified list. The ”splices” list is not necessarily of the same length.

A description of each parameter follows:

images The image list.
offset The position in the image list.
length The length of the image list to remove.
splices Replace the removed image list with this list.
exception Return any errors or warnings in this structure.

PrependImageToList() add an image list to the beginning of the specified list.

unsigned int *PrependImageToList(Image **images, const Image *image, ExceptionInfo *exception)

PrependImageToList() adds an image list to the beginning of the specified image list.

A description of each parameter follows:

images The image list.
image The image list to be added.
exception Return any errors or warnings in this structure.

21.4 Methods to Count the Colors in an Image

CompressColormap() remove duplicate or unused colormap entries.

void CompressColormap(Image *image)

CompressColormap() compresses an image colormap by removing any duplicate or unused color entries.

A description of each parameter follows:

image The image.

GetNumberColors() count the number of unique colors.

unsigned long GetNumberColors(const Image *image, FILE *file, ExceptionInfo *exception)

GetNumberColors() returns the number of unique colors in an image.

A description of each parameter follows:

image The image.
file Write a histogram of the color distribution to this file handle.
exception Return any errors or warnings in this structure.

IsGrayImage() is the image grayscale?

unsigned int IsGrayImage(Image *image, ExceptionInfo *exception)

IsGrayImage() returns True if all the pixels in the image have the same red, green, and blue intensities.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

IsMonochromeImage() is the image monochrome?

unsigned int IsMonochromeImage(Image *image, ExceptionInfo *exception)

IsMonochromeImage() returns True if all the pixels in the image have the same red, green, and blue intensities and the intensity is either 0 or MaxRGB.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

IsOpaqueImage() does the image have transparent pixels?

unsigned int IsOpaqueImage(Image *image, ExceptionInfo *exception)

IsOpaqueImage() returns True if none of the pixels in the image have an opacity value other than opaque (0).

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

IsPaletteImage() does the image have less than 256 unique colors?

unsigned int IsPaletteImage(Image *image, ExceptionInfo *exception)

IsPaletteImage() returns True if the image is colormapped and has 256 unique colors or less.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

ListColorsInfo list color names.

unsigned int ListColorInfo(FILE *file, ExceptionInfo *exception)

ListColorInfo() lists color names to the specified file. Color names are a convenience. Rather than defining a color by its red, green, and blue intensities just use a color name such as `white`, `blue`, or `yellow`.

A description of each parameter follows:

file List color names to this file handle.
exception Return any errors or warnings in this structure.

QueryColorDatabase() return numerical values corresponding to a color name.

unsigned int QueryColorDatabase(const char *name, PixelPacket *color, ExceptionInfo *exception)

QueryColorDatabase() returns the red, green, blue, and opacity intensities for a given color name.

A description of each parameter follows:

name The color name (e.g. white, blue, yellow).
color The red, green, blue, and opacity intensities values of the named color in this structure.
exception Return any errors or warnings in this structure.

QueryColorname() return a color name for the corresponding numerical values.

unsigned int QueryColorname(const Image *image, const PixelPacket *color, ComplianceType compliance, char *name, ExceptionInfo *exception)

QueryColorname() returns a named color for the given color intensity. If an exact match is not found, a hex value is return instead. For example an intensity of rgb:(0,0,0) returns `black` whereas rgb:(223,223,223) returns #dfdfdf.

A description of each parameter follows:

image The image.
color The color intensities.
compliance Adhere to this color standard: SVG or X11.
name Return the color name or hex value.
exception Return any errors or warnings in this structure.

21.5 Methods to Reduce the Number of Unique Colors in an Image

CloneQuantizeInfo()

QuantizeInfo *CloneQuantizeInfo(const QuantizeInfo *quantize_info)

Method CloneQuantizeInfo() makes a duplicate of the given quantize info structure, or if quantize info is NULL, a new one. A description of each parameter follows:

quantize_info a structure of type info.

DestroyQuantizeInfo()

DestroyQuantizeInfo(QuantizeInfo *quantize_info)

Method DestroyQuantizeInfo() deallocates memory associated with an QuantizeInfo structure.

A description of each parameter follows:

quantize_info Specifies a pointer to an QuantizeInfo structure.

GetQuantizeInfo()

GetQuantizeInfo(QuantizeInfo *quantize_info)

Method GetQuantizeInfo() initializes the QuantizeInfo structure.

A description of each parameter follows:

quantize_info Specifies a pointer to a QuantizeInfo structure.

MapImage()

unsigned int MapImage(Image *image, Image *map_image, const unsigned int dither)

MapImage replaces the colors of an image with the closest color from a reference image.

A description of each parameter follows:

image The image.
map_image Specifies a pointer to a Image structure. Reduce image to a set of colors represented by this image.
dither Set this integer value to something other than zero to dither the quantized image.

MapImages()

unsigned int MapImages(Image *images, Image *map_image, const unsigned int dither)

MapImages replaces the colors of a sequence of images with the closest color from a reference image.

A description of each parameter follows:

image The image.

map_image Specifies a pointer to a Image structure. Reduce image to a set of colors represented by this image.

dither Set this integer value to something other than zero to dither the quantized image.

GetImageQuantizeError()

unsigned int GetImageQuantizeError(Image *image)

Method GetImageQuantizeError() measures the difference between the original and quantized images. This difference is the total quantization error. The error is computed by summing over all pixels in an image the distance squared in RGB space between each reference pixel value and its quantized value. These values are computed:

A description of each parameter follows:

mean_error_per_pixel This value is the mean error for any single pixel in the image.

normalized_mean_square_error This value is the normalized mean quantization error for any single pixel in the image. This distance measure is normalized to a range between 0 and 1. It is independent of the range of red, green, and blue values in the image.

normalized_maximum_square_error This value is the normalized maximum quantization error for any single pixel in the image. This distance measure is normalized to a range between 0 and 1. It is independent of the range of red, green, and blue values in your image.

A description of each parameter follows:

image The image.

QuantizeImage()

unsigned int QuantizeImage(const QuantizeInfo *quantize_info, Image *image)

Method QuantizeImage() analyzes the colors within a reference image and chooses a fixed number of colors to represent the image. The goal of the algorithm is to minimize the difference between the input and output image while minimizing the processing time.

A description of each parameter follows:

quantize_info Specifies a pointer to an QuantizeInfo structure.
image Specifies a pointer to a Image structure.

QuantizeImages()

unsigned int QuantizeImages(const QuantizeInfo *quantize_info, Image *images)

QuantizeImages analyzes the colors within a set of reference images and chooses a fixed number of colors to represent the set. The goal of the algorithm is to minimize the difference between the input and output images while minimizing the processing time.

A description of each parameter follows:

quantize_info Specifies a pointer to an QuantizeInfo structure.
images Specifies a pointer to a list of Image structures.

21.6 Methods to Segment an Image with Thresholding Fuzzy c-Means

SegmentImage()

unsigned int SegmentImage(Image *image, const ColorspaceType colorspace, const unsigned int verbose, const double cluster_threshold, const double smoothing_threshold)

Method SegmentImage() segments an image by analyzing the histograms of the color components and identifying units that are homogeneous with the fuzzy c-means technique.

Specify cluster threshold as the number of pixels in each cluster must exceed the the cluster threshold to be considered valid. Smoothing threshold eliminates

noise in the second derivative of the histogram. As the value is increased, you can expect a smoother second derivative. The default is 1.5.

A description of each parameter follows:

image Specifies a pointer to an Image structure returned from ReadImage.
colorspace An unsigned integer value that indicates the colorspace. Empirical evidence suggests that distances in YUV or YIQ correspond to perceptual color differences more closely than do distances in RGB space. The image is then returned to RGB colorspace after color reduction.
verbose A value greater than zero prints detailed information about the identified classes.

21.7 Methods to Resize an Image

MagnifyImage() scale the image to twice its size.

```
Image *MagnifyImage(image, ExceptionInfo *exception)
```

MagnifyImage() is a convenience method that scales an image proportionally to twice its size.

image The image.
exception Return any errors or warnings in this structure.

MinifyImage() scale the image to half its size.

```
Image *MinifyImage(Image *image, ExceptionInfo *exception)
```

MinifyImage() is a convenience method that scales an image proportionally to half its size.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

ResizeImage() scale an image with a filter.

```
Image *ResizeImage(Image *image, const unsigned long columns, const
    unsigned long rows, const FilterType filter, const double blur, Except-
    ionInfo *exception)
```

ResizeImage() scales an image to the desired dimensions with one of these filters:

Bessel	Blackman	Box
Catrom	Cubic	Gaussian
Hanning	Hermite	Lanczos
Mitchell	Point	Quadratic
Sinc	Triangle	

A description of each parameter follows:

image The image.
columns The number of columns in the scaled image.
rows The number of rows in the scaled image.
filter Image filter to use.
blur The blur factor where ¿ 1 is blurry, ¡ 1 is sharp.
exception Return any errors or warnings in this structure.

SampleImage()

Image *SampleImage(Image *image, const unsigned long columns, const unsigned long rows, ExceptionInfo *exception)

SampleImage() scales an image to the desired dimensions with pixel sampling. Unlike other scaling methods, this method does not introduce any additional color into the scaled image.

A description of each parameter follows:

image The image.
columns The number of columns in the sampled image.
rows The number of rows in the sampled image.
exception Return any errors or warnings in this structure.

ScaleImage() scale an image to given dimensions.

Image *ScaleImage(Image *image, const unsigned long columns, const unsigned long rows, ExceptionInfo *exception)

ScaleImage() changes the size of an image to the given dimensions.

A description of each parameter follows:

image The image.
columns The number of columns in the scaled image.
rows The number of rows in the scaled image.
exception Return any errors or warnings in this structure.

21.8 Methods to Transform an Image

ChopImage() chop an image.

Image *ChopImage(Image *image, const RectangleInfo *chop_info, ExceptionInfo *exception)

Chop() removes a region of an image and collapses the image to occupy the removed portion.

A description of each parameter follows:

image The image.
chop_info Define the region of the image to chop with members `x`, `y`, `width`, and `height`. If the image gravity is `Northeast`, `East`, or `SouthEast`, the offset `x` specifies the distance from the right edge of the region to the right edge of the chopping region. Similarly, if the image gravity is `SouthEast`, `South`, or `SouthWest`, `y` is the distance between the bottom edges.
exception Return any errors or warnings in this structure.

CoalesceImages() coalesce a set of images.

Image *CoalesceImages(Image *image, ExceptionInfo *exception)

CoalesceImages() composites a set of images while respecting any page offsets and disposal methods. GIF, MIFF, and MNG animation sequences typically start with an image background and each subsequent image varies in size and offset. Coalesce() returns a new sequence where each image in the sequence is the same size as the first and composited over the previous images in the sequence.

Offsets are measured from the top left corner of the composition to the top left corner of each image. Positive offsets represent a location of the image to the right and downward from the corner of the composition.

A description of each parameter follows:

image The image sequence.
exception Return any errors or warnings in this structure.

CropImage() crop an image.

Image *CropImage(Image *image, const RectangleInfo *crop_info, ExceptionInfo *exception)

Use CropImage() to extract a region of the image starting at the offset defined by `crop_info`.

A description of each parameter follows:

image The image.
crop_info Define the region of the image to crop with members `x`, `y`, `width`, and `height`. If the image gravity is `Northeast`, `East`, or `SouthEast`, the offset `x` specifies the distance from the right edge of the region to the right edge of the cropping region. Similarly, if the image gravity is `SouthEast`, `South`, or `SouthWest`, `y` is the distance between the bottom edges. If the offset `x` is negative, it specifies the distance from the right edge of the region to the right edge of the chopping region.
exception Return any errors or warnings in this structure.

DeconstructImages() return the constituent parts of an image sequence

Image *DeconstructImages(Image *image, ExceptionInfo *exception)

DeconstructImages() compares each image with the next in a sequence and returns the maximum bounding region of any pixel differences it discovers. This method can undo a coalesced sequence returned by CoalesceImages(), and is useful for removing redundant information from a GIF or MNG animation.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

FlipImage() reflect an image vertically.

Image *FlipImage(Image *image, ExceptionInfo *exception)

FlipImage() creates a vertical mirror image by reflecting the pixels around the central x-axis.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

FlopImage() reflect an image horizontally.

Image *FlopImage(Image *image, ExceptionInfo *exception)

FlopImage() creates a horizontal mirror image by reflecting the pixels around the central y-axis.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

MosaicImages() inlay an image sequence to form a single coherent picture.

Image *MosaicImages(const Image *image, ExceptionInfo *exception)

MosaicImages() inlays an image sequence to form a single coherent picture. It returns a single image with each image in the sequence composited at the location defined by the `page` member of `image`.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

RollImage() offset and roll over an image.

Image *RollImage(Image *image, const int x_offset, const int y_offset, ExceptionInfo *exception)

RollImage() offsets an image as defined by `x_offset` and `y_offset`.

A description of each parameter follows:

image The image.
x_offset The number of columns to roll in the horizontal direction, right-to-left (left-to-right if x_offset is negative).
y_offset The number of rows to roll in the vertical direction, bottom-to-top (top-to-bottom if y_offset is negative).
exception Return any errors or warnings in this structure.

ShaveImage()

Image *ShaveImage(const Image *image, const RectangleInfo *shave_info, ExceptionInfo *exception)

Method ShaveImage() shaves pixels from the image edges. It allocates the memory necessary for the new Image structure and returns a pointer to the new image.

A description of each parameter follows:

image The image.
shave_info Specifies a pointer to a structure of type Rectangle which defines the shave region.
exception Return any errors or warnings in this structure.

TransformImage() resize or crop an image.

void TransformImage(Image **image, const char *crop_geometry, const char *image_geometry)

TransformImage() is a convenience method that behaves like ResizeImage() or CropImage() but accepts scaling and/or cropping information as a region geometry specification. If the operation fails, the original image handle is returned.

A description of each parameter follows:

image The image. The transformed image is returned as this parameter.
crop_geometry A crop geometry string. This geometry defines a subregion of the image to crop.
image_geometry An image geometry string. This geometry defines the final size of the image.

21.9 Methods to Shear or Rotate an Image by an Arbitrary Angle

RotateImage

Image *RotateImage(Image *image, const double degrees, ExceptionInfo *exception)

Method RotateImage() creates a new image that is a rotated copy of an existing one. Positive angles rotate counter-clockwise(right-hand rule), while negative angles rotate clockwise. Rotated images are usually larger than the originals and have 'empty' triangular corners. X axis. Empty triangles left over from shearing the image are filled with the color defined by the pixel at location(0, 0). RotateImage allocates the memory necessary for the new Image structure and returns a pointer to the new image.

Method RotateImage() is based on the paper "A Fast Algorithm for General Raster Rotatation" by Alan W. Paeth. RotateImage is adapted from a similar method based on the Paeth paper written by Michael Halle of the Spatial Imaging Group, MIT Media Lab.

A description of each parameter follows:

image The image.
degrees Specifies the number of degrees to rotate the image.
exception Return any errors or warnings in this structure.

ShearImage()

Image *ShearImage(Image *image, const double x_shear, const double y_shear, ExceptionInfo *exception)

Method ShearImage() creates a new image that is a shear_image copy of an existing one. Shearing slides one edge of an image along the X or Y axis, creating a parallelogram. An X direction shear slides an edge along the X axis, while a Y direction shear slides an edge along the Y axis. The amount of the shear is controlled by a shear angle. For X direction shears, x_shear is measured relative to the Y axis, and similarly, for Y direction shears y_shear is measured relative to the X axis. Empty triangles left over from shearing the image are filled with the color defined by the pixel at location(0, 0). ShearImage allocates the memory necessary for the new Image structure and returns a pointer to the new image.

Method ShearImage() is based on the paper "A Fast Algorithm for General Raster Rotatation" by Alan W. Paeth.

A description of each parameter follows:

image The image.
x_shear, y_shear Specifies the number of degrees to shear the image.
exception Return any errors or warnings in this structure.

21.10 Methods to Enhance an Image

ContrastImage() enhance or reduce the image contrast.

unsigned int ContrastImage(Image *image, const unsigned int sharpen)

Contrast() enhances the intensity differences between the lighter and darker elements of the image. Set `sharpen` to a value other than 0 to increase the image contrast otherwise the contrast is reduced.

A description of each parameter follows:

image The image.
sharpen Increase or decrease image contrast.

EqualizeImage() equalize an image.

unsigned int EqualizeImage(Image *image)

EqualizeImage() applies a histogram equalization to the image.

A description of each parameter follows:

image The image.

GammaImage() gamma-correct the image.

unsigned int GammaImage(Image *image, const char *gamma)

Use GammaImage() to gamma-correct an image. The same image viewed on different devices will have perceptual differences in the way the image's intensities are represented on the screen. Specify individual gamma levels for the red, green, and blue channels, or adjust all three with the `gamma` parameter. Values typically range from 0.8 to 2.3.

You can also reduce the influence of a particular channel with a gamma value of 0.

A description of each parameter follows:

image The image.
gamma Define the level of gamma correction.

LevelImage() adjust the level of image contrast.

unsigned int LevelImage(Image *image, const char *levels)

Give three values delineated with commas: black, gamma, and white (e.g. 10,1.0,65000 or 2,0.5,980 to MaxRGB or from 0 to 100from 0.1 to 10. If a "is present, the black and white points are percentages of MaxRGB.

A description of each parameter follows:

image The image.
levels Define the image black and white levels and gamma.

ModulateImage() adjust the brightness, saturation, and hue.

unsigned int ModulateImage(Image *image, const char *modulate)

ModulateImage() lets you control the brightness, saturation, and hue of an image. `Modulate` represents the brightness, saturation, and hue as one parameter (e.g. 90,150,100).

A description of each parameter follows:

image The image.
modulate Define the percent change in brightness, saturation, and hue.

NormalizeImage() enhance image contrast.

unsigned int NormalizeImage(Image *image)

The NormalizeImage() method enhances the contrast of a color image by adjusting the pixels color to span the entire range of colors available.

A description of each parameter follows:

image The image.

21.11 ImageMagick Image Effects Methods

AddNoiseImage() add noise to an image.

Image *AddNoiseImage(const Image *image, const NoiseType noise_type, ExceptionInfo *exception)

AddNoiseImage() adds random noise to the image.

A description of each parameter follows:

image The image.
noise_type The type of noise: Uniform, Gaussian, Multiplicative, Impulse, Laplacian, or Poisson.
exception Return any errors or warnings in this structure.

BlurImage() blur the image.

Image *BlurImage(const Image *image, const double radius, const double sigma, ExceptionInfo *exception)

BlurImage() blurs an image. We convolve the image with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, the radius should be larger than sigma. Use a radius of 0 and BlurImage() selects a suitable radius for you.

A description of each parameter follows:

radius The radius of the Gaussian, in pixels, not counting the center pixel.
sigma The standard deviation of the Gaussian, in pixels.
exception Return any errors or warnings in this structure.

ColorizeImage() colorize an image.

Image *ColorizeImage(const Image *image, const char *opacity, const PixelPacket target, ExceptionInfo *exception)

ColorizeImage() blends the fill color with each pixel in the image. A percentage blend is specified with `opacity`. Control the application of different color components by specifying a different percentage for each component (e.g. 90/100/10 is 90% red, 100% green, and 10% blue).

A description of each parameter follows:

image The image.
opacity A character string indicating the level of opacity as a percentage.
target A color value.
exception Return any errors or warnings in this structure.

ConvolveImage() apply a convolution kernel to the image.

Image *ConvolveImage(const Image *image, const unsigned int order, const double *kernel, ExceptionInfo *exception)

ConvolveImage() applies a custom convolution kernel to the image.

A description of each parameter follows:

image The image.
order The number of columns and rows in the filter kernel.
kernel An array of double representing the convolution kernel.
exception Return any errors or warnings in this structure.

DespeckleImage() filter speckles.

Image *DespeckleImage(const Image *image, ExceptionInfo *exception)

DespeckleImage() reduces the *speckle* noise in an image while perserving the edges of the original image.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

EdgeImage() detect edges within an image.

Image *EdgeImage(const Image *image, const double radius, ExceptionInfo *exception)

EdgeImage() finds edges in an image. `Radius` defines the radius of the convolution filter. Use a radius of 0 and Edge() selects a suitable radius for you.

A description of each parameter follows:

image The image.
radius the radius of the pixel neighborhood.
exception Return any errors or warnings in this structure.

EmbossImage emboss the image.

Image *EmbossImage(const Image *image, const double radius, const double sigma, ExceptionInfo *exception)

EmbossImage() returns a grayscale image with a three-dimensional effect. We convolve the image with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, radius should be larger than sigma. Use a radius of 0 and Emboss() selects a suitable radius for you.

A description of each parameter follows:

image The image.
radius the radius of the pixel neighborhood.
sigma The standard deviation of the Gaussian, in pixels.
exception Return any errors or warnings in this structure.

EnhanceImage() filter a noisy image.

Image *EnhanceImage(const Image *image, ExceptionInfo *exception)

EnhanceImage() applies a digital filter that improves the quality of a noisy image.

A description of each parameter follows:

image The image.
exception Return any errors or warnings in this structure.

GaussianBlurImage() blur an image.

Image *GaussianBlurImage(const Image *image, const double radius, const double sigma, ExceptionInfo *exception)

GaussianBlurImage() blurs an image. We convolve the image with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, the radius should be larger than sigma. Use a radius of 0 and GaussianBlurImage() selects a suitable radius for you.

A description of each parameter follows:

image The image.
radius the radius of the Gaussian, in pixels, not counting the center pixel.
sigma the standard deviation of the Gaussian, in pixels.
exception Return any errors or warnings in this structure.

ImplodeImage() apply an implosion/explosion filter.

Image *ImplodeImage(const Image *image, const double amount, ExceptionInfo *exception)

ImplodeImage() applies a special effects filter to the image where `amount` determines the amount of implosion. Use a negative amount for an explosive effect.

A description of each parameter follows:

image The image.
amount Define the extent of the implosion.
exception Return any errors or warnings in this structure.

MedianFilterImage() filter a noisy image.

Image *MedianFilterImage(const Image *image, const double radius, ExceptionInfo *exception)

MedianFilterImage() applies a digital filter that improves the quality of a noisy image. Each pixel is replaced by the median in a set of neighboring pixels as defined by `radius`.

A description of each parameter follows:

image The image.
radius The radius of the pixel neighborhood.
exception Return any errors or warnings in this structure.

MorphImages() morph a set of images.

Image *MorphImages(const Image *image, const unsigned long number_frames, ExceptionInfo *exception)

The MorphImages() method requires a minimum of two images. The first image is transformed into the second by a number of intervening images as specified by `frames`.

A description of each parameter follows:

image The image.
number_frames Define the number of in-between image to generate. The more in-between frames, the smoother the morph.
exception Return any errors or warnings in this structure.

MotionBlurImage() simulate motion blur.

Image *MotionBlurImage(const Image *image, const double radius, const double sigma, ExceptionInfo *exception)

MotionBlurImage() simulates motion blur. We convolve the image with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, radius should be larger than sigma. Use a radius of 0 and MotionBlurImage()selects a suitable radius for you. `Angle` gives the angle of the blurring motion.

A description of each parameter follows:

image The image.
radius The radius of the Gaussian, in pixels, not counting the center pixel.
sigma The standard deviation of the Motion, in pixels.
angle Apply the effect along this angle.
exception Return any errors or warnings in this structure.

NegateImage()

unsigned int NegateImage(Image *image, const unsigned int grayscale)

Method NegateImage() negates the colors in the reference image. The Grayscale option means that only grayscale values within the image are negated.

A description of each parameter follows:

image The image.

OilPaintImage() simulate an oil painting.

Image *OilPaintImage(const Image *image, const double radius, ExceptionInfo *exception)

OilPaintImage() applies a special effect filter that simulates an oil painting. Each pixel is replaced by the most frequent color occurring in a circular region defined by `radius`.

A description of each parameter follows:

image The image.
radius The radius of the circular neighborhood.
exception Return any errors or warnings in this structure.

PlasmaImage() initialize an image with plasma fractal values.

unsigned int PlasmaImage(const Image *image, const SegmentInfo *segment, int attenuate, int depth)

PlasmaImage() initializes an image with plasma fractal values. The image must be initialized with a base color and the random number generator seeded before this method is called.

A description of each parameter follows:

image The image.
segment Define the region to apply plasma fractals values.
attenuate Define the plasma attenuation factor.
depth Limit the plasma recursion depth.

ReduceNoiseImage() smooth an image.

Image *ReduceNoiseImage(Image *image, const double radius, ExceptionInfo *exception)

ReduceNoiseImage() smooths the contours of an image while still preserving edge information. The algorithm works by replacing each pixel with its neighbor closest in value. A neighbor is defined by `radius`. Use a radius of 0 and ReduceNoise() selects a suitable radius for you.

A description of each parameter follows:

image The image.
radius The radius of the pixel neighborhood.
exception Return any errors or warnings in this structure.

ShadeImage shade the image with light source.

Image *ShadeImage(const Image *image, const unsigned int color_shading, double azimuth, double elevation, ExceptionInfo *exception)

ShadeImage() shines a distant light on an image to create a three-dimensional effect. You control the positioning of the light with *azimuth* and *elevation*; azimuth is measured in degrees off the x axis and elevation is measured in pixels above the Z axis.

A description of each parameter follows:

image The image.
color_shading A value other than zero shades the red, green, and blue components of the image.
azimuth, elevation Define the light source direction.
exception Return any errors or warnings in this structure.

SharpenImage() sharpen an image.

Image *SharpenImage(Image *image, const double radius, const double sigma, ExceptionInfo *exception)

SharpenImage() sharpens an image. We convolve the image with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, radius should be larger than sigma. Use a radius of 0 and SharpenImage() selects a suitable radius for you.

A description of each parameter follows:

radius The radius of the Gaussian, in pixels, not counting the center pixel.
sigma The standard deviation of the Laplacian, in pixels.
exception Return any errors or warnings in this structure.

SolarizeImage() apply solorization special effect.

void SolarizeImage(Image *image, const double threshold)

SolarizeImage() applies a special effect to the image, similar to the effect achieved in a photo darkroom by selectively exposing areas of photo sensitive paper to light. `Threshold` ranges from 0 to MaxRGB and is a measure of the extent of the solarization.

A description of each parameter follows:

image The image.
threshold Define the extent of the solarization.

SpreadImage() randomly displace pixels.

Image *SpreadImage(const Image *image, const unsigned int amount, ExceptionInfo *exception)

SpreadImage() is a special effects method that randomly displaces each pixel in a block defined by the amount parameter.

A description of each parameter follows:

image The image.
radius An unsigned value constraining the ”vicinity” for choosing a random pixel to swap.
exception Return any errors or warnings in this structure.

SteganoImage() hide a digital watermark.

Image *SteganoImage(const Image *image, Image *watermark, ExceptionInfo *exception)

Use SteganoImage() to hide a digital watermark within the image. Recover the hidden watermark later to prove that the authenticity of an image. textttOffset defines the start position within the image to hide the watermark.

A description of each parameter follows:

image The image.
watermark The watermark image.
exception Return any errors or warnings in this structure.

StereoImage() create a stereo special effect.

Image *StereoImage(cosnt Image *image, Image *offset_image, ExceptionInfo *exception)

StereoImage() combines two images and produces a single image that is the composite of a left and right image of a stereo pair. Special red-green stereo glasses are required to view this effect.

A description of each parameter follows:

image The left-hand image.
offset_image The right-hand image.
exception Return any errors or warnings in this structure.

SwirlImage() swirl pixels about image center.

Image *SwirlImage(const Image *image, double degrees, ExceptionInfo *exception)

SwirlImage() swirls the pixels about the center of the image, where `degrees` indicates the sweep of the arc through which each pixel is moved. You get a more dramatic effect as the degrees move from 1 to 360.

A description of each parameter follows:

image The image.
degrees Define the tightness of the swirling effect.
exception Return any errors or warnings in this structure.

ThresholdImage() divide pixels based on intensity values.

unsigned int ThresholdImage(Image *image, const double threshold)

ThresholdImage() changes the value of individual pixels based on the intensity of each pixel compared to `threshold`. The result is a high-contrast, two color image.

A description of each parameter follows:

image The image.
threshold Define the threshold value.

UnsharpMaskImage() sharpen an image.

Image *UnsharpMaskImage(const Image *image, const double radius, const double sigma, const double amount, const double threshold, ExceptionInfo *exception)

UnsharpMaskImage() sharpens an image. We convolve the image with a Gaussian operator of the given radius and standard deviation (sigma). For reasonable results, radius should be larger than sigma. Use a radius of 0 and UnsharpMaskImage() selects a suitable radius for you.

A description of each parameter follows:

image The image.
radius The radius of the Gaussian, in pixels, not counting the center pixel.
sigma The standard deviation of the Gaussian, in pixels.
amount The percentage of the difference between the original and the blur image that is added back into the original.
threshold The threshold, as a fraction of MaxRGB, needed to apply the difference amount.
exception Return any errors or warnings in this structure.

WaveImage() special effects filter.

Image *WaveImage(const Image *image, const double amplitude, const double wave_length, ExceptionInfo *exception)

The WaveImage() filter creates a ”ripple” effect in the image by shifting the pixels vertically along a sine wave whose amplitude and wavelength is specified by the given parameters.

A description of each parameter follows:

image The image.
amplitude, frequency Define the amplitude and wavelength of the sine wave.
exception Return any errors or warnings in this structure.

21.12 ImageMagick Image Decoration Methods

BorderImage() frame the image with a border.

Image *BorderImage(const Image *image, const RectangleInfo *border_info, ExceptionInfo *exception)

BorderImage() surrounds the image with a border of the color defined by the `border_color` member of the `image` structure. The width and height of the border are defined by the corresponding members of the `border_info` structure.

A description of each parameter follows:

image The image.
border_info Define the width and height of the border.
exception Return any errors or warnings in this structure.

FrameImage() surround the image with a decorative border.

Image *FrameImage(const Image *image, const FrameInfo *frame_info, ExceptionInfo *exception)

FrameImage() adds a simulated three-dimensional border around the image. The color of the border is defined by the `matte_color` member of `image`. Members `width` and `height` of `frame_info` specify the border width of the vertical and horizontal sides of the frame. Members `inner` and `outer` indicate the width of the inner and outer `shadows` of the frame.

A description of each parameter follows:

image The image.
frame_info Define the width and height of the frame and its bevels.
exception Return any errors or warnings in this structure.

RaiseImage() lighten or darken edges to create a 3-D effect.

unsigned int RaiseImage(Image *image, const RectangleInfo *raise_info, const int raised)

RaiseImage() creates a simulated three-dimensional button-like effect by lightening and darkening the edges of the image. Members `width` and `height` of `raise_info` define the width of the vertical and horizontal edge of the effect.

A description of each parameter follows:

image The image.
raise_info Define the width and height of the raised area. region.
raised A value other than zero creates a 3-D raised effect, otherwise it has a lowered effect.

21.13 Methods to Annotate an Image

AnnotateImage() annotate an image with text.

unsigned int AnnotateImage(Image *image, DrawInfo *draw_info)

Annotate() allows you to scribble text across an image. The text may be represented as a string or filename. Precede the filename with an "at" sign (@) and the contents of the file are drawn on the image. Your text can optionally embed any of these special characters:

%b file size in bytes.
%c comment.
%d directory in which the image resides.
%e extension of the image file.
%f original filename of the image.
%h height of image.
%i filename of the image.
%k number of unique colors.
%l image label.
%m image file format.
%n number of images in a image sequence.
%o output image filename.

%p page number of the image.
%q image depth (8 or 16).
%s image scene number.
%t image filename without any extension.
%u a unique temporary filename.
%w image width.
%x x resolution of the image.
%y y resolution of the image.

A description of each parameter follows:

image The image.
draw_info The draw info.

GetTypeMetrics() get font attributes.

unsigned int GetTypeMetrics(Image *image, const DrawInfo *draw_info, TypeMetric *metrics)

GetTypeMetrics() returns the following information for the supplied font and text:

- character width
- character height
- ascender
- descender
- text width
- text height
- maximum horizontal advance

A description of each parameter follows:

image The image.
draw_info The draw info.
metrics Return the font metrics in this structure.

21.14 Methods to Draw on an Image

CloneDrawInfo clone a draw info structure.

DrawInfo *CloneDrawInfo(const ImageInfo *image_info, const DrawInfo *draw_info)

CloneDrawInfo() makes a copy of the given draw info structure. If NULL is specified, a new image info structure is created initialized to default values.

A description of each parameter follows:

image_info The image info.
draw_info The draw info.

ColorFloodfillImage() floodfill the designed area with color.

unsigned int ColorFloodfillImage(Image *image, const DrawInfo *draw_info, const PixelPacket target, const long x, const long y, const PaintMethod method)

ColorFloodfill() changes the color value of any pixel that matches `target` and is an immediate neighbor. If the method `FillToBorderMethod` is specified, the color value is changed for any neighbor pixel that does not match the `bordercolor` member of `image`.

By default `target` must match a particular pixel color exactly. However, in many cases two colors may differ by a small amount. The `fuzz` member of `image` defines how much tolerance is acceptable to consider two colors as the same. For example, set fuzz to 10 and the color red at intensities of 100 and 102 respectively are now interpreted as the same color for the purposes of the floodfill.

A description of each parameter follows:

image The image.
draw_info The draw info.
target The RGB value of the target color.
x, y The starting location of the operation.
method Choose either `FloodfillMethod` or `FillToBorderMethod`.

DestroyDrawInfo() destroy draw info.

void DestroyDrawInfo(DrawInfo *draw_info)

DestroyDrawInfo() deallocates memory associated with `draw_info`.

A description of each parameter follows:

draw_info The draw info.

DrawImage annotate an image with a graphic primitive.

unsigned int DrawImage(Image *image, const DrawInfo *draw_info)

Use DrawImage() to draw a graphic primitive on your image. The primitive may be represented as a string or filename. Precede the filename with an "at" sign (@) and the contents of the file are drawn on the image. You can affect how text is drawn by setting one or more members of the draw info structure:

primitive The primitive describes the type of graphic to draw. Choose from these primitives:

PointPrimitive	LinePrimitive	RectanglePrimitive
roundRectanglePrimitive	ArcPrimitive	EllipsePrimitive
CirclePrimitive	PolylinePrimitive	PolygonPrimitive
BezierPrimitive	PathPrimitive	ColorPrimitive
MattePrimitive	TextPrimitive	ImagePrimitive

antialias The visible effect of antialias is to smooth out the rounded corners of the drawn shape. Set to 0 to keep crisp edges.

bordercolor The Color primitive with a method of FloodFill changes the color value of any pixel that matches `fill` and is an immediate neighbor. If `bordercolor` is specified, the color value is changed for any neighbor pixel that is not `fill`.

density This parameter sets the vertical and horizontal resolution of the font. The default is 72 pixels/inch.

fill The fill color paints any areas inside the outline of drawn shape.

font A font can be a Truetype (arial.ttf), Postscript (Helvetica), or a fully-qualified X11 font (-*-helvetica-medium-r-*-*-12-*-*-*-*-*-iso8859-*).

geometry Geometry defines the baseline position where the graphic primitive is rendered (e.g. +100+50).

method Primitives Matte and Image behavior depends on the painting method you choose:

Point	Replace	Floodfull
FillToBorder	Reset	

points List one or more sets of coordinates as required by the graphic primitive you selected.

pointsize The font pointsize. The default is 12.

rotate Specifies a rotation of *rotate-angle* degrees about a given point.

scale Specifies a scale operation by *sx* and *sy*.

skewX Specifies a skew transformation along the x-axis.

skewY Specifies a skew transformation along the y-axis.

stroke A stroke color paints along the outline of the shape.

stroke_width The width of the stroke of the shape. A zero value means no stroke is painted.

translate Specifies a translation by *tx* and *ty*.

A description of each parameter follows:

image The image.
draw_info The draw info.

MatteFloodfillImage() floodfill an area with transparency.

unsigned int MatteFloodfillImage(Image *image, const PixelPacket target, const unsigned int opacity, const long x, const long y, const PaintMethod method)

MatteFloodfill() changes the transparency value of any pixel that matches `target` and is an immediate neighbor. If the method `FillToBorderMethod` is specified, the transparency value is changed for any neighbor pixel that does not match the `bordercolor` member of `image`.

By default `target` must match a particular pixel transparency exactly. However, in many cases two transparency values may differ by a small amount. The `fuzz` member of `image` defines how much tolerance is acceptable to consider two transparency values as the same. For example, set fuzz to 10 and the opacity values of 100 and 102 respectively are now interpreted as the same value for the purposes of the floodfill.

A description of each parameter follows:

image The image.
target The RGB value of the target color.
opacity The level of transparency: 0 is fully opaque and MaxRGB is fully transparent.
x, y The starting location of the operation.
method Choose either `FloodfillMethod` or `FillToBorderMethod`.

OpaqueImage globally change a color.

unsigned int OpaqueImage(Image *image, const PixelPacket target, const PixelPacket fill)

OpaqueImage() changes any pixel that matches `color` with the color defined by `fill`.

By default `color` must match a particular pixel color exactly. However, in many cases two colors may differ by a small amount. `Fuzz` defines how much tolerance is acceptable to consider two colors as the same. For example, set fuzz to 10

and the color red at intensities of 100 and 102 respectively are now interpreted as the same color.

A description of each parameter follows:

image The image.
target The RGB value of the target color.
fill The replacement color.

TransparentImage() make color transparent.

unsigned int TransparentImage(Image *image, const PixelPacket target, const unsigned int opacity)

TransparentImage() changes the opacity value associated with any pixel that matches `color` to the value defined by `opacity`.

By default `color` must match a particular pixel color exactly. However, in many cases two colors may differ by a small amount. `Fuzz` defines how much tolerance is acceptable to consider two colors as the same. For example, set fuzz to 10 and the color red at intensities of 100 and 102 respectively are now interpreted as the same color.

A description of each parameter follows:

image The image.
target The RGB value of the target color.
fill The replacement opacity value.

21.15 Methods to Create a Montage

CloneMontageInfo() clone a montage info structure.

MontageInfo *CloneMontageInfo(const ImageInfo *image_info, const MontageInfo *montage_info)

CloneMontageInfo() makes a copy of the given montage info structure. If NULL is specified, a new image info structure is created initialized to default values.

A description of each parameter follows:

image_info The image info.
montage_info The montage info.

DestroyMontageInfo() destroy montage info.

void DestroyMontageInfo(MontageInfo *montage_info)

DestroyMontageInfo() deallocates memory associated with `montage_info`.

A description of each parameter follows:

montage_info The montage info.

GetMontageInfo() get montage info.

void GetMontageInfo(const ImageInfo *image_info, MontageInfo *montage_info)

GetMontageInfo() initializes `montage_info` to default values.

A description of each parameter follows:

image_info The image info.
montage_info The montage info.

MontageImages() uniformly tile thumbnails across an image canvas.

Image *MontageImages(const Image *image, const MontageInfo *montage_info, ExceptionInfo *exception)

Montageimages() is a layout manager that lets you tile one or more thumbnails across an image canvas.

A description of each parameter follows:

image The image.
montage_info The montage info.
exception Return any errors or warnings in this structure.

21.16 Image Text Attributes Methods

DestroyImageAttributes() destroy an image attribute.

DestroyImageAttributes(Image *image)

DestroyImageAttributes() deallocates memory associated with the image attribute list.

A description of each parameter follows:

image The image.

GetImageAttribute() get an image attribute.

ImageAttribute *GetImageAttribute(const Image *image, const char *key)

GetImageAttribute() searches the list of image attributes and returns a pointer to attribute if it exists otherwise NULL.

A description of each parameter follows:

image The image.
key These character strings are the name of an image attribute to return.

SetImageAttribute() set an image attribute.

unsigned int SetImageAttribute(Image *image, const char *key, const char *value)

SetImageAttribute searches the list of image attributes and replaces the attribute value. If it is not found in the list, the attribute name and value is added to the list. If the attribute exists in the list, the value is concatenated to the attribute. SetImageAttribute returns True if the attribute is successfully concatenated or added to the list, otherwise False. If the value is NULL, the matching key is deleted from the list.

A description of each parameter follows:

image The image.
key, value These character strings are the name and value of an image attribute to replace or add to the list.

21.17 Methods to Compute a Digital Signature for an Image

SignatureImage()

unsigned int SignatureImage(Image *image)

SignatureImage() computes a message digest from an image pixel stream with an implementation of the NIST SHA-256 Message Digest algorithm. This signature uniquely identifies the image and is convenient for determining whether two images are identical.

A description of each parameter follows:

image The image.

21.18 Methods to Interactively Animate an Image Sequence

XAnimateBackgroundImage

void XAnimateBackgroundImage(Display *display, XResourceInfo *resource_info, Image *image)

XAnimateBackgroundImage() animates an image sequence in the background of a window.

A description of each parameter follows:

display Specifies a connection to an X server returned from XOpenDisplay.
resource_info Specifies a pointer to a X11 XResourceInfo structure.
image Specifies a pointer to a Image structure returned from ReadImage.

XAnimateImage animate an image in an X window.

Image *XAnimateImages(Display *display, XResourceInfo *resource_info, char **argv, const int argc, Image *image)

XAnimateImages() displays an image via X11.

A description of each parameter follows:

display Specifies a connection to an X server returned from XOpenDisplay.
resource_info Specifies a pointer to a X11 XResourceInfo structure.
argv Specifies the application's argument list.
argc Specifies the number of arguments.
image Specifies a pointer to a Image structure returned from ReadImage.

21.19 Methods to Interactively Display and Edit an Image

XDisplayBackgroundImage display an image to the background of an X window.

unsigned int XDisplayBackgroundImage(Display *display, XResourceInfo *resource_info, Image *image)

XDisplayBackgroundImage() displays an image in the background of a window.

A description of each parameter follows:

display Specifies a connection to an X server returned from XOpenDisplay.
resource_info Specifies a pointer to a X11 XResourceInfo structure.
image Specifies a pointer to a Image structure returned from ReadImage.

XDisplayImage display an image on an X window.

Image *XDisplayImage(Display *display, XResourceInfo *resource_info, char **argv, int argc, Image **image, unsigned long *state)

XDisplayImage() displays an image via X11. A new image is created and returned if the user interactively transforms the displayed image.

A description of each parameter follows:

display Specifies a connection to an X server returned from XOpenDisplay.
resource_info Specifies a pointer to a X11 XResourceInfo structure.
argv Specifies the application's argument list.
argc Specifies the number of arguments.
image The image.

21.20 Methods to Get or Set Image Pixels

AcquirePixelCache() acquire image pixels.

PixelPacket *AcquirePixelCache(Image *image, const int x, const int y, const unsigned long columns, const unsigned long rows, ExceptionInfo *exception)

AcquirePixelCache() acquires pixels from the in-memory or disk pixel cache as defined by the geometry parameters. A pointer to the pixels is returned if the pixels are transferred, otherwise a NULL is returned.

A description of each parameter follows:

image The image.
x, y, columns, rows These values define the perimeter of a region of
exception Return any errors or warnings in this structure. pixels.

GetIndexes() get indexes.

IndexPacket *GetIndexes(const Image *image)

GetIndexes() returns the colormap indexes associated with the last call to the SetPixelCache() or GetPixelCache() methods.

A description of each parameter follows:

image The image.

GetOnePixel() get one pixel from cache.

PixelPacket *GetOnePixel(const Image image, const int x, const int y)

GetOnePixelFromCache() returns a single pixel at the specified(x, y) location. The image background color is returned if an error occurs.

A description of each parameter follows:

image The image.
x, y These values define the location of the pixel to return.

GetPixelCache() get pixels from cache.

PixelPacket *GetPixelCache(Image *image, const int x, const int y, const unsigned long columns, const unsigned long rows)

GetPixelCache() gets pixels from the in-memory or disk pixel cache as defined by the geometry parameters. A pointer to the pixels is returned if the pixels are transferred, otherwise a NULL is returned.

A description of each parameter follows:

image The image.
x, y, columns, rows These values define the perimeter of a region of pixels.

SetPixelCache() set pixel cache.

PixelPacket *SetPixelCache(Image *image, const int x, const int y, const unsigned long columns, const unsigned long rows)

SetPixelCache() allocates an area to store image pixels as defined by the region rectangle and returns a pointer to the area. This area is subsequently transferred from the pixel cache with method SyncPixelCache. A pointer to the pixels is returned if the pixels are transferred, otherwise a NULL is returned.

A description of each parameter follows:

image The image.
x, y, columns, rows These values define the perimeter of a region of pixels.

SyncPixelCache() synchronize pixel cache.

unsigned int SyncPixelCache(Image *image)

SyncPixelCache() saves the image pixels to the in-memory or disk cache. The method returns True if the pixel region is synced, otherwise False.

A description of each parameter follows:

image The image.

21.21 ImageMagick Cache Views Methods

CloseCacheView close cache view.

void CloseCacheView(ViewInfo *view)

CloseCacheView() closes the specified view returned by a previous call to OpenCacheView().

A description of each parameter follows:

view The address of a structure of type ViewInfo.

GetCacheView get cache view.

PixelPacket *GetCacheView(ViewInfo *view, const int x, const int y, const unsigned long columns, const unsigned long rows)

GetCacheView() gets pixels from the in-memory or disk pixel cache as defined by the geometry parameters. A pointer to the pixels is returned if the pixels are transferred, otherwise a NULL is returned.

A description of each parameter follows:

view The address of a structure of type ViewInfo.
x, y, columns, rows These values define the perimeter of a region of pixels.

GetCacheViewIndexes get cache view indexes.

IndexPacket *GetCacheViewIndexes(const ViewInfo *view)

GetCacheViewIndexes() returns the colormap indexes associated with the specified view.

A description of each parameter follows:

view The address of a structure of type ViewInfo.

GetCacheViewPixels get cache view.

PixelPacket *GetCacheViewPixels(const ViewInfo *view)

GetCacheViewPixels() returns the pixels associated with the specified specified view.

A description of each parameter follows:

view The address of a structure of type ViewInfo.

OpenCacheView open a cache view.

ViewInfo *OpenCacheView(Image *image)

OpenCacheView() opens a view into the pixel cache.

A description of each parameter follows:

image The image.

SetCacheView set a cache view.

PixelPacket *SetCacheView(ViewInfo *view, const long x, const long y, const unsigned long columns, const unsigned long rows)

SetCacheView() gets pixels from the in-memory or disk pixel cache as defined by the geometry parameters. A pointer to the pixels is returned if the pixels are transferred, otherwise a NULL is returned.

A description of each parameter follows:

view The address of a structure of type ViewInfo.
x, y, columns, rows These values define the perimeter of a region of pixels.

SyncCacheView synchronize a cache view.

unsigned int SyncCacheView(ViewInfo *view)

SyncCacheView() saves the view pixels to the in-memory or disk cache. The method returns True if the pixel region is synced, otherwise False.

A description of each parameter follows:

view The address of a structure of type ViewInfo.

21.22 Image Pixel FIFO

ReadStream() read a stream.

unsigned int ReadStream(const ImageInfo *image_info, void (*Stream)(const Image *, const void *, const size_t), ExceptionInfo *exception)

ReadStream() makes the image pixels available to a user supplied callback method immediately upon reading a scanline with the ReadImage() method.

A description of each parameter follows:

image_info The image info.
stream A callback method.
exception Return any errors or warnings in this structure.

WriteStream() write a stream.

unsigned int WriteStream(const ImageInfo *image_info, Image *, int(*Stream) (const Image *, const void *, const size_t))

WriteStream() makes the image pixels available to a user supplied callback method immediately upon writing pixel data with the WriteImage() method.

A description of each parameter follows:

image_info The image info.
stream A callback method.

21.23 Methods to Read or Write Binary Large Objects

BlobToImage() convert a blob to an image.

Image *BlobToImage(const ImageInfo *image_info, const void *blob, const size_t length, ExceptionInfo *exception)

BlobToImage() implements direct to memory image formats. It returns the blob as an image.

A description of each parameter follows:

image_info The image info.
blob The address of a character stream in one of the image formats understood by ImageMagick.
length This size_t integer reflects the length in bytes of the blob.
exception Return any errors or warnings in this structure.

DestroyBlobInfo() destroy a blob.

void DestroyBlobInfo(BlobInfo *blob)

DestroyBlobInfo() deallocates memory associated with an BlobInfo structure.

A description of each parameter follows:

blob Specifies a pointer to a BlobInfo structure.

GetBlobInfo() initialize a blob.

void GetBlobInfo(BlobInfo *blob)

GetBlobInfo() initializes the BlobInfo structure.

A description of each parameter follows:

blob Specifies a pointer to a BlobInfo structure.

ImageToBlob() convert image to a blob.

void *ImageToBlob(const ImageInfo *image_info, Image *image, size_t *length, ExceptionInfo *exception)

ImageToBlob() implements direct to memory image formats. It returns the image as a blob and its length. The magick member of the Image structure determines the format of the returned blob(GIG, JPEG, PNG, etc.).

A description of each parameter follows:

image_info Specifies a pointer to an ImageInfo structure.
image The image.
length This pointer to a size_t integer sets the initial length of the blob. On return, it reflects the actual length of the blob.
exception Return any errors or warnings in this structure.

21.24 ImageMagick Registry Methods

DeleteMagickRegistry delete a blob from the registry.

unsigned int DeleteMagickRegistry(const long id)

DeleteMagickRegistry() deletes an entry in the registry as defined by the id. It returns True if the entry is deleted otherwise False if no entry is found in the registry that matches the id.

A description of each parameter follows:

id The registry id.

GetImageFromMagickRegistry get an image from the registry by name.

```
Image *GetImageFromMagickRegistry(const char *name, ExceptionInfo
  *exception)
```

GetImageFromMagickRegistry() gets an image from the registry as defined by its name. If the blob that matches the name is not found, NULL is returned.

A description of each parameter follows:

name The image name.
exception Return any errors or warnings in this structure.

GetMagickRegistry get a blob from the registry.

```
const void *GetMagickRegistry(const long id,RegistryType *type, size_t
  *length, ExceptionInfo *exception)
```

GetMagickRegistry() gets a blob from the registry as defined by the id. If the blob that matches the id is not found, NULL is returned.

A description of each parameter follows:

id The registry id.
type The registry type.
length The blob length in number of bytes.
exception Return any errors or warnings in this structure.

SetMagickRegistry save a blob to the registry.

```
long SetMagickRegistry(const void *blob,const size_t length, Exception-
  Info *exception)
```

SetMagickRegistry() sets a blob into the registry and returns a unique ID. If an error occurs, -1 is returned.

A description of each parameter follows:

type The registry type.
blob The address of a Binary Large OBject.
length The blob length in number of bytes.
exception Return any errors or warnings in this structure.

21.25 Methods to Read or List ImageMagick Image formats

DestroyMagickInfo() destroy magick info.

void DestroyMagickInfo()

DestroyMagickInfo() deallocates memory associated MagickInfo list.

GetImageMagick() return an image format that matches the magic number.

char *GetImageMagick(const unsigned char *magick, const size_t length)

Method GetImageMagick() searches for an image format that matches the specified magick string. If one is found the tag is returned otherwise NULL.

A description of each parameter follows:

magick The image format we are searching for.
length The length of the binary string.

GetMagickConfigurePath() get the path of a configuration file.

char *GetMagickConfigurePath(const char *filename)

GetMagickConfigurePath() searches a number of pre-defined locations for the specified ImageMagick configuration file and returns the path. The search order follows:

```
<current directory>/
<client path>/
$MAGICK_HOME/
$HOME/.magick/
MagickLibPath
MagickModulesPath
MagickSharePath
```

A description of each parameter follows:

filename The desired configuration file.

GetMagickInfo() get image format attributes.

MagickInfo *GetMagickInfo(const char *tag)

GetMagickInfo() returns a pointer MagickInfo structure that matches the specified tag. If tag is NULL, the head of the image format list is returned.

A description of each parameter follows:

tag The image format we are looking for.
exception Return any errors or warnings in this structure.

GetMagickVersion() get the ImageMagick version.

char *GetMagickVersion(unsigned int *version)

GetMagickVersion() returns the ImageMagick API version as a string and as a number.

A description of each parameter follows:

version The ImageMagick version is returned as a number.

InitializeMagick() initialize the ImageMagick API.

InitializeMagick(const char *path)

InitializeMagick() initializes the ImageMagick environment.

A description of each parameter follows:

path The execution path of the current ImageMagick client.

ListMagickInfo() list the recognized image formats.

void ListMagickInfo(FILE *file)

ListMagickInfo() lists the image formats to a file.

A description of each parameter follows:

file A file handle.
exception Return any errors or warnings in this structure.

RegisterMagickInfo() register a new image format.

MagickInfo *RegisterMagickInfo(MagickInfo *entry)

RegisterMagickInfo() adds attributes for a particular image format to the list of supported formats. The attributes include the image format tag, a method to read and/or write the format, whether the format supports the saving of more than one frame to the same file or blob, whether the format supports native in-memory I/O, and a brief description of the format.

A description of each parameter follows:

entry The magick info.

SetMagickInfo()

MagickInfo *SetMagickInfo(const char *tag)

Method SetMagickInfo() allocates a MagickInfo structure and initializes the members to default values.

A description of each parameter follows:

tag a character string that represents the image format associated with the MagickInfo structure.

UnregisterMagickInfo()

unsigned int UnregisterMagickInfo(const char *tag)

Method UnregisterMagickInfo() removes a tag from the magick info list. It returns False if the tag does not exist in the list otherwise True.

A description of each parameter follows:

tag a character string that represents the image format we are looking for.

21.26 ImageMagick Error Methods

CatchImageException()

CatchImageException(Image *image)

CatchImageException() returns if no exceptions are found in the image sequence, otherwise it determines the most severe exception and reports it as a warning or error depending on the severity.

A description of each parameter follows:

image An image sequence.

DestroyExceptionInfo() destroy exception info.

void DestroyExceptionInfo(ExceptionInfo *exception)

DestroyExceptionInfo() deallocates memory associated with `exception`.

A description of each parameter follows:

exception The exception info.

GetExceptionInfo get exception info.

GetExceptionInfo(ExceptionInfo *exception)

GetExceptionInfo() initializes `exception` to default values.

A description of each parameter follows:

exception The exception info.

GetImageException() get the severest error.

GetImageException(Image *image, ExceptionInfo *exception)

GetImageException() traverses an image sequence and returns any error more severe than noted by the exception parameter.

A description of each parameter follows:

image An image sequence.
exception Return the highest severity exception in the seqeunce.

MagickError() declare an error.

void MagickError(const ExceptionType error, const char *reason, const char *description)

MagickError() calls the error handler method with an error reason.

A description of each parameter follows:

exception The error severity.
reason Define the reason for the error.
description Describe the error.

MagickWarning() declare a warning.

void MagickWarning(const ExceptionType warning, const char *reason, const char *description)

MagickWarning() calls the warning handler method with a warning reason.

A description of each parameter follows:

warning The warning severity.
reason Define the reason for the warning.
description Describe the warning.

SetErrorHandler() set the warning handler.

ErrorHandler SetErrorHandler(ErrorHandler handler)

SetErrorHandler() sets the error handler to the specified method and returns the previous error handler.

A description of each parameter follows:

handler The method to handle errors.

SetWarningHandler() set the warning handler.

ErrorHandler SetWarningHandler(ErrorHandler handler)

SetWarningHandler() sets the warning handler to the specified method and returns the previous warning handler.

A description of each parameter follows:

handler The method to handle warnings.

ThrowException() throw an exception.

void ThrowException(ExceptionInfo *exception, const ExceptionType severity, const char *reason, const char *description)

ThrowException() throws an exception with the specified severity code, reason, and optional description.

A description of each parameter follows:

exception The exception.
severity Define the severity of the exception.
reason Define the reason for the exception.
description Describe the exception.

21.27 ImageMagick Memory Allocation Methods

AcquireMemory allocate memory.

void *AcquireMemory(const size_t size)

AcquireMemory() returns a pointer to a block of memory at least size bytes suitably aligned for any use.

A description of each parameter follows:

size The size of the memory in bytes to allocate.

LiberateMemory free allocated memory.

void LiberateMemory(void **memory)

LiberateMemory() frees memory that has already been allocated.

A description of each parameter follows:

span A pointer to a block memory to free for reuse.

ReacquireMemory change the size of allocated memory.

void ReacquireMemory(void **memory, const size_t size)

ReacquireMemory() changes the size of allocated memory and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

A description of each parameter follows:

memory A pointer to a memory allocation. On return the pointer may change but the contents of the original allocation will not.
size The new size of the allocated memory.

21.28 ImageMagick Progress Monitor Methods

MagickMonitor measure progress toward completion of a task.

MagickExport unsigned int MagickMonitor(const char *text,const off_t quantum, const size_t span,ExceptionInfo *exception)

MagickMonitor() calls the monitor handler method with a text string that describes the task and a measure of completion. The method returns False on success otherwise True if an error is encountered, e.g. if there was a user interrupt.

A description of each parameter follows:

quantum The position relative to the span parameter which represents how much progress has been made toward completing a task.
span The span relative to completing a task.
exception Return any errors or warnings in this structure.

SetMonitorHandler define a custom progress monitor.

MonitorHandler SetMonitorHandler(MonitorHandler handler)

SetMonitorHandler() sets the monitor handler to the specified method and returns the previous monitor handler.

A description of each parameter follows:

handler The progress monitor handler method.

22 C++ API Methods

This section was converted from HTML files in the “www/Magick++” directory of the ImageMagick distribution. Some of the files contain figures which are not yet visible here. Refer to the HTML to see them.

Magick++ provides a simple C++ API to the ImageMagick image processing library which supports reading and writing a huge number of image formats as well as supporting a broad spectrum of traditional image processing operations. The ImageMagick C API is complex and the data structures are currently not documented. Magick++ provides access to most of the features available from the C API but in a simple object-oriented and well-documented framework.

Magick++ is intended to support commercial-grade application development. In order to avoid possible conflicts with the user's application, all symbols contained in Magick++ (included by the header <Magick++.h>) are scoped to the namespace Magick. Symbols from the ImageMagick C library are imported under the MagickLib namespace to avoid possible conflicts and ImageMagick macros are only included within the Magick++ implementation so they won't impact the user's application.

The core class in Magick++ is the Image class. The Image class provides methods to manipulate a single image frame (e.g. a JPEG image). Standard Template Library (STL) compatable algorithms and function objects are provided in order to manipulate multiple image frames or to read and write file formats which support multiple image frames (e.g. GIF animations, MPEG animations, and Postscript files).

The Image class supports reference-counted memory management which supports the semantics of an intrinsic variable type (e.g. 'int') with an extremely efficient operator = and copy constructor (only a pointer is assigned) while ensuring that the image data is replicated as required so that it the image may be modified without impacting earlier generations. Since the Image class manages heap memory internally, images are best allocated via C++ automatic (stack-based) memory allocation. This support allows most programs using Magick++ to be written without using any pointers, simplifying the implementation and avoiding the risks of using pointers.

The image class uses a number of supportive classes in order to specify arguments. Colors are specified via the Color class. Colors specified in X11-style string form are implicitly converted to the Color class. Geometry arguments (those specifying width, height, and/or x and y offset) are specified via the Geometry class. Similar to the Color class, geometries specified as an X11-style string are implicitly converted to the Geometry class. Two dimensional drawable objects are specified via the Drawable class. Drawable objects may be provided as a single object or as a list of objects to be rendered using the current image options. Montage options (a montage is a rendered grid of thumbnails in one image) are specified via the Montage class.

Errors are reported using C++ exceptions derived from the Exception class, which is itself derived from the standard C++ exception class. Exceptions are reported synchronous with the operation and are caught by the first matching try block as the stack is unraveled. This allows a clean coding style in which multiple related Magick++ commands may be executed with errors handled as a unit rather than line-by-line. Since the Image object provides reference-counted memory management, unreferenced images on the stack are automatically cleaned up, avoiding the potential for memory leaks.

For ease of access, the documentation for the available user-level classes is available via the following table.

Magick++ User-Level Classes

Blob	Binary Large OBject container.
CoderInfo	Report information about supported image formats (use with coderInfoList())
Color	Color specification.
Drawable	Drawable shape (for input to 'draw').

Exception	C++ exception objects.
Geometry	Geometry specification.
Image	Image frame. This is the primary object in Magick++.
Montage	Montage options for montageImages().
Pixels	Low-level access to image pixels.
STL	STL algorithms and function objects for operating on containers of image frames.
TypeMetric	Container for font type metrics (use with Image::fontTypeMetrics).

22.1 Magick::Blob

Blob provides the means to contain any opaque data. It is named after the term "Binary Large OBject" commonly used to describe unstructured data (such as encoded images) which is stored in a database. While the function of Blob is very simple (store a pointer and and size associated with allocated data), the Blob class provides some very useful capabilities. In particular, it is fully reference counted just like the Image class.

The Blob class supports value assignment while preserving any outstanding earlier versions of the object. Since assignment is via a pointer internally, Blob is efficient enough to be stored directly in an STL container or any other data structure which requires assignment. In particular, by storing a Blob in an associative container (such as STL's 'map') it is possible to create simple indexed in-memory "database" of Blobs.

Magick++ currently uses Blob to contain encoded images (e.g. JPEG) as well as ICC and IPTC profiles. Since Blob is a general-purpose class, it may be used for other purposes as well.

The methods Blob provides are shown in the following table:

Blob Methods

Method	Return Type	Signature(s)	Description
Blob		void	Default constructor
	const void* data_, size_t length_	Construct object with data, making a copy of the supplied data	
	const Blob& blob_	Copy constructor (reference counted)	
operator=	Blob	const Blob& blob_	Assignment operator (reference counted)
update	void	const void* data_, size_t length_	Update object contents, making a copy of the supplied data. Any existing data in the object is deallocated.
data	const void*	void	Obtain pointer to data
length	size_t	void	Obtain data length
updateNoCopy	void	void* data_, size_t length_, Blob::Allocator allocator_ = Blob::NewAllocator	Update object contents, using supplied pointer directly (no copy) Any existing data in the object is deallocated. The user must ensure that the pointer supplied is not deleted or otherwise modified after it has been supplied to this method. The optional allocator_ parameter

```
allows the user to
specify if the C
(MallocAllocator) or C++
(NewAllocator) memory
allocation system was
used to allocate the
memory. The default is to
use the C++ memory
allocator.
```

22.2 Magick::CoderInfo

The CoderInfo class provides the means to provide information regarding ImageMagick support for an image format (designated by a magick string). It may be used to provide support for a specific named format (provided as an argument to the constructor), or as an element of a container when format support is queried using the coderInfoList() templated function.

The following code fragment illustrates how CoderInfo may be used.

```
    CoderInfo info("GIF");
    cout << info->name() << ": (" << info->description() << ") : ";
    cout << "Readable = ";
    if ( info->isReadable() )
      cout << "true";
    else
      cout << "false";
    cout << ", ";
    cout << "Writable = ";
    if ( info->isWritable() )
      cout << "true";
    else
      cout << "false";
    cout << ", ";
    cout << "Multiframe = ";
    if ( info->isMultiframe() )
      cout << "true";
    else
      cout << "false";
    cout << endl;
```

The methods available in the CoderInfo class are shown in the following table:

CoderInfo Methods

Method	Returns	Signature	Description
CoderInfo		void	Construct object corresponding to named format (e.g. "GIF"). An exception is thrown if the format is not supported.
name	std::string	void	Format name (e.g. "GIF").
description	std::string	void	Format description (e.g. "CompuServe graphics interchange format").
isReadable	bool	void	Format is readable.
isWritable	bool	void	Format is writeable.
isMultiFrame	bool	void	Format supports multiple frames.

22.3 Magick::Color

Color is the base color class in Magick++. It is a simple container class for the pixel red, green, blue, and alpha values scaled to fit ImageMagick's Quantum size. Normally users will instantiate a class derived from Color which supports the color model that fits the needs of the application. The Color class may be constructed directly from an X11-style color string.

Available derived color specification classes are shown in the following table:

Color Derived Classes

ColorRGB	Representation of RGB color with red, green, and blue specified as ratios (0 to 1)
ColorGray	Representation of grayscale RGB color (equal parts red, green, and blue) specified as a ratio (0 to 1)
ColorMono	Representation of a black/white color (true/false)
ColorYUV	Representation of a color in the YUV colorspace

ImageMagick may be compiled to support 32 or 64 bit pixels of type PixelPacket. This is controlled by the value of the QuantumDepth define. The default is 64 bit pixels, which provide the best accuracy. If memory consumption must be minimized, or processing time must be minimized, then ImageMagick may be compiled with QuantumDepth=8. The following table shows the relationship between QuantumDepth, the type of Quantum, and the overall PixelPacket size.

Effect Of QuantumDepth Values

QuantumDepth	Quantum Typedef	PixelPacket Size
8	unsigned char	32 bits
16	unsigned short	64 bits

Color Class

The Color base class is not intended to be used directly. Normally a user will construct a derived class or inherit from this class. Color arguments are must be scaled to fit the Quantum size. The Color class contains a pointer to a PixelPacket, which may be allocated by the Color class, or may refer to an existing pixel in an image.

An alternate way to contruct the class is via an X11-compatable color specification string.

```
class Color
{
  public:
    Color ( Quantum red_,
     Quantum green_,
     Quantum blue_ );
    Color ( Quantum red_,
     Quantum green_,
     Quantum blue_,
     Quantum alpha_ );
    Color ( const std::string &x11color_ );
    Color ( const char * x11color_ );
    Color ( void );
    virtual        ~Color ( void );
```

```
    Color ( const Color & color_ );

    // Red color (range 0 to MaxRGB)
    void           redQuantum ( Quantum red_ );
    Quantum        redQuantum ( void ) const;

    // Green color (range 0 to MaxRGB)
    void           greenQuantum ( Quantum green_ );
    Quantum        greenQuantum ( void ) const;

    // Blue color (range 0 to MaxRGB)
    void           blueQuantum ( Quantum blue_ );
    Quantum        blueQuantum ( void ) const;

    // Alpha level (range OpaqueOpacity=0 to TransparentOpacity=MaxRGB)
    void           alphaQuantum ( Quantum alpha_ );
    Quantum        alphaQuantum ( void ) const;

    // Scaled (to 1.0) version of alpha for use in sub-classes
    // (range opaque=0 to transparent=1.0)
    void           alpha ( double alpha_ );
    double         alpha ( void ) const;

    // Does object contain valid color?
    void           isValid ( bool valid_ );
    bool           isValid ( void ) const;

    // Set color via X11 color specification string
    const Color& operator= ( const std::string &x11color_ );
    const Color& operator= ( const char * x11color_ );

    // Assignment operator
    Color& operator= ( const Color& color_ );

    // Return X11 color specification string
    /* virtual */ operator std::string() const;

    // Return ImageMagick PixelPacket
    operator PixelPacket() const;

    // Construct color via ImageMagick PixelPacket
    Color ( const PixelPacket &color_ );

    // Set color via ImageMagick PixelPacket
    const Color& operator= ( PixelPacket &color_ );
};


ColorRGB

Representation of an RGB color. All color arguments have a valid range of
0.0 - 1.0.

class ColorRGB : public Color
{
  public:
    ColorRGB ( double red_, double green_, double blue_ );
    ColorRGB ( void );
    ColorRGB ( const Color & color_ );
    /* virtual */  ~ColorRGB ( void );

    void           red ( double red_ );
    double         red ( void ) const;

    void           green ( double green_ );
    double         green ( void ) const;

    void           blue ( double blue_ );
    double         blue ( void ) const;
```

```
    // Assignment operator from base class
    ColorRGB& operator= ( const Color& color_ );
};
```

ColorGray

Representation of a grayscale color (in RGB colorspace). Grayscale is simply RGB with equal parts of red, green, and blue. All double arguments have a valid range of 0.0 - 1.0.

```
class ColorGray : public Color
{
  public:
    ColorGray ( double shade_ );
    ColorGray ( void );
    ColorGray ( const Color & color_ );
    /* virtual */ ~ColorGray ();

    void           shade ( double shade_ );
    double         shade ( void ) const;

    // Assignment operator from base class
    ColorGray& operator= ( const Color& color_ );
};
```

ColorMono

Representation of a black/white pixel (in RGB colorspace). Color arguments are constrained to 'false' (black pixel) and 'true' (white pixel).

```
class ColorMono : public Color
{
  public:
    ColorMono ( bool mono_ );
    ColorMono ( void );
    ColorMono ( const Color & color_ );
    /* virtual */ ~ColorMono ();

    void           mono ( bool mono_ );
    bool           mono ( void ) const;

    // Assignment operator from base class
    ColorMono& operator= ( const Color& color_ );
};
```

ColorHSL

Representation of a color in Hue/Saturation/Luminosity (HSL) colorspace.

```
class ColorHSL : public Color
{
  public:
    ColorHSL ( double hue_, double saturation_, double luminosity_ );
    ColorHSL ( void );
    ColorHSL ( const Color & color_ );
    /* virtual */  ~ColorHSL ( );

    void           hue ( double hue_ );
    double         hue ( void ) const;

    void           saturation ( double saturation_ );
    double         saturation ( void ) const;

    void           luminosity ( double luminosity_ );
    double         luminosity ( void ) const;
```

```
    // Assignment operator from base class
    ColorHSL& operator= ( const Color& color_ );
};
```

ColorYUV

Representation of a color in YUV colorspace (used to encode color for television transmission).

Argument ranges:
Y: 0.0 through 1.0
U: -0.5 through 0.5
V: -0.5 through 0.5

```
class ColorYUV : public Color
{
  public:
    ColorYUV ( double y_, double u_, double v_ );
    ColorYUV ( void );
    ColorYUV ( const Color & color_ );
    /* virtual */ ~ColorYUV ( void );

    void           u ( double u_ );
    double         u ( void ) const;

    void           v ( double v_ );
    double         v ( void ) const;

    void           y ( double y_ );
    double         y ( void ) const;

    // Assignment operator from base class
    ColorYUV& operator= ( const Color& color_ );
};
```

22.4 Magick::Drawable

Drawable provides a convenient interface for preparing vector, image, or text arguments for the Image::draw() method. Each instance of a Drawable sub-class represents a single drawable object. Drawable objects may be drawn "one-by-one" via multiple invocations of the Image draw() method, or may be drawn "all-at-once" by passing a list of Drawable objects to the Image draw() method. The one-by-one approach is convenient for simple drawings, while the list-based approach is appropriate for drawings which require more sophistication.

The following is an example of using the Drawable subclasses with the one-by-one approach to draw the following figure:

[Drawable_example_1.png]

```
#include <string>
#include <iostream>
#include <Magick++.h>

using namespace std;
using namespace Magick;

int main(int /*argc*/,char **/*argv*/)
{
  try {
    // Create base image (white image of 300 by 200 pixels)
    Image image( Geometry(300,200), Color("white") );

    // Set draw options
    image.strokeColor("red"); // Outline color
    image.fillColor("green"); // Fill color
    image.strokeWidth(5);

    // Draw a circle
    image.draw( DrawableCircle(100,100, 50,100) );

    // Draw a rectangle
    image.draw( DrawableRectangle(200,200, 270,170) );

    // Display the result
    image.display( );
  }
  catch( exception &error_ )
    {
      cout << "Caught exception: " << error_.what() << endl;
      return 1;
    }

  return 0;
}
```

Since Drawable is an object it may be saved in an array or a list for later (perhaps repeated) use. The following example shows how to draw the same figure using the list-based approach

```
#include <string>
#include <iostream>
#include <list>
#include <Magick++.h>

using namespace std;
using namespace Magick;

int main(int /*argc*/,char **/*argv*/)
{
  try {
```

```
    // Create base image (white image of 300 by 200 pixels)
    Image image( Geometry(300,200), Color("white") );

    // Construct drawing list
    std::list<Magick::Drawable> drawList;

    // Add some drawing options to drawing list
    drawList.push_back(DrawableStrokeColor("red")); // Outline color
    drawList.push_back(DrawableStrokeWidth(5)); // Stroke width
    drawList.push_back(DrawableFillColor("green")); // Fill color

    // Add a Circle to drawing list
    drawList.push_back(DrawableCircle(100,100, 50,100));

    // Add a Rectangle to drawing list
    drawList.push_back(DrawableRectangle(200,100, 270,170));

    // Draw everything using completed drawing list
    image.draw(drawList);

    // Display the result
    image.display( );
  }
  catch( exception &error_ )
    {
      cout << "Caught exception: " << error_.what() << endl;
      return 1;
    }

  return 0;
}
```

Drawable depends on the simple Coordinate structure which represents a pair of x,y coodinates. The methods provided by the Coordinate structure are shown in the following table:

Coordinate Structure Methods

Method/Member	Signature	Description
Coordinate	void	Default Constructor
	double x_, double y_	Constructor, setting first & second
x	double x_	x coordinate member
y	double y_	y coordinate member

The Drawable classes are shown in the following table:

Drawable Classes

Sub-Class	Constructor Signature	Description
DrawableAffine	double sx_, double sy_, double rx_, double ry_, double tx_, double ty_	Set scaling, rotation, and translation (coordinate transformation).
DrawableAngle	double angle_	Set drawing angle
DrawableArc	double startX_, double startY_, double endX_, double endY_, double startDegrees, double endDegrees_	Draw an arc using the stroke color and based on the circle starting at coordinates startX_,startY_, and ending with coordinates endX_,endY_, and bounded by the rotational arc startDegrees_,endDegrees_

DrawableBezier	const std::list<Magick::Coordinate> &coordinates_	Draw a Bezier curve using the stroke color and based on the coordinates specified by the coordinates_ list.
DrawableCircle	double originX_, double originY_, double perimX_, double perimY_	Draw a circle using the stroke color and thickness using specified origin and perimeter coordinates. If a fill color is specified, then the object is filled.
DrawableColor	double x_, double y_, PaintMethod paintMethod_	Color image according to paintMethod. The point method recolors the target pixel. The replace method recolors any pixel that matches the color of the target pixel. Floodfill recolors any pixel that matches the color of the target pixel and is a neighbor, whereas filltoborder recolors any neighbor pixel that is not the border color. Finally, reset recolors all pixels.
DrawableCompositeImage	double x_, double y_, const std::string &filename_	Composite current image with contents of specified image, at specified coordinates. If the matte attribute is set to true, then the image composition will consider an alpha channel, or transparency, present in the image file so that non-opaque portions allow part (or all) of the composite image to show through.
	double x_, double y_, const Image &image_	
	double x_, double y_, double width_, double height_, const std::string &filename_	Composite current image with contents of specified image, rendered with specified width and height, at specified coordinates. If the matte attribute is set to true, then the image composition will consider an alpha channel, or transparency, present in the image file so that non-opaque portions allow part (or all) of the composite image to show through. If the specified width or height is zero, then the image is composited at its natural size, without enlargement or reduction.
	double x_, double y_, double width_, double height_, const Image &image_	

		Composite current image with contents of specified image, rendered with specified width and height, using specified composition algorithm, at specified coordinates. If the matte attribute is set to true, then the image
	double x_, double y_, double width_, double height_, const std::string &filename_, CompositeOperator composition_	composition will consider an alpha channel, or transparency, present in the image file so that non-opaque portions allow part (or all) of the composite image to show through. If the specified width or height is zero, then the image is composited at its natural size, without enlargement or reduction.
	double x_, double y_, double width_, double height_, const Image &image_, CompositeOperator composition_	
DrawableTextDecoration	DecorationType decoration_	Specify decoration to apply to text.
DrawableDashArray	const unsigned int* dasharray_	Specify the pattern of dashes and gaps used to stroke paths. The strokeDashArray represents a zero-terminated array of numbers that specify the lengths of alternating dashes and gaps in pixels. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. A typical strokeDashArray_ array might contain the members 5 3 2 0, where the zero value indicates the end of the pattern array.
DrawableDashOffset	unsigned int offset_	Specify the distance into the dash pattern to start the dash. See documentation on SVG’s stroke-dashoffset property for usage details.
DrawableEllipse	double originX_, double originY_, double radiusX_, double radiusY_, double arcStart_, double arcEnd_	Draw an ellipse using the stroke color and thickness, specified origin, x & y radius, as well as specified start and end of arc in degrees. If a fill color is specified, then the object is filled.
DrawableFillColor	const Color &color_	Specify drawing object fill color.
		Specify the algorithm which is to be used to determine what parts of the canvas are

DrawableFillRule	FillRule fillRule_	included inside the shape. See documentation on SVG's fill-rule property for usage details.
DrawableFillOpacity	double opacity_	Specify opacity to use when drawing using fill color.
DrawableFont	const std::string &font_	Specify font name to use when drawing text.
	const std::string &family_, StyleType style_, unsigned long weight_, StretchType stretch_	Specify font family, style, weight (one of the set { 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 } with 400 being the normal size), and stretch to be used to select the font used when drawing text. Wildcard matches may be applied to style via the AnyStyle enumeration, applied to weight if weight is zero, and applied to stretch via the AnyStretch enumeration.
DrawableGravity	GravityType gravity_	Specify text positioning gravity.
DrawableLine	double startX_, double startY_, double endX_, double endY_	Draw a line using stroke color and thickness using starting and ending coordinates
DrawableMatte	double x_, double y_, PaintMethod paintMethod_	Change the pixel matte value to transparent. The point method changes the matte value of the target pixel. The replace method changes the matte value of any pixel that matches the color of the target pixel. Floodfill changes the matte value of any pixel that matches the color of the target pixel and is a neighbor, whereas filltoborder changes the matte value of any neighbor pixel that is not the border color, Finally reset changes the matte value of all pixels.
DrawableMiterLimit	unsigned int miterLimit_	Specify miter limit. When two line segments meet at a sharp angle and miter joins have been specified for 'lineJoin', it is possible for the miter to extend far beyond the thickness of the line stroking the path. The miterLimit' imposes a limit on the ratio of the miter length to the 'lineWidth'. The default value of this

		parameter is 4.
DrawablePath	const std::list<Magick::VPath> &path_	Draw on image using vector path.
DrawablePoint	double x_, double y_	Draw a point using stroke color and thickness at coordinate
DrawablePointSize	double pointSize_	Set font point size.
DrawablePolygon	const std::list<Magick::Coordinate> &coordinates_	Draw an arbitrary polygon using stroke color and thickness consisting of three or more coordinates contained in an STL list. If a fill color is specified, then the object is filled.
DrawablePolyline	const std::list<Magick::Coordinate> &coordinates_	Draw an arbitrary polyline using stroke color and thickness consisting of three or more coordinates contained in an STL list. If a fill color is specified, then the object is filled.
DrawablePopGraphicContext	void	Pop Graphic Context. Removing the current graphic context from the graphic context stack restores the options to the values they had prior to the preceding DrawablePushGraphicContext operation.
DrawablePushGraphicContext	void	Push Graphic Context. When a graphic context is pushed, options set after the context is pushed (such as coordinate transformations, color settings, etc.) are saved to a new graphic context. This allows related options to be saved on a graphic context "stack" in order to support heirarchical nesting of options. When DrawablePopGraphicContext is used to pop the current graphic context, the options in effect during the last DrawablePushGraphicContext operation are restored.
DrawablePushPattern	std::string &id_, long x_, long y_, long width_, long height_	Start a pattern definition with arbitrary pattern name specified by id_, pattern offset specified by x_ and y_, and pattern size specified by width_ and height_. The pattern is defined within the coordinate system defined by the specified offset and size. Arbitrary drawing objects (including DrawableCompositeImage) may

		be specified between DrawablePushPattern and DrawablePopPattern in order to draw the pattern. Normally the pair DrawablePushGraphicContext & DrawablePopGraphicContext are used to enclose a pattern definition. Pattern definitions are terminated by a DrawablePopPattern object.
DrawablePopPattern	void	Terminate a pattern definition started via DrawablePushPattern.
DrawableRectangle	double upperLeftX_, double upperLeftY_, double lowerRightX_, double lowerRightY	Draw a rectangle using stroke color and thickness from upper-left coordinates to lower-right coordinates. If a fill color is specified, then the object is filled.
DrawableRotation	double angle_	Set rotation to use when drawing (coordinate transformation).
DrawableRoundRectangle	double centerX_, double centerY_, double width_, double hight_, double cornerWidth_, double cornerHeight_	Draw a rounded rectangle using stroke color and thickness, with specified center coordinate, specified width and height, and specified corner width and height. If a fill color is specified, then the object is filled.
DrawableScaling	double x_, double y_	Apply scaling in x and y direction while drawing objects (coordinate transformation).
DrawableSkewX	double angle_	Apply Skew in X direction (coordinate transformation)
DrawableSkewY	double angle_	Apply Skew in Y direction
DrawableStrokeAntialias	bool flag_	Antialias while drawing lines or object outlines.
DrawableStrokeColor	const Color &color_	Set color to use when drawing lines or object outlines.
DrawableStrokeLineCap	LineCap linecap_	Specify the shape to be used at the end of open subpaths when they are stroked. Values of LineCap are UndefinedCap, ButtCap, RoundCap, and SquareCap.
DrawableStrokeLineJoin	LineJoin linejoin_	Specify the shape to be used at the corners of paths (or other vector shapes) when they are stroked. Values of LineJoin are UndefinedJoin, MiterJoin, RoundJoin, and

		BevelJoin.
DrawableStrokeOpacity	double opacity_	Opacity to use when drawing lines or object outlines.
DrawableStrokeWidth	double width_	Set width to use when drawing lines or object outlines.
DrawableText	double x_, double y_, std::string text_	Annotate image with text using stroke color, font, font pointsize, and box color (text background color), at specified coordinates. If text contains special format characters the image filename, type, width, height, or other image attributes may be incorporated in the text (see label()).
DrawableTranslation	double x_, double y_	Apply coordinate translation (set new coordinate origin).
DrawableTextAntialias	bool flag_	Antialias while drawing text.
DrawableViewbox	unsigned long x1_, unsigned long y1_, unsigned long x2_, unsigned long y2_	Dimensions of the output viewbox. If the image is to be written to a vector format (e.g. MVG or SVG), then a DrawablePushGraphicContext() object should be pushed to the head of the list, followed by a DrawableViewbox() statement to establish the output canvas size. A matching DrawablePopGraphicContext() object should be pushed to the tail of the list.

Vector Path Classes

The vector paths supported by Magick++ are based on those supported by the SVG XML specification. Vector paths are not directly drawable, they must first be supplied as a constructor argument to the DrawablePath class in order to create a drawable object. The DrawablePath class effectively creates a drawable compound component which may be replayed as desired. If the drawable compound component consists only of vector path objects using relative coordinates then the object may be positioned on the image by preceding it with a DrawablePath which sets the current drawing coordinate. Alternatively coordinate transforms may be used to translate the origin in order to position the object, rotate it, skew it, or scale it.

The "moveto" commands

The "moveto" commands establish a new current point. The effect is as if the "pen" were lifted and moved to a new location. A path data segment must begin with either one of the "moveto" commands or one of the "arc" commands. Subsequent "moveto" commands (i.e., when the "moveto" is not the first command) represent the start of a new subpath:

Moveto Classes

Sub-Class	Constructor Signature	Description
PathMovetoAbs	const Magick::Coordinate &coordinate_	Start a new sub-path at the given coordinate. PathMovetoAbs indicates that absolute coordinates will follow; PathMovetoRel indicates that relative coordinates will follow. If a relative moveto appears as the first element of the path, then it is treated as a pair of absolute coordinates. If a moveto is followed by multiple pairs of coordinates, the subsequent pairs are treated as implicit lineto commands.
	const std::list<Magick::Coordinate> &coordinates_	
PathMovetoRel	const Magick::Coordinate &coordinate_	
	const std::list<Magick::Coordinate> &coordinates_	

The "closepath" command

The "closepath" command causes an automatic straight line to be drawn from the current point to the initial point of the current subpath:

Closepath Classes

Sub-Class	Constructor Signature	Description
PathClosePath	void	Close the current subpath by drawing a straight line from the current point to current subpath's most recent starting point (usually, the most recent moveto point).

The "lineto" commands

The various "lineto" commands draw straight lines from the current point to a new point:

Lineto Classes

Sub-Class	Constructor Signature	Description
		Draw a line from the current point to the given coordinate which becomes the new current point. PathLinetoAbs indicates that absolute

		coordinates are used; PathLinetoRel indicates
PathLinetoAbs	const Magick::Coordinate& coordinate_	that relative coordinates are used. A number of coordinates pairs may be specified in a list to draw a polyline. At the end of the command, the new current point is set to the final set of coordinates provided.
	const std::list<Magick::Coordinate> &coordinates_	
PathLinetoRel	const Magick::Coordinate& coordinate_	
	const std::list<Magick::Coordinate> &coordinates_	
		Draws a horizontal line from the current point (cpx, cpy) to (x, cpy). PathLinetoHorizontalAbs indicates that absolute coordinates are supplied;
PathLinetoHorizontalAbs	double x_	PathLinetoHorizontalRel indicates that relative coordinates are supplied. At the end of the command, the new current point becomes (x, cpy) for the final value of x.
	PathLinetoHorizontalRel	double x_
		Draws a vertical line from the current point (cpx, cpy) to (cpx, y). PathLinetoVerticalAbs indicates that absolute coordinates are supplied;
PathLinetoVerticalAbs	double y_	PathLinetoVerticalRel indicates that relative coordinates are supplied. At the end of the command, the new current point becomes (cpx, y) for the final value of y.
	PathLinetoVerticalRel	double y_

The curve commands

These three groups of commands draw curves:

* Cubic Bezier commands. A cubic Bezier segment is defined by a start point, an end point, and two control points.
* Quadratic Bezier commands. A quadratic Bezier segment is defined by a

start point, an end point, and one control point.
* Elliptical arc commands. An elliptical arc segment draws a segment of an ellipse.

The cubic Bezier curve commands

The cubic Bezier commands depend on the PathCurvetoArgs argument class, which has the constructor signature

```
PathCurvetoArgs( double x1_, double y1_,
                 double x2_, double y2_,
                 double x_, double y_ );
```

The commands are as follows:

Cubic Bezier Curve Classes

Sub-Class	Constructor Signature	Description
PathCurvetoAbs	const Magick::PathCurvetoArgs &args_ const std::list<Magick::PathCurvetoArgs> &args_	Draws a cubic Bezier curve from the current point to (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve. PathCurvetoAbs indicates that absolutecoordinates will follow; PathCurvetoRel indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polyBezier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polyBezier.
PathCurvetoRel	const Magick::PathCurvetoArgs &args_ const std::list<Magick::PathCurvetoArgs> &args_	Draws a cubic Bezier curve from the current point to (x,y). The first control point is assumed to be the reflection of the second control point on the previous command relative to the current point. (If there is no previous

		command or if the previous command was not an PathCurvetoAbs, PathCurvetoRel, PathSmoothCurvetoAbs or PathSmoothCurvetoRel, assume the first control point is coincident with the current point.)
PathSmoothCurvetoAbs	const Magick::Coordinate &coordinates_	(x2,y2) is the second control point (i.e., the control point at the end of the curve). PathSmoothCurvetoAbs indicates that absolute coordinates will follow; PathSmoothCurvetoRel indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polyBezier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polyBezier.
	const std::list<Magick::Coordinate> &coordinates_	
PathSmoothCurvetoRel	const Magick::Coordinate &coordinates_	
	const std::list<Magick::Coordinate> &coordinates_	

The quadratic Bezier curve commands

The quadratic Bezier commands depend on the PathQuadraticCurvetoArgs argument class, which has the constructor signature:

```
PathQuadraticCurvetoArgs( double x1_, double y1_,
                          double x_, double y_ );
```

The quadratic Bezier commands are as follows:

Quadratic Bezier Curve Classes

Sub-Class	Constructor Signature	Description
		Draws a quadratic Bezier curve from the current point to (x,y) using (x1,y1) as the control point. PathQuadraticCurvetoAbs indicates that absolute coordinates will follow;

PathQuadraticCurvetoAbs	const Magick::PathQuadraticCurvetoArgs &args_	PathQuadraticCurvetoRel indicates that relative coordinates will follow. Multiple sets of coordinates may be specified to draw a polyBezier. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polyBezier.
	const std::list<Magick::PathQuadraticCurvetoArgs> &args_	
PathQuadraticCurvetoRel	const Magick::PathQuadraticCurvetoArgs &args_	
	const std::list<Magick::PathQuadraticCurvetoArgs> &args_	
PathSmoothQuadraticCurvetoAbs	const Magick::Coordinate &coordinate_	Draws a quadratic Bezier curve from the current point to (x,y). The control point is assumed to be the reflection of the control point on the previous command relative to the current point. (If there is no previous command or if the previous command was not a PathQuadraticCurvetoAbs, PathQuadraticCurvetoRel, PathSmoothQuadraticCurvetoAbs or PathSmoothQuadraticCurvetoRel, assume the control point is coincident with the current point.) PathSmoothQuadraticCurvetoAbs indicates that absolute coordinates will follow; PathSmoothQuadraticCurvetoRel indicates that relative coordinates will follow. At the end of the command, the new current point becomes the final (x,y) coordinate pair used in the polyBezier.
	const std::list<Magick::Coordinate> &coordinates_	
PathSmoothQuadraticCurvetoRel	const Magick::Coordinate &coordinate_	
	const std::list<Magick::Coordinate> &coordinates_	

The elliptical arc curve commands

The elliptical arc curve commands depend on the PathArcArgs argument class, which has the constructor signature:

```
PathArcArgs( double radiusX_, double radiusY_,
             double xAxisRotation_, bool largeArcFlag_,
             bool sweepFlag_, double x_, double y_ );
```

The elliptical arc commands are as follows:

Elliptical Arc Curve Classes

Sub-Class	Constructor Signature	Description
PathArcAbs	const Magick::PathArcArgs &coordinates_	Draws an elliptical arc from the current point to (x, y). The size and orientation of the ellipse are defined by two radii (radiusX, radiusY) and an xAxisRotation, which indicates how the ellipse as a whole is rotated relative to the current coordinate system. The center (cx, cy) of the ellipse is calculated automatically to satisfy the constraints imposed by the other parameters. largeArcFlag and sweepFlag contribute to the automatic calculations and help determine how the arc is drawn. If largeArcFlag is true then draw the larger of the available arcs. If sweepFlag is true, then draw the arc matching a clock-wise rotation.
	const std::list<Magick::PathArcArgs> &coordinates_	
PathArcRel	const Magick::PathArcArgs &coordinates_	
	const std::list<Magick::PathArcArgs> &coordinates_	

22.5 Magick::Exception Classes

Exception represents the base class of objects thrown when ImageMagick reports an error. Magick++ throws C++ exceptions synchronous with the operation when an error is detected. This allows errors to be trapped within the enclosing code (perhaps the code to process a single image) while allowing the code to be written simply.

A try/catch block should be placed around any sequence of operations which can be considered a unit of work. For example, if your program processes lists of images and some of these images may be defective, by placing the try/catch block around the entire sequence of code that processes one image (including instantiating the image object), you can minimize the overhead of error checking while ensuring that all objects created to deal with that object are safely destroyed (C++ exceptions unroll the stack until the enclosing try block, destroying any created objects).

The pseudocode for the main loop of your program may look like:

```
for each image in list
  try {
    create image object
    read image
    process image
    save result
  }
  catch( ErrorFileOpen &error )
  {
    process Magick++ file open error
  }
  catch( Exception &error )
  {
    process any Magick++ error
  }
  catch( exception &error )
  {
    process any other exceptions derived from standard C++ exception
  }
  catch( ... )
  {
    process *any* exception (last-ditch effort)
  }
```

This catches errors opening a file first, followed by any Magick++ exception if the exception was not caught previously.

The Exception class is derived from the C++ standard exception class. This means that it contains a C++ string containing additional information about the error (e.g to display to the user). Obtain access to this string via the what() method. For example:

```
 catch( Exception &error_ )
    {
      cout << "Caught exception: " << error_.what() << endl;
    }
```

The classes Warning and Error derive from the Exception class. Exceptions derived from Warning are thrown to represent non-fatal errors which may effect the completeness or quality of the result (e.g. one image provided as an argument to montage is defective). In most cases, a Warning exception may be ignored by catching it immediately, processing it (e.g. printing a diagnostic) and continuing on. Exceptions derived from Error are thrown to represent fatal errors that can not produce a valid result (e.g. attempting to read a file which does not exist).

The specific derived exception classes are shown in the following tables:

Warning Sub-Classes

Warning	Warning Description
WarningUndefined	Unspecified warning type.
WarningResourceLimit	A program resource is exhausted (e.g. not enough memory).
WarningXServer	An X resource is unavailable.
WarningOption	An option was malformed or out of range.
WarningDelegate	An ImageMagick delegate returned an error.
WarningMissingDelegate	The image type can not be read or written because the appropriate Delegate is missing.
WarningCorruptImage	The image file is corrupt (or otherwise can’t be read).
WarningFileOpen	The image file could not be opened (permission problem, wrong file type, or does not exist).
WarningBlob	A binary large object could not be allocated.
WarningCache	Pixels could not be saved to the pixel cache.

Error Sub-Classes

Error	Error Description
ErrorUndefined	Unspecified error type.
ErrorResourceLimit	A program resource is exhausted (e.g. not enough memory).
ErrorXServer	An X resource is unavailable.
ErrorOption	An option was malformed or out of range.
ErrorDelegate	An ImageMagick delegate returned an error.
ErrorMissingDelegate	The image type can not be read or written because the appropriate Delegate is missing.
ErrorCorruptImage	The image file is corrupt (or otherwise can’t be read).
ErrorFileOpen	The image file could not be opened (permission problem, wrong file type, or does not exist).
ErrorBlob	A binary large object could not be allocated.
ErrorCache	Pixels could not be saved to the pixel cache.

22.6 Magick::Geometry

Geometry provides a convenient means to specify a geometry argument. The object may be initialized from a C string or C++ string containing a geometry specification. It may also be initialized by more efficient parameterized constructors.

X11 Geometry Specifications

X11 geometry specifications are in the form "<width>x<height>{+-}<xoffset>{+-}<yoffset>" (where width, height, xoffset, and yoffset are numbers) for specifying the size and placement location for an object.

The width and height parts of the geometry specification are measured in pixels. The xoffset and yoffset parts are also measured in pixels and are used to specify the distance of the placement coordinate from the left and top edges of the image, respectively.

+xoffset The left edge of the object is to be placed xoffset pixels in from the left edge of the image.

-xoffset The left edge of the object is to be placed outside the image, xoffset pixels from the left edge of the image.

The Y offset has similar meanings:

+yoffset The top edge of the object is to be yoffset pixels below the top edge of the image.

-yoffset The top edge of the object is to be outside the image, yoffset pixels above the top edge of the image.

Offsets must be given as pairs; in other words, in order to specify either xoffset or yoffset both must be present.

ImageMagick Extensions To X11 Geometry Specifications

ImageMagick has added a number of qualifiers to the standard geometry string for use when resizing images. The form of an extended geometry string is "<width>x<height>{+-}<xoffset>{+-}<yoffset>{%}{!}{<}{>}". Extended geometry strings should only be used when resizing an image. Using an extended geometry string for other applications may cause the API call to fail. The available qualifiers are shown in the following table:

ImageMagick Geometry Qualifiers

Qualifier	Description
%	Interpret width and height as a percentage of the current size.
!	Resize to width and height exactly, loosing original aspect ratio.
<	Resize only if the image is smaller than the geometry specification.
>	Resize only if the image is greater than the geometry specification.

Postscript Page Size Extension To Geometry Specifications

Any geometry string specification supplied to the Geometry contructor is considered to be a Postscript page size nickname if the first character is not numeric. The Geometry constructor converts these page size specifications into the equivalent numeric geometry string specification (preserving any offset component) prior to conversion to the internal object format. Postscript page size specifications are short-hand for the pixel geometry required to fill a page of that size. Since the 11x17 inch page size used in the US starts with a digit, it is not supported as a Postscript page size nickname. Instead, substitute the geometry specification "792x1224>" when 11x17 output is desired.

An example of a Postscript page size specification is "letter+43+43>".

Postscript Page Size Nicknames

Postscript Page Size Nickname	Equivalent Extended Geometry Specification
Ledger	1224x792>
Legal	612x1008>
Letter	612x792>
LetterSmall	612x792>
ArchE	2592x3456>
ArchD	1728x2592>
ArchC	1296x1728>
ArchB	864x1296>
ArchA	648x864>
A0	2380x3368>
A1	1684x2380>
A2	1190x1684>
A3	842x1190>
A4	595x842>
A4Small	595x842>
A5	421x595>
A6	297x421>
A7	210x297>
A8	148x210>
A9	105x148>
A10	74x105>
B0	2836x4008>
B1	2004x2836>
B2	1418x2004>
B3	1002x1418>
B4	709x1002>

B5	501x709>
C0	2600x3677>
C1	1837x2600>
C2	1298x1837>
C3	918x1298>
C4	649x918>
C5	459x649>
C6	323x459>
Flsa	612x936>
Flse	612x936>
HalfLetter	396x612>

Geometry Methods

Geometry provides methods to initialize its value from strings, from a set of parameters, or via attributes. The methods available for use in Geometry are shown in the following table:

Geometry Methods

Method	Return Type	Signature(s)	Description
Geometry		unsigned int width_, unsigned int height_, unsigned int xOff_ = 0, unsigned int yOff_ = 0, bool xNegative_ = false, bool yNegative_ = false	Construct X11 geometry via explicit parameters.
	const string geometry_	Construct geometry from C++ string	
	const char * geometry_	Construct geometry from C string	
width	void	unsigned int width_	Width
	unsigned int	void	
height	void	unsigned int height_	Height
	unsigned int	void	
xOff	void	unsigned int xOff_	X offset from origin
	int	void	
yOff	void	unsigned int yOff_	Y offset from origin
	int	void	
xNegative	void	bool xNegative_	Sign of X offset negative? (X origin at right)

	bool	void	
yNegative	void	bool yNegative_	Sign of Y offset negative? (Y origin at bottom)
	bool	void	
percent	void	bool percent_	Width and height are expressed as percentages
	bool	void	
aspect	void	bool aspect_	Resize without preserving aspect ratio (!)
	bool	void	
greater	void	bool greater_	Resize if image is greater than size (>)
	bool	void	
less	void	bool less_	Resize if image is less than size (<)
	bool	void	
isValid	void	bool isValid_	Does object contain valid geometry?
	bool	void	
operator =	const Geometry&	const string geometry_	Set geometry via C++ string
operator =	const Geometry&	const char * geometry_	Set geometry via C string
operator string	string	Geometry&	Obtain C++ string representation of geometry
operator<<	ostream&	ostream& stream_, const Geometry& geometry_	Stream onto ostream

22.7 Magick::Image Class

Quick Contents

Image is the primary object in Magick++ and represents a single image frame (see design). The STL interface must be used to operate on image sequences or images (e.g. of format GIF, TIFF, MIFF, Postscript, & MNG) which are comprized of multiple image frames. Individual frames of a multi-frame image may be requested by adding array-style notation to the end of the file name (e.g. "animation.gif[3]" retrieves the fourth frame of a GIF animation. Various image manipulation operations may be applied to the image. Attributes may be set on the image to influence the operation of the manipulation operations. The Pixels class provides low-level access to image pixels. As a convenience, including <Magick++.h> is sufficient in order to use the complete Magick++ API. The Magick++ API is enclosed within the Magick namespace so you must either add the prefix "Magick::" to each class/enumeration name or add the statement "using namespace Magick;" after including the Magick++.h header.

The preferred way to allocate Image objects is via automatic allocation (on the stack). There is no concern that allocating Image objects on the stack will excessively enlarge the stack since Magick++ allocates all large data objects (such as the actual image data) from the heap. Use of automatic allocation is preferred over explicit allocation (via new) since it is much less error prone and allows use of C++ scoping rules to avoid memory leaks. Use of automatic allocation allows Magick++ objects to be assigned and copied just like the C++ intrinsic data types (e.g. 'int'), leading to clear and easy to read code. Use of automatic allocation leads to naturally exception-safe code since if an exception is thrown, the object is automatically deallocated once the stack unwinds past the scope of the allocation (not the case for objects allocated via new).

Image is very easy to use. For example, here is a the source to a program which reads an image, crops it, and writes it to a new file (the exception handling is optional but strongly recommended):

```
#include <Magick++.h>
#include <iostream>
using namespace std;
using namespace Magick;
int main(int argc,char **argv)
{
  try {
    // Create an image object and read an image
    Image image( "girl.gif" );

    // Crop the image to specified size
    // (Geometry implicitly initialized by char *)
    image.crop("100x100+100+100" );

    // Write the image to a file
    image.write( "x.gif" );
  }
  catch( Exception &error_ )
    {
      cout << "Caught exception: " << error_.what() << endl;
      return 1;
    }
  return 0;
}
```

The following is the source to a program which illustrates the use of Magick++'s efficient reference-counted assignment and copy-constructor operations which minimize use of memory and eliminate unncessary copy operations (allowing Image objects to be efficiently assigned, and copied into containers). The program accomplishes the following:

1. Read master image.
2. Assign master image to second image.
3. Zoom second image to the size 640x480.
4. Assign master image to a third image.
5. Zoom third image to the size 800x600.
6. Write the second image to a file.
7. Write the third image to a file.

```
#include <Magick++.h>
#include <iostream>
using namespace std;
using namespace Magick;
int main(int argc,char **argv)
{
    Image master("horse.jpg");
    Image second = master;
    second.zoom("640x480");
    Image third = master;
    third.zoom("800x600");
    second.write("horse640x480.jpg");
    third.write("horse800x600.jpg");
    return 0;
}
```

During the entire operation, a maximum of three images exist in memory and the image data is never copied.

The following is the source for another simple program which creates a 100 by 100 pixel white image with a red pixel in the center and writes it to a file:

```
#include <Magick++.h>
using namespace std;
using namespace Magick;
int main(int argc,char **argv)
{
    Image image( "100x100", "white" );
    image.pixelColor( 49, 49, "red" );
    image.write( "red_pixel.png" );
    return 0;
}
```

If you wanted to change the color image to grayscale, you could add the lines:

```
image.quantizeColorSpace( GRAYColorspace );
image.colors( 256 );
image.quantize( );
```

or, more simply:

```
image.type( GrayscaleType );
```

prior to writing the image.

BLOBs

While encoded images (e.g. JPEG) are most often written-to and read-from a disk file, encoded images may also reside in memory. Encoded images in memory are known as BLOBs (Binary Large OBjects) and may be represented using the Blob class. The encoded image may be initially placed in memory by reading it directly from a file, reading the image from a database,

memory-mapped from a disk file, or could be written to memory by Magick++. Once the encoded image has been placed within a Blob, it may be read into a Magick++ Image via a constructor or read(). Likewise, a Magick++ image may be written to a Blob via write().

An example of using Image to write to a Blob follows:

```
#include <Magick++.h>
using namespace std;
using namespace Magick;
int main(int argc,char **argv)
{
    // Read GIF file from disk
    Image image( "giraffe.gif" );

    // Write to BLOB in JPEG format
    Blob blob;
    image.magick( "JPEG" ) // Set JPEG output format
    image.write( &blob );

    [ Use BLOB data (in JPEG format) here ]

    return 0;
}
```

likewise, to read an image from a Blob, you could use one of the following examples:

[Entry condition for the following examples is that data is pointer to encoded image data and length represents the size of the data]

```
Blob blob( data, length );
Image image( blob );
```

or

```
Blob blob( data, length );
Image image;
image.read( blob);
```

some images do not contain their size or format so the size and format must be specified in advance:

```
Blob blob( data, length );
Image image;
image.size( "640x480")
image.magick( "RGBA" );
image.read( blob);
```

Constructors

Image may be constructed in a number of ways. It may be constructed from a file, a URL, or an encoded image (e.g. JPEG) contained in an in-memory BLOB. The available Image constructors are shown in the following table:

Image Constructors

Signature	Description
const std::string &imageSpec_	Construct Image by reading from file or URL specified by imageSpec_. Use array notation (e.g. filename[9]) to select a specific scene from a multi-frame image.
const Geometry &size_, const Color &color_	Construct a blank image canvas of specified size and color

Construct Image by reading from encoded image data contained in an in-memory BLOB. Depending on the constructor arguments, the Blob size, depth, magick (format) may also be specified. Some image formats require that size be specified. The default ImageMagick uses for depth depends on the compiled-in Quantum size (8 or 16). If ImageMagick's Quantum size does not match that of the image, the depth may need to be specified. ImageMagick can usually automatically detect the image's format. When a format can't be automatically detected, the format (magick) must be specified.

`const Blob &blob_`

`const Blob &blob_, const Geometry &size_`

`const Blob &blob_, const Geometry &size, unsigned int depth`

`const Blob &blob_, const Geometry &size, unsigned int depth_, const string &magick_`

`const Blob &blob_, const Geometry &size, const string &magick_`

Construct a new Image based on an array of image pixels. The pixel data must be in scanline order top-to-bottom. The data can be character, short int, integer, float, or double. Float and double require the pixels to be normalized [0..1]. The other types are [0..MaxRGB]. For example, to create a 640x480 image from unsigned red-green-blue character data, use

```
Image image( 640, 480, "RGB", 0, pixels );
```

The parameters are as follows:

`const unsigned int width_, const unsigned int height_, std::string map_, const StorageType type_, const void *pixels_`

width_	Width in pixels of the image.
height_	Height in pixels of the image.
map_	This character string can be any combination or order of R = red, G = green, B = blue, A = alpha, C = cyan, Y = yellow M = magenta, and K = black. The ordering reflects the order of the pixels in the supplied pixel array.
type_	Pixel storage type (CharPixel, ShortPixel, IntegerPixel, FloatPixel, or DoublePixel)
pixels_	This array of values contain the pixel components as defined by the map_ and type_ parameters. The length of the arrays must equal the area specified by the width_ and height_ values and type_ parameters.

Image Manipulation Methods

Image supports access to all the single-image (versus image-list) manipulation operations provided by the ImageMagick library. If you must process a multi-image file (such as an animation), the STL interface, which provides a multi-image abstraction on top of Image, must be used.

The operations supported by Image are shown in the following table:

Image Image Manipulation Methods

Method	Signature(s)	Description
addNoise	NoiseType noiseType_	Add noise to image with specified noise type.
annotate	const std::string &text_, const Geometry &location_	Annotate using specified text, and placement location
	string text_, const Geometry &boundingArea_, GravityType gravity_	Annotate using specified text, bounding area, and placement gravity. If boundingArea_ is invalid, then bounding area is entire image.
	const std::string &text_, const Geometry &boundingArea_, GravityType gravity_, double degrees_,	Annotate with text using specified text, bounding area, placement gravity, and rotation. If boundingArea_ is invalid, then bounding area is entire image.
	const std::string &text_, GravityType gravity_	Annotate with text (bounding area is entire image) and placement gravity.
blur	const double radius_ = 1, const double sigma_ = 0.5	Blur image. The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
border	const Geometry &geometry_ = "6x6+0+0"	Border image (add border to image). The color of the border is specified by the borderColor attribute.
channel	ChannelType layer_	Extract channel from image. Use this option to extract a particular channel from the image. MatteChannel for example, is useful for extracting the opacity values from an image.
charcoal	const double radius_ = 1, const double sigma_ = 0.5	Charcoal effect image (looks like charcoal sketch). The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
chop	const Geometry &geometry_	Chop image (remove vertical or horizontal subregion of image)
colorize	const unsigned int opacityRed_, const unsigned int opacityGreen_, const	Colorize image with pen color, using specified percent opacity for red,

	unsigned int opacityBlue_, const Color &penColor_	green, and blue quantums.
	const unsigned int opacity_, const Color &penColor_	Colorize image with pen color, using specified percent opacity.
comment	const string &comment_	Comment image (add comment string to image). By default, each image is commented with its file name. Use this method to assign a specific comment to the image. Optionally you can include the image filename, type, width, height, or other image attributes by embedding special format characters.
composite	const Image &compositeImage_, int xOffset_, int yOffset_, CompositeOperator compose_ = InCompositeOp	Compose an image onto the current image at offset specified by xOffset_, yOffset_ using the composition algorithm specified by compose_.
	const Image &compositeImage_, const Geometry &offset_, CompositeOperator compose_ = InCompositeOp	Compose an image onto the current image at offset specified by offset_ using the composition algorithm specified by compose_.
	const Image &compositeImage_, GravityType gravity_, CompositeOperator compose_ = InCompositeOp	Compose an image onto the current image with placement specified by gravity_ using the composition algorithm specified by compose_.
contrast	unsigned int sharpen_	Contrast image (enhance intensity differences in image)
convolve	unsigned int order_, const double *kernel_	Convolve image. Applies a user-specfied convolution to the image. The order_ parameter represents the number of columns and rows in the filter kernel, and kernel_ is a two-dimensional array of doubles representing the convolution kernel to apply.
crop	const Geometry &geometry_	Crop image (subregion of original image)
cycleColormap	int amount_	Cycle image colormap
despeckle	void	Despeckle image (reduce speckle noise)
display	void	Display image on screen. Caution: if an image format is is not compatable with the display visual (e.g. JPEG on a colormapped display) then the original image will be

		altered. Use a copy of the original if this is a problem.
draw	const Drawable &drawable_	Draw shape or text on image.
	const std::list<Drawable> &drawable_	Draw shapes or text on image using a set of Drawable objects contained in an STL list. Use of this method improves drawing performance and allows batching draw objects together in a list for repeated use.
edge	unsigned int radius_ = 0.0	Edge image (hilight edges in image). The radius is the radius of the pixel neighborhood.. Specify a radius of zero for automatic radius selection.
emboss	const double radius_ = 1, const double sigma_ = 0.5	Emboss image (hilight edges with 3D effect). The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
enhance	void	Enhance image (minimize noise)
equalize	void	Equalize image (histogram equalization)
erase	void	Set all image pixels to the current background color.
flip	void	Flip image (reflect each scanline in the vertical direction)
floodFill-Color	unsigned int x_, unsigned int y_, const Color &fillColor_	Flood-fill color across pixels that match the color of the target pixel and are neighbors of the target pixel. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Color &fillColor_	
	unsigned int x_, unsigned int y_, const Color &fillColor_, const Color &borderColor_	Flood-fill color across pixels starting at target-pixel and stopping at pixels matching specified border color. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Color &fillColor_, const Color &borderColor_	
floodFillOpacity	const long x_, const long y_, const unsigned int opacity_, const PaintMethod method_	Floodfill pixels matching color (within fuzz factor) of target pixel(x,y) with replacement opacity value using method.
	unsigned int x_,	Flood-fill texture across pixels that

floodFill-Texture	unsigned int y_, const Image &texture_	match the color of the target pixel and are neighbors of the target pixel. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Image &texture_	
	unsigned int x_, unsigned int y_, const Image &texture_, const Color &borderColor_	Flood-fill texture across pixels starting at target-pixel and stopping at pixels matching specified border color. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Image &texture_, const Color &borderColor_	
flop	void	Flop image (reflect each scanline in the horizontal direction)
frame	const Geometry &geometry_ = "25x25+6+6"	Add decorative frame around image
	unsigned int width_, unsigned int height_, int x_, int y_, int innerBevel_ = 0, int outerBevel_ = 0	
gamma	double gamma_	Gamma correct image (uniform red, green, and blue correction).
	double gammaRed_, double gammaGreen_, double gammaBlue_	Gamma correct red, green, and blue channels of image.
gaussianBlur	double width_, double sigma_	Gaussian blur image. The number of neighbor pixels to be included in the convolution mask is specified by 'width_'. For example, a width of one gives a (standard) 3x3 convolution mask. The standard deviation of the Gaussian bell curve is specified by 'sigma_'.
implode	double factor_	Implode image (special effect)
label	const string &label_	Assign a label to an image. Use this option to assign a specific label to the image. Optionally you can include the image filename, type, width, height, or scene number in the label by embedding special format characters. If the first character of string is @, the image label is read from a file titled by the remaining characters in the string. When converting to Postscript, use this option to specify a header string to print above the image.
magnify	void	Magnify image by integral size

map	const Image &mapImage_ , bool dither_ = false	Remap image colors with closest color from reference image. Set dither_ to true in to apply Floyd/Steinberg error diffusion to the image. By default, color reduction chooses an optimal set of colors that best represent the original image. Alternatively, you can choose a particular set of colors from an image file with this option.
matteFloodfill	const Color &target_, unsigned int opacity_, long x_, long y_, PaintMethod method_	Floodfill designated area with a replacement opacity value.
medianFilter	const double radius_ = 0.0	Filter image by replacing each pixel component with the median color in a circular neighborhood
minify	void	Reduce image by integral size
modifyImage	void	Prepare to update image. Ensures that there is only one reference to the underlying image so that the underlying image may be safely modified without effecting previous generations of the image. Copies the underlying image to a new image if necessary.
modulate	double brightness_, double saturation_, double hue_	Modulate percent hue, saturation, and brightness of an image
negate	bool grayscale_ = false	Negate colors in image. Replace every pixel with its complementary color (white becomes black, yellow becomes blue, etc.). Set grayscale to only negate grayscale values in image.
normalize	void	Normalize image (increase contrast by normalizing the pixel values to span the full range of color values).
oilPaint	unsigned int radius_ = 3	Oilpaint image (image looks like oil painting)
opacity	unsigned int opacity_	Set or attenuate the opacity channel in the image. If the image pixels are opaque then they are set to the specified opacity value, otherwise they are blended with the supplied opacity value. The value of opacity_ ranges from 0 (completely opaque) to MaxRGB. The defines OpaqueOpacity and TransparentOpacity are available to specify completely opaque or completely transparent, respectively.
opaque	const Color &opaqueColor_, const Color &penColor_	Change color of pixels matching opaqueColor_ to specified penColor_.
		Ping is similar to read except only

		enough of the image is read to determine the image columns, rows, and
ping	const std::string &imageSpec_	filesize. The columns, rows, and fileSize attributes are valid after invoking ping. The image data is not valid after calling ping.
quantize	bool measureError_ = false	Quantize image (reduce number of colors). Set measureError_ to true in order to calculate error attributes.
raise	const Geometry &geometry_ = "6x6+0+0", bool raisedFlag_ = false	Raise image (lighten or darken the edges of an image to give a 3-D raised or lowered effect)
read	const string &imageSpec_	Read image into current object
	const Geometry &size_, const std::string &imageSpec_	Read image of specified size into current object. This form is useful for images that do not specifiy their size or to specify a size hint for decoding an image. For example, when reading a Photo CD, JBIG, or JPEG image, a size request causes the library to return an image which is the next resolution greater or equal to the specified size. This may result in memory and time savings.
	const Blob &blob_	Read encoded image of specified size from an in-memory BLOB into current object. Depending on the method arguments, the Blob size, depth, and format may also be specified. Some image formats require that size be specified. The default ImageMagick uses for depth depends on its Quantum size (8 or 16). If ImageMagick's Quantum size does not match that of the image, the depth may need to be specified. ImageMagick can usually automatically detect the image's format. When a format can't be automatically detected, the format must be specified.
	const Blob &blob_, const Geometry &size_	
	const Blob &blob_, const Geometry &size_, unsigned int depth_	
	const Blob &blob_, const Geometry &size_, unsigned short depth_, const string &magick_	
	const Blob &blob_, const Geometry	

	&size_, const string &magick_	
	const unsigned int width_, const unsigned int height_, std::string map_, const StorageType type_, const void *pixels_	Read image based on an array of image pixels. The pixel data must be in scanline order top-to-bottom. The data can be character, short int, integer, float, or double. Float and double require the pixels to be normalized [0..1]. The other types are [0..MaxRGB]. For example, to create a 640x480 image from unsigned red-green-blue character data, use image.read(640, 480, "RGB", 0, pixels); The parameters are as follows: width_ Width in pixels of the image. height_Height in pixels of the image. map_ This character string can be any combination or order of R = red, G = green, B = blue, A = alpha, C = cyan, Y = yellow M = magenta, and K = black. The ordering reflects the order of the pixels in the supplied pixel array. type_ Pixel storage type (CharPixel, ShortPixel, IntegerPixel, FloatPixel, or DoublePixel) pixels_ This array of values contain the pixel components as defined by the map_ and type_ parameters. The length of the arrays must equal the area specified by the width_ and height_ values and type_ parameters.
reduceNoise	void	Reduce noise in image using a noise peak elimination filter.
	unsigned int order_	
roll	int columns_, int rows_	Roll image (rolls image vertically and horizontally) by specified number of columnms and rows)
rotate	double degrees_	Rotate image counter-clockwise by specified number of degrees.
sample	const Geometry &geometry_	Resize image by using pixel sampling algorithm
scale	const Geometry &geometry_	Resize image by using simple ratio algorithm

segment	double clusterThreshold_ = 1.0, double smoothingThreshold_ = 1.5	Segment (coalesce similar image components) by analyzing the histograms of the color components and identifying units that are homogeneous with the fuzzy c-means technique. Also uses quantizeColorSpace and verbose image attributes. Specify clusterThreshold_, as the number of pixels each cluster must exceed the cluster threshold to be considered valid. SmoothingThreshold_ eliminates noise in the second derivative of the histogram. As the value is increased, you can expect a smoother second derivative. The default is 1.5.
shade	double azimuth_ = 30, double elevation_ = 30, bool colorShading_ = false	Shade image using distant light source. Specify azimuth_ and elevation_ as the position of the light source. By default, the shading results as a grayscale image.. Set colorShading_ to true to shade the red, green, and blue components of the image.
sharpen	const double radius_ = 1, const double sigma_ = 0.5	Sharpen pixels in image. The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
shave	const Geometry &geometry_	Shave pixels from image edges.
shear	double xShearAngle_, double yShearAngle_	Shear image (create parallelogram by sliding image by X or Y axis). Shearing slides one edge of an image along the X or Y axis, creating a parallelogram. An X direction shear slides an edge along the X axis, while a Y direction shear slides an edge along the Y axis. The amount of the shear is controlled by a shear angle. For X direction shears, x degrees is measured relative to the Y axis, and similarly, for Y direction shears y degrees is measured relative to the X axis. Empty triangles left over from shearing the image are filled with the color defined as borderColor.
solarize	double factor_ = 50.0	Solarize image (similar to effect seen when exposing a photographic film to light during the development process)
spread	unsigned int amount_ = 3	Spread pixels randomly within image by specified amount
stegano	const Image &watermark_	Add a digital watermark to the image (based on second image)
stereo	const Image	Create an image which appears in stereo when viewed with red-blue glasses (Red

	&rightImage_	image on left, blue on right)
swirl	double degrees_	Swirl image (image pixels are rotated by degrees)
texture	const Image &texture_	Layer a texture on pixels matching image background color.
threshold	double threshold_	Threshold image
transform	const Geometry &imageGeometry_	Transform image based on image and crop geometries. Crop geometry is optional.
	const Geometry &imageGeometry_, const Geometry &cropGeometry_	
transparent	const Color &color_	Add matte image to image, setting pixels matching color to transparent.
trim	void	Trim edges that are the background color from the image.
unsharpmask	double radius_, double sigma_, double amount_, double threshold_	Replace image with a sharpened version of the original image using the unsharp mask algorithm. The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Gaussian, in pixels. The amount_ parameter specifies the percentage of the difference between the original and the blur image that is added back into the original. The threshold_ parameter specifies the threshold in pixels needed to apply the diffence amount.
wave	double amplitude_ = 25.0, double wavelength_ = 150.0	Alter an image along a sine wave.
write	const string &imageSpec_	Write image to a file using filename imageSpec_. Caution: if an image format is selected which is capable of supporting fewer colors than the original image or quantization has been requested, the original image will be quantized to fewer colors. Use a copy of the original if this is a problem.
	Blob *blob_	Write image to a in-memory BLOBstored in blob_. The magick_ parameter specifies the image format to write (defaults to magick). The depth_ parameter species the image depth (defaults to depth). Caution: if an image format is selected which is capable of supporting fewer colors than the original image or

quantization has been requested, the original image will be quantized to fewer colors. Use a copy of the original if this is a problem.

`Blob *blob_, std::string &magick_`

`Blob *blob_, std::string &magick_, unsigned int depth_`

Write pixel data into a buffer you supply. The data is saved either as char, short int, integer, float or double format in the order specified by the type_ parameter. For example, we want to extract scanline 1 of a 640x480 image as character data in red-green-blue order:

```
image.write(0,0,640,1,"RGB",0,pixels);
```

The parameters are as follows:

`const int x_, const int y_, const unsigned int columns_, const unsigned int rows_, const std::string &map_, const StorageType type_, void *pixels_`

x_	Horizontal ordinate of left-most coordinate of region to extract.
y_	Vertical ordinate of top-most coordinate of region to extract.
columns_	Width in pixels of the region to extract.
rows_	Height in pixels of the region to extract.
map_	This character string can be any combination or order of R = red, G = green, B = blue, A = alpha, C = cyan, Y = yellow, M = magenta, and K = black. The ordering reflects the order of the pixels in the supplied pixel array.
type_	Pixel storage type (CharPixel, ShortPixel, IntegerPixel, FloatPixel, or DoublePixel)
pixels_	This array of values contain the pixel components as defined by the map_ and type_ parameters. The length of the arrays must equal the area specified by the width_ and height_ values and type_ parameters.

zoom	const Geometry &geometry_	Zoom image to specified size.

Image Attributes

Image attributes are set and obtained via methods in Image. Except for methods which accept pointer arguments (e.g. chromaBluePrimary) all methods return attributes by value.

The supported image attributes and the method arguments required to obtain them are shown in the following table:

Image Image Attributes

Attribute	Type	Get Signature	Set Signature	Description
adjoin	bool	void	bool flag_	Join images into a single multi-image file.
antiAlias	bool	void	bool flag_	Control antialiasing of rendered Postscript and Postscript or TrueType fonts. Enabled by default.
animation-Delay	unsigned int (0 to 65535)	void	unsigned int delay_	Time in 1/100ths of a second (0 to 65535) which must expire before displaying the next image in an animated sequence. This option is useful for regulating the animation of a sequence of GIF images within Netscape.
animation-Iterations	unsigned int	void	unsigned int iterations_	Number of iterations to loop an animation (e.g. Netscape loop extension) for.
background-Color	Color	void	const Color &color_	Image background color
background-Texture	string	void	const string &texture_	Image file name to use as the background texture. Does not modify image pixels.
baseColumns	unsigned int	void		Base image width (before transformations)
baseFilename	string	void		Base image filename (before transformations)
baseRows	unsigned int	void		Base image height (before transformations)
borderColor	Color	void	const Color	Image border color

			&color_	
boundingBox	Geometry	void		Return smallest bounding box enclosing non-border pixels. The current fuzz value is used when discriminating between pixels. This is the crop bounding box used by crop(Geometry(0,0)).
boxColor	Color	void	const Color &boxColor_	Base color that annotation text is rendered on.
cacheThreshold	unsigned int		unsigned int	Pixel cache threshold in megabytes. Once this threshold is exceeded, all subsequent pixels cache operations are to/from disk. This is a static method and the attribute it sets is shared by all Image objects.
chroma-BluePrimary	float x & y	float *x_, float *y_	float x_, float y_	Chromaticity blue primary point (e.g. x=0.15, y=0.06)
chroma-GreenPrimary	float x & y	float *x_, float *y_	float x_, float y_	Chromaticity green primary point (e.g. x=0.3, y=0.6)
chroma-RedPrimary	float x & y	float *x_, float *y_	float x_, float y_	Chromaticity red primary point (e.g. x=0.64, y=0.33)
chroma-WhitePoint	float x & y	float *x_, float *y_	float x_, float y_	Chromaticity white point (e.g. x=0.3127, y=0.329)
classType	ClassType	void	ClassType class_	Image storage class. Note that conversion from a DirectClass image to a PseudoClass image may result in a loss of color due to the limited size of the palette (256 or 65535 colors).
clipMask	Image	void	const Image	Associate a clip mask image with the current image. The clip mask image must have the same dimensions as the current image or an exception is thrown.

			&clipMask_	Clipping occurs wherever pixels are transparent in the clip mask image. Clipping Pass an invalid image to unset an existing clip mask.
colorFuzz	double	void	double fuzz_	Colors within this distance are considered equal. A number of algorithms search for a target color. By default the color must be exact. Use this option to match colors that are close to the target color in RGB space.
colorMap	Color	unsigned int index_	unsigned int index_, const Color &color_	Color at color-pallet index.
colorSpace	ColorspaceType colorSpace_	void	ColorspaceType colorSpace_	The colorspace (e.g. CMYK) used to represent the image pixel colors. Image pixels are always stored as RGB(A) except for the case of CMY(K).
columns	unsigned int	void		Image width
comment	string	void		Image comment
compress-Type	CompressionType	void	CompressionType compressType_	Image compresion type. The default is the compression type of the specified image file.
density	Geometry (default 72x72)	void	const Geometry &density_	Vertical and horizontal resolution in pixels of the image. This option specifies an image density when decoding a Postscript or Portable Document page. Often used with psPageSize.
depth	unsigned int (8 or 16)	void	unsigned int depth_	Image depth. Used to specify the bit depth when reading or writing raw images or when the output format supports multiple depths. Defaults to the quantum depth that ImageMagick is compiled with.

endian	EndianType	void	EndianType endian_	Specify (or obtain) endian option for formats which support it.
directory	string	void		Tile names from within an image montage
fileName	string	void	const string &fileName_	Image file name.
fileSize	off_t	void		Number of bytes of the image on disk
fillColor	Color	void	const Color &fillColor_	Color to use when filling drawn objects
fillPattern	Image	void	const Image &fillPattern_	Pattern image to use when filling drawn objects.
fillRule	FillRule	void	const Magick::FillRule &fillRule_	Rule to use when filling drawn objects.
filterType	FilterTypes	void	FilterTypes filterType_	Filter to use when resizing image. The reduction filter employed has a sigificant effect on the time required to resize an image and the resulting quality. The default filter is Lanczos which has been shown to produce high quality results when reducing most images.
font	string	void	const string &font_	Text rendering font. If the font is a fully qualified X server font name, the font is obtained from an X server. To use a TrueType font, precede the TrueType filename with an @. Otherwise, specify a Postscript font name (e.g. "helvetica").
fontPointsize	unsigned int	void	unsigned int pointSize_	Text rendering font point size
fontTypeMetrics	TypeMetric	const std::string &text_, TypeMetric		Update metrics with font type metrics using specified text, and current font and

		*metrics		fontPointSize settings.
format	string	void		Long form image format description.
gamma	double (typical range 0.8 to 2.3)	void		Gamma level of the image. The same color image displayed on two different workstations may look different due to differences in the display monitor. Use gamma correction to adjust for this color difference.
geometry	Geometry	void		Preferred size of the image when encoding.
gifDispose-Method	unsigned int { 0 = Disposal not specified, 1 = Do not dispose of graphic, 3 = Overwrite graphic with background color, 4 = Overwrite graphic with previous graphic. }	void	unsigned int disposeMethod_	GIF disposal method. This option is used to control how successive frames are rendered (how the preceding frame is disposed of) when creating a GIF animation.
iccColorProfile	Blob	void	const Blob &colorProfile_	ICC color profile. Supplied via a Blob since Magick++/ and ImageMagick do not currently support formating this data structure directly. Specifications are available from the International Color Consortium for the format of ICC color profiles.
interlace-Type	InterlaceType	void	InterlaceType interlace_	The type of interlacing scheme (default NoInterlace). This option is used to specify the type of interlacing scheme for raw image formats such as RGB or YUV. NoInterlace means do not interlace, LineInterlace uses scanline interlacing, and PlaneInterlace uses plane

				interlacing. PartitionInterlace is like PlaneInterlace except the different planes are saved to individual files (e.g. image.R, image.G, and image.B). Use LineInterlace or PlaneInterlace to create an interlaced GIF or progressive JPEG image.
iptcProfile	Blob	void	const Blob& iptcProfile_	IPTC profile. Supplied via a Blob since Magick++ and ImageMagick do not currently support formating this data structure directly. Specifications are available from the International Press Telecommunications Council for IPTC profiles.
label	string	void	const string &label_	Image label
magick	string	void	const string &magick_	Get image format (e.g. "GIF")
matte	bool	void	bool matteFlag_	True if the image has transparency. If set True, store matte channel if the image has one otherwise create an opaque one.
matteColor	Color	void	const Color &matteColor_	Image matte (transparent) color
meanError-PerPixel	double	void		The mean error per pixel computed when an image is color reduced. This parameter is only valid if verbose is set to true and the image has just been quantized.
monochrome	bool	void	bool flag_	Transform the image to black and white
montage-Geometry	Geometry	void		Tile size and offset within an image montage. Only valid for montage images.
				The normalized max

normalized-MaxError	double	void		error per pixel computed when an image is color reduced. This parameter is only valid if verbose is set to true and the image has just been quantized.
normalized-MeanError	double	void		The normalized mean error per pixel computed when an image is color reduced. This parameter is only valid if verbose is set to true and the image has just been quantized.
packets	unsigned int	void		The number of runlength-encoded packets in the image
packetSize	unsigned int	void		The number of bytes in each pixel packet
page	Geometry	void	const Geometry &pageSize_	Preferred size and location of an image canvas. Use this option to specify the dimensions and position of the Postscript page in dots per inch or a TEXT page in pixels. This option is typically used in concert with density. Page may also be used to position a GIF image (such as for a scene in an animation)
pixelColor	Color	unsigned int x_, unsigned int y_	unsigned int x_, unsigned int y_, const Color &color_	Get/set pixel color at location x & y.
quality	unsigned int (0 to 100)	void	unsigned int quality_	JPEG/MIFF/PNG compression level (default 75).
quantize-Colors	unsigned int	void	unsigned int colors_	Preferred number of colors in the image. The actual number of colors in the image may be less than your request, but never more. Images with

				less unique colors than specified with this option will have any duplicate or unused colors removed.
quantize-ColorSpace	ColorspaceType	void	ColorspaceType colorSpace_	Colorspace to quantize colors in (default RGB). Empirical evidence suggests that distances in color spaces such as YUV or YIQ correspond to perceptual color differences more closely than do distances in RGB space. These color spaces may give better results when color reducing an image.
quantize-Dither	bool	void	bool flag_	Apply Floyd/Steinberg error diffusion to the image. The basic strategy of dithering is to trade intensity resolution for spatial resolution by averaging the intensities of several neighboring pixels. Images which suffer from severe contouring when reducing colors can be improved with this option. The quantizeColors or monochrome option must be set for this option to take effect.
quantize-TreeDepth	unsigned int	void	unsigned int treeDepth_	Depth of the quantization color classification tree. Values of 0 or 1 allow selection of the optimal tree depth for the color reduction algorithm. Values between 2 and 8 may be used to manually adjust the tree depth.
rendering-Intent	RenderingIntent	void	RenderingIntent render_	The type of rendering intent
resolution-Units	ResolutionType	void	ResolutionType units_	Units of image resolution
rows	unsigned int	void		The number of pixel

				rows in the image
scene	unsigned int	void	unsigned int scene_	Image scene number
signature	string	bool force_ = false		Image MD5 signature. Set force_ to 'true' to force re-computation of signature.
size	Geometry	void	const Geometry &geometry_	Width and height of a raw image (an image which does not support width and height information). Size may also be used to affect the image size read from a multi-resolution format (e.g. Photo CD, JBIG, or JPEG.
strokeAntiAlias	bool	void	bool flag_	Enable or disable anti-aliasing when drawing object outlines.
strokeColor	Color	void	const Color &strokeColor_	Color to use when drawing object outlines
strokeDashOffset	unsigned int	void	double strokeDashOffset_	While drawing using a dash pattern, specify distance into the dash pattern to start the dash (default 0).
strokeDashArray	const double*	void	const double* strokeDashArray_	Specify the pattern of dashes and gaps used to stroke paths. The strokeDashArray represents a zero-terminated array of numbers that specify the lengths (in pixels) of alternating dashes and gaps in user units. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. A typical strokeDashArray_ array might contain the members 5 3 2 0, where the zero value indicates the end of the pattern array.
				Specify the shape to

strokeLineCap	LineCap	void	LineCap lineCap_	Specify the shape to be used at the corners of paths (or other vector shapes) when they are stroked. Values of LineJoin are UndefinedJoin, MiterJoin, RoundJoin, and BevelJoin.
strokeLineJoin	LineJoin	void	LineJoin lineJoin_	Specify the shape to be used at the corners of paths (or other vector shapes) when they are stroked. Values of LineJoin are UndefinedJoin, MiterJoin, RoundJoin, and BevelJoin.
strokeMiterLimit	unsigned int	void	unsigned int miterLimit_	Specify miter limit. When two line segments meet at a sharp angle and miter joins have been specified for 'lineJoin', it is possible for the miter to extend far beyond the thickness of the line stroking the path. The miterLimit' imposes a limit on the ratio of the miter length to the 'lineWidth'. The default value of this parameter is 4.
strokeWidth	double	void	double strokeWidth_	Stroke width for use when drawing vector objects (default one)
strokePattern	Image	void	const Image &strokePattern_	Pattern image to use while drawing object stroke (outlines).
subImage	unsigned int	void	unsigned int subImage_	Subimage of an image sequence
subRange	unsigned int	void	unsigned int subRange_	Number of images relative to the base image
tileName	string	void	const string &tileName_	Tile name
totalColors	unsigned long	void		Number of colors in the image
type	ImageType	void	ImageType	Image type.

verbose	bool	void	bool verboseFlag_	Print detailed information about the image
view	string	void	const string &view_	FlashPix viewing parameters.
x11Display	string (e.g. "hostname:0.0")	void	const string &display_	X11 display to display to, obtain fonts from, or to capture image from
xResolution	double	void		x resolution of the image
yResolution	double	void		y resolution of the image

Raw Image Pixel Access

Image pixels (of type PixelPacket) may be accessed directly via the Image Pixel Cache. The image pixel cache is a rectangular window into the actual image pixels (which may be in memory, memory-mapped from a disk file, or entirely on disk). Two interfaces exist to access the Image Pixel Cache. The interface described here (part of the Image class) supports only one view at a time. See the Pixels class for a more abstract interface which supports simultaneous pixel views (up to the number of rows). As an analogy, the interface described here relates to the Pixels class as stdio's gets() relates to fgets(). The Pixels class provides the more general form of the interface.

Obtain existing image pixels via getPixels(). Create a new pixel region using setPixels().

Depending on the capabilities of the operating system, and the relationship of the window to the image, the pixel cache may be a copy of the pixels in the selected window, or it may be the actual image pixels. In any case calling syncPixels() insures that the base image is updated with the contents of the modified pixel cache. The method readPixels() supports copying foreign pixel data formats into the pixel cache according to the QuantumTypes. The method writePixels() supports copying the pixels in the cache to a foreign pixel representation according to the format specified by QuantumTypes.

The pixel region is effectively a small image in which the pixels may be accessed, addressed, and updated, as shown in the following example:

```
Image image("cow.png");
// Obtain pixel region with size 60x40, and top origin at 20x30

int columns = 60;
PixelPacket *pixel_cache = image.GetPixels(20,30,columns,40);
// Set pixel at column 5, and row 10 in the pixel cache to red.

int column = 5;                                                    [Cache.png]
int row = 10;
PixelPacket *pixel =
pixel_cache+row*columns*sizeof(PixelPacket)+column;
pixel = Color("red");
// Save updated pixel cache back to underlying image
image.syncPixels();
image.write("horse.png");
```

The image cache supports the following methods:

Image Cache Methods

Method	Returns	Signature	Description
getConstPixels	const PixelPacket*	int x_, int y_, unsigned int columns_, unsigned int rows_	Transfers pixels from the image to the pixel cache as defined by the specified rectangular region.
getConstIndexes	const IndexPacket*	void	Returns a pointer to the Image pixel indexes. Only valid for PseudoClass images or CMYKA images. The pixel indexes represent an array of type IndexPacket, with each entry corresponding to an x,y pixel position. For PseudoClass images, the entry's value is the offset into the colormap (see colorMap) for that pixel. For CMYKA images, the indexes are used to contain the alpha channel.
getIndexes	IndexPacket*	void	Returns a pointer to the Image pixel indexes corresponding to the pixel region requested by the last getConstPixels, getPixels, or setPixels call. Only valid for PseudoClass images or CMYKA images. The pixel indexes represent an array of type IndexPacket, with each entry corresponding to a pixel x,y position. For PseudoClass images, the entry's value is the offset into the colormap (see colorMap) for that pixel. For CMYKA images, the indexes are used to contain the alpha channel.
getPixels	PixelPacket*	int x_, int y_, unsigned int columns_, unsigned int rows_	Transfers pixels from the image to the pixel cache as defined by the specified rectangular region. Modified pixels may be subsequently transferred back to the image via syncPixels.

setPixels	PixelPacket*	int x_, int y_, unsigned int columns_, unsigned int rows_	Allocates a pixel cache region to store image pixels as defined by the region rectangle. This area is subsequently transferred from the pixel cache to the image via syncPixels.
syncPixels	void	void	Transfers the image cache pixels to the image.
readPixels	void	QuantumTypes quantum_, unsigned char *source_,	Transfers one or more pixel components from a buffer or file into the image pixel cache of an image. ReadPixels is typically used to support image decoders.
writePixels	void	QuantumTypes quantum_, unsigned char *destination_	Transfers one or more pixel components from the image pixel cache to a buffer or file. WritePixels is typically used to support image encoders.

22.8 Magick::Pixels

The Pixels class provides efficient access to raw image pixels. Image pixels (of type PixelPacket) may be accessed directly via the Image Pixel Cache. The image pixel cache is a rectangular window (a view) into the actual image pixels (which may be in memory, memory-mapped from a disk file, or entirely on disk). Obtain existing image pixels via get(). Create a new pixel region using set().

Depending on the capabilities of the operating system, and the relationship of the window to the image, the pixel cache may be a copy of the pixels in the selected window, or it may be the actual image pixels. In any case calling sync() insures that the base image is updated with the contents of the modified pixel cache. The method decode() supports copying foreign pixel data formats into the pixel cache according to the QuantumTypes. The method encode() supports copying the pixels in the cache to a foreign pixel representation according to the format specified by QuantumTypes.

Setting a view using the Pixels class does not cause the number of references to the underlying image to be reduced to one. Therefore, in order to ensure that only the current generation of the image is modified, the Image's modifyImage() method should be invoked to reduce the reference count on the underlying image to one. If this is not done, then it is possible for a previous generation of the image to be modified due to the use of reference counting when copying or constructing an Image.

The PixelPacket* returned by the set and get methods, and the IndexPacket* returned by the indexes method point to pixel data managed by the Pixels class. The Pixels class is responsible for releasing resources associated with the pixel view. This means that the pointer should never be passed to delete() or free().

The pixel view is a small image in which the pixels may be accessed, addressed, and updated, as shown in the following example, which produces an image similar to the one on the right (minus lines and text):

```
  // Create base image
   Image image(Geometry(254,218), "white");

  // Set image pixels to DirectClass representation
   image.classType( DirectClass );

   // Ensure that there is only one reference to underlying
image
   image.modifyImage();

  // Allocate pixel view
   Pixels view(image);

  // Set all pixels in region anchored at 38x36, with size
160x230 to green.
   unsigned int columns = 196; unsigned int rows = 162;
   Color green("green");
   PixelPacket *pixels = view.get(38,36,columns,rows);
   for ( unsigned int row = 0; row < rows ; ++row )
     for ( unsigned int column = 0; column < columns ; ++column  [Cache.png]
)
       *pixels++=green;
   view.sync();

  // Set all pixels in region anchored at 86x72, with size
108x67 to yellow.
   columns = 108; rows = 67;
   Color yellow("yellow");
   pixels = view.get(86,72,columns,rows);
   for ( unsigned int row = 0; row < rows ; ++row )
      for ( unsigned int column = 0; column < columns ;
```

```
++column )
        *pixels++=yellow;
    view.sync();

  // Set pixel at position 108,94 to red
   *(view.get(108,94,1,1)) = Color("red");
   view.sync();
```

Pixels supports the following methods:

Pixel Cache Methods

Method	Returns	Signature	Description
get	PixelPacket*	int x_, int y_, unsigned int columns_, unsigned int rows_	Transfers pixels from the image to the pixel cache as defined by the specified rectangular region. Modified pixels may be subsequently transferred back to the image via sync. The value returned is intended for pixel access only. It should never be deallocated.
set	PixelPacket*	int x_, int y_, unsigned int columns_, unsigned int rows_	Allocates a pixel cache region to store image pixels as defined by the region rectangle. This area is subsequently transferred from the pixel cache to the image via sync. The value returned is intended for pixel access only. It should never be deallocated.
sync	void	void	Transfers the image cache pixels to the image.
indexes	IndexPacket*	void	Returns the PsuedoColor pixel indexes corresponding to the pixel region defined by the last get or set call. Only valid for PseudoColor and CMYKA images. The pixel indexes (an array of type IndexPacket, which is typedef Quantum, which is itself typedef unsigned char, or unsigned short, depending on the value of the QuantumDepth define) provide the colormap index (see colorMap) for each pixel in the image. For CMYKA images, the indexes represent the matte channel. The value returned is intended for pixel access only. It should never be deallocated.
x	unsigned int	void	Left ordinate of view
y	unsigned int	void	Top ordinate of view
columns	unsigned int	void	Width of view
rows	unsigned int	void	Height of view

22.9 Magick++ STL Support

Magick++ provides a set of Standard Template Libary (STL) algorithms for operating across ranges of image frames in a container. It also provides a set of STL unary function objects to apply an operation on image frames in a container via an algorithm which uses unary function objects. A good example of a standard algorithm which is useful for processing containers of image frames is the STL for_each algorithm which invokes a unary function object on a range of container elements.

Magick++ uses a limited set of template argument types. The current template argument types are:

Container

A container having the properties of a Back Insertion Sequence. Sequences support forward iterators and Back Insertion Sequences support the additional abilty to append an element via push_back(). Common compatable container types are the STL <vector> and <list> template containers. This template argument is usually used to represent an output container in which one or more image frames may be appended. Containers like STL <vector> which have a given default capacity may need to have their capacity adjusted via reserve() to a larger capacity in order to support the expected final size . Since Magick++ images are very small, it is likely that the default capacity of STL <vector> is sufficient for most situations.

InputIterator

An input iterator used to express a position in a container. These template arguments are typically used to represent a range of elements with first_ representing the first element to be processed and last_ representing the element to stop at. When processing the entire contents of a container, it is handy to know that STL containers usually provide the begin() and end() methods to return input interators which correspond with the first and last elements, respectively.

The following is an example of how frames from a GIF animation "test_image_anim.gif" may be appended horizontally with the resulting image written to the file "appended_image.miff":

```
#include <list>
#include <Magick++.h>
using namespace std;
using namespace Magick;

int main(int /*argc*/,char **/*argv*/)
{
   list<Image> imageList;
   readImages( &imageList, "test_image_anim.gif" );

   Image appended;
   appendImages( &appended, imageList.begin(), imageList.end() );
   appended.write( "appended_image.miff" );
   return 0;
}
```

The available Magick++ specific STL algorithms for operating on sequences of image frames are shown in the following table:

Magick++ STL Algorithms For Image Sequences

Algorithm	Signature	Description
animateImages	InputIterator first_, InputIterator last_	Animate a sequence of image frames. Image frames are displayed in succession, creating an animated effect. The animation options are taken from the first image frame. This feature is only supported under X11 at the moment.
appendImages	Image *appendedImage_, InputIterator first_, InputIterator last_, bool stack_ = false	Append a sequence of image frames, writing the result to appendedImage_. All the input image frames must have the same width or height. Image frames of the same width are stacked top-to-bottom. Image frames of the same height are stacked left-to-right. If the stack_ parameter is false, rectangular image frames are stacked left-to-right otherwise top-to-bottom.
averageImages	Image *averagedImage_, InputIterator first_, InputIterator last_	Average a sequence of image frames, writing the result to averagedImage_. All the input image frames must be the same size in pixels.
coalesceImages	InputIterator first_, InputIterator last_	Merge a sequence of images. This is useful for GIF animation sequences that have page offsets and disposal methods. The input images are modified in-place.
deconstructImages	Container *deconstructedImages_, InputIterator first_, InputIterator last_	Break down an image sequence into constituent parts. This is useful for creating GIF or MNG animation sequences. The input sequence is specified by first_ and last_, and the deconstruted images are returned via deconstructedImages_.
displayImages	InputIterator first_, InputIterator last_	Display a sequence of image frames. Through use of a pop-up menu, image frames may be selected in succession. This feature is fully supported under X11 but may have only limited support in other environments. Caution: if an image format is is not compatable with the display visual (e.g. JPEG on a colormapped display) then the original image will be altered. Use a copy of the original if this is a problem.
		Merge a sequence of image frames which represent image

flattenImages	Image *flattendImage_, InputIterator first_, InputIterator last_	layers into a single composited representation. The flattendImage_ parameter points to an existing Image to update with the flattened image. This function is useful for combining Photoshop layers into a single image.
mapImages	InputIterator first_, InputIterator last_, const Image& mapImage_, bool dither_, bool measureError_ = false	Replace the colors of a sequence of images with the closest color from a reference image. Set dither_ to true to enable dithering. Set measureError_ to true in order to evaluate quantization error.
montageImages	Container *montageImages_, InputIterator first_, InputIterator last_, const Montage &montageOpts_	Create a composite image by combining several separate image frames. Multiple frames may be generated in the output container montageImages_ depending on the tile setting and the number of image frames montaged. Montage options are provided via the parameter montageOpts_. Options set in the first image frame (backgroundColor,borderColor, matteColor, penColor,font, and fontPointsize) are also used as options by montageImages().
morphImages	Container *morphedImages_, InputIterator first_, InputIterator last_, unsigned int frames_	Morph a seqence of image frames. This algorithm expands the number of image frames (output to the container morphedImages_) by adding the number of intervening frames specified by frames_ such that the original frames morph (blend) into each other when played as an animation.
mosaicImages	Image *mosaicImage_, InputIterator first_, InputIterator last_	Inlay a number of images to form a single coherent picture. The mosicImage_ argument is updated with a mosaic constructed from the image sequence represented by first_ through last_.
readImages	Container *sequence_, const std::string &imageSpec_	Read a sequence of image frames into existing container (appending to container sequence_) with image names specified in the string imageSpec_.
	Container *sequence_, const Blob &blob_	Read a sequence of image frames into existing container (appending to container sequence_) from Blob blob_.
		Write images in container to file specified by string imageSpec_. Set adjoin_ to false to write a set of image

writeImages	InputIterator first_, InputIterator last_, const std::string &imageSpec_, bool adjoin_ = true	frames via a wildcard imageSpec_ (e.g. image%02d.miff). The wildcard must be one of %0Nd, %0No, or %0Nx. Caution: if an image format is selected which is capable of supporting fewer colors than the original image or quantization has been requested, the original image will be quantized to fewer colors. Use a copy of the original if this is a problem. Write images in container to in-memory BLOB specified by Blob blob_. Set adjoin_ to false to write a set of image frames via a wildcard imageSpec_ (e.g. image%02d.miff). Caution: if an image format is selected which is capable of supporting fewer colors than the original image or quantization has been requested, the original image will be quantized to fewer colors. Use a copy of the original if this is a problem.
quantizeImages	InputIterator first_, InputIterator last_, bool measureError_ = false	Quantize colors in images using current quantization settings. Set measureError_ to true in order to measure quantization error.

Magick++ Unary Function Objects

Magick++ unary function objects inherit from the STL unary_function template class . The STL unary_function template class is of the form

```
unary_function<Arg, Result>
```

and expects that derived classes implement a method of the form:

```
Result operator()( Arg argument_ );
```

which is invoked by algorithms using the function object. In the case of unary function objects defined by Magick++, the invoked function looks like:

```
void operator()( Image &image_);
```

with a typical implementation looking similar to:

```
void operator()( Image &image_ )
  {
    image_.contrast( _sharpen );
  }
```

where contrast is an Image method and _sharpen is an argument stored within the function object by its contructor. Since constructors may be polymorphic, a given function object may have several constructors and

selects the appropriate Image method based on the arguments supplied.

In essence, unary function objects (as provided by Magick++) simply provide the means to construct an object which caches arguments for later use by an algorithm designed for use with unary function objects. There is a unary function object corresponding each algorithm provided by the Image class and there is a contructor available compatable with each synonymous method in the Image class.

The unary function objects that Magick++ provides to support manipulating images are shown in the following table:

Magick++ Unary Function Objects For Image Manipulation

Function Object	Constructor Signatures(s)	Description
addNoiseImage	NoiseType noiseType_	Add noise to image with specified noise type.
annotateImage	const std::string &text_, const Geometry &location_	Annotate with text using specified text, bounding area, placement gravity, and rotation. If boundingArea_ is invalid, then bounding area is entire image.
	std::string text_, const Geometry &boundingArea_, GravityType gravity_	Annotate using specified text, bounding area, and placement gravity. If boundingArea_ is invalid, then bounding area is entire image.
	const std::string &text_, const Geometry &boundingArea_, GravityType gravity_, double degrees_,	Annotate with text using specified text, bounding area, placement gravity, and rotation. If boundingArea_ is invalid, then bounding area is entire image.
	const std::string &text_, GravityType gravity_	Annotate with text (bounding area is entire image) and placement gravity.
blurImage	const double radius_ = 1, const double sigma_ = 0.5	Blur image. The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
borderImage	const Geometry &geometry_ = "6x6+0+0"	Border image (add border to image). The color of the border is specified by the borderColor attribute.
charcoalImage	const double radius_ = 1,	Charcoal effect image (looks like charcoal sketch). The radius_ parameter specifies the radius of the Gaussian, in

	const double sigma_ = 0.5	pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
chopImage	const Geometry &geometry_	Chop image (remove vertical or horizontal subregion of image)
colorizeImage	const unsigned int opacityRed_, const unsigned int opacityGreen_, const unsigned int opacityBlue_, const Color &penColor_	Colorize image with pen color, using specified percent opacity for red, green, and blue quantums.
	const unsigned int opacity_, const Color &penColor_	Colorize image with pen color, using specified percent opacity.
commentImage	const std::string &comment_	Comment image (add comment string to image). By default, each image is commented with its file name. Use this method to assign a specific comment to the image. Optionally you can include the image filename, type, width, height, or other image attributes by embedding special format characters.
compositeImage	const Image &compositeImage_, int xOffset_, int yOffset_, CompositeOperator compose_ = InCompositeOp	Compose an image onto another at specified offset and using specified algorithm
	const Image &compositeImage_, const Geometry &offset_, CompositeOperator compose_ = InCompositeOp	
condenseImage	void	Condense image (Re-run-length encode image in memory).
contrastImage	unsigned int sharpen_	Contrast image (enhance intensity differences in image)
cropImage	const Geometry &geometry_	Crop image (subregion of original image)
cycleColormap-Image	int amount_	Cycle image colormap
despeckleImage	void	Despeckle image (reduce speckle noise)
drawImage	const Drawable &drawable_	Draw shape or text on image.

	const std::list<Drawable> &drawable_	Draw shapes or text on image using a set of Drawable objects contained in an STL list. Use of this method improves drawing performance and allows batching draw objects together in a list for repeated use.
edgeImage	unsigned int radius_ = 0.0	Edge image (hilight edges in image). The radius is the radius of the pixel neighborhood.. Specify a radius of zero for automatic radius selection.
embossImage	const double radius_ = 1, const double sigma_ = 0.5	Emboss image (hilight edges with 3D effect). The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
enhanceImage	void	Enhance image (minimize noise)
equalizeImage	void	Equalize image (histogram equalization)
flipImage	void	Flip image (reflect each scanline in the vertical direction)
floodFill-ColorImage	unsigned int x_, unsigned int y_, const Color &fillColor_	Flood-fill color across pixels that match the color of the target pixel and are neighbors of the target pixel. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Color &fillColor_	
	unsigned int x_, unsigned int y_, const Color &fillColor_, const Color &borderColor_	Flood-fill color across pixels starting at target-pixel and stopping at pixels matching specified border color. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Color &fillColor_, const Color &borderColor_	
floodFill-	unsigned int x_, unsigned	Flood-fill texture across pixels that match the color of the target pixel and are

TextureImage	int y_, const Image &texture_	neighbors of the target pixel. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Image &texture_	
	unsigned int x_, unsigned int y_, const Image &texture_, const Color &borderColor_	Flood-fill texture across pixels starting at target-pixel and stopping at pixels matching specified border color. Uses current fuzz setting when determining color match.
	const Geometry &point_, const Image &texture_, const Color &borderColor_	
flopImage	void	Flop image (reflect each scanline in the horizontal direction)
frameImage	const Geometry &geometry_ = "25x25+6+6"	Add decorative frame around image
	unsigned int width_, unsigned int height_, int x_, int y_, int innerBevel_ = 0, int outerBevel_ = 0	
gammaImage	double gamma_	Gamma correct image (uniform red, green, and blue correction).
	double gammaRed_, double gammaGreen_, double gammaBlue_	Gamma correct red, green, and blue channels of image.
gaussianBlurImage	double width_, double sigma_	Gaussian blur image. The number of neighbor pixels to be included in the convolution mask is specified by 'width_'. For example, a width of one gives a (standard) 3x3 convolution mask. The standard deviation of the Gaussian bell curve is specified by 'sigma_'.
implodeImage	double factor_	Implode image (special effect)
labelImage	const string &label_	Assign a label to an image. Use this option to assign a specific label to the image. Optionally you can include the image filename, type, width, height, or scene number in the label by embedding special format characters. If the first character of string

		is @, the image label is read from a file titled by the remaining characters in the string. When converting to Postscript, use this option to specify a header string to print above the image.
layerImage	LayerType layer_	Extract layer from image. Use this option to extract a particular layer from the image. MatteLayer, for example, is useful for extracting the opacity values from an image.
magnifyImage	void	Magnify image by integral size
mapImage	const Image &mapImage_ , bool dither_ = false	Remap image colors with closest color from reference image. Set dither_ to true in to apply Floyd/Steinberg error diffusion to the image. By default, color reduction chooses an optimal set of colors that best represent the original image. Alternatively, you can choose a particular set of colors from an image file with this option.
matteFloodfill-Image	const Color &target_, unsigned int matte_, int x_, int y_, PaintMethod method_	Floodfill designated area with a matte value
medianFilterImage	const double radius_ = 0.0	Filter image by replacing each pixel component with the median color in a circular neighborhood
minifyImage	void	Reduce image by integral size
modulateImage	double brightness_, double saturation_, double hue_	Modulate percent hue, saturation, and brightness of an image
negateImage	bool grayscale_ = false	Negate colors in image. Replace every pixel with its complementary color (white becomes black, yellow becomes blue, etc.). Set grayscale to only negate grayscale values in image.
normalizeImage	void	Normalize image (increase contrast by normalizing the pixel values to span the full range of color values).

oilPaintImage	unsigned int radius_ = 3	Oilpaint image (image looks like oil painting)
opacityImage	unsigned int opacity_	Set or attenuate the opacity channel in the image. If the image pixels are opaque then they are set to the specified opacity value, otherwise they are blended with the supplied opacity value. The value of opacity_ ranges from 0 (completely opaque) to MaxRGB. The defines OpaqueOpacity and TransparentOpacity are available to specify completely opaque or completely transparent, respectively.
opaqueImage	const Color &opaqueColor_, const Color &penColor_	Change color of pixels matching opaqueColor_ to specified penColor_.
quantizeImage	bool measureError_ = false	Quantize image (reduce number of colors). Set measureError_ to true in order to calculate error attributes.
raiseImage	const Geometry &geometry_ = "6x6+0+0", bool raisedFlag_ = false	Raise image (lighten or darken the edges of an image to give a 3-D raised or lowered effect)
reduceNoise-Image	void	Reduce noise in image using a noise peak elimination filter.
	unsigned int order_	
rollImage	int columns_, int rows_	Roll image (rolls image vertically and horizontally) by specified number of columnms and rows)
rotateImage	double degrees_	Rotate image counter-clockwise by specified number of degrees
sampleImage	const Geometry &geometry_	Resize image by using pixel sampling algorithm
scaleImage	const Geometry &geometry_	Resize image by using simple ratio algorithm
		Segment (coalesce similar image components) by analyzing the histograms of the color components and

segmentImage	double clusterThreshold_ = 1.0, double smoothingThreshold_ = 1.5	identifying units that are homogeneous with the fuzzy c-means technique. Also uses quantizeColorSpace and verbose image attributes. Specify clusterThreshold_, as the number of pixels each cluster must exceed the cluster threshold to be considered valid. SmoothingThreshold_ eliminates noise in the second derivative of the histogram. As the value is increased, you can expect a smoother second derivative. The default is 1.5.
shadeImage	double azimuth_ = 30, double elevation_ = 30, bool colorShading_ = false	Shade image using distant light source. Specify azimuth_ and elevation_ as the position of the light source. By default, the shading results as a grayscale image.. Set colorShading_ to true to shade the red, green, and blue components of the image.
sharpenImage	const double radius_ = 1, const double sigma_ = 0.5	Sharpen pixels in image. The radius_ parameter specifies the radius of the Gaussian, in pixels, not counting the center pixel. The sigma_ parameter specifies the standard deviation of the Laplacian, in pixels.
shaveImage	const Geometry &geometry_	Shave pixels from image edges.
shearImage	double xShearAngle_, double yShearAngle_	Shear image (create parallelogram by sliding image by X or Y axis). Shearing slides one edge of an image along the X or Y axis, creating a parallelogram. An X direction shear slides an edge along the X axis, while a Y direction shear slides an edge along the Y axis. The amount of the shear is controlled by a shear angle. For X direction shears, x degrees is measured relative to the Y axis, and similarly, for Y direction shears y degrees is measured relative to the X axis. Empty triangles left over from shearing the image

		are filled with the color defined as borderColor.
solarizeImage	double factor_	Solarize image (similar to effect seen when exposing a photographic film to light during the development process)
spreadImage	unsigned int amount_ = 3	Spread pixels randomly within image by specified amount
steganoImage	const Image &watermark_	Add a digital watermark to the image (based on second image)
stereoImage	const Image &rightImage_	Create an image which appears in stereo when viewed with red-blue glasses (Red image on left, blue on right)
swirlImage	double degrees_	Swirl image (image pixels are rotated by degrees)
textureImage	const Image &texture_	Layer a texture on image background
thresholdImage	double threshold_	Threshold image
transformImage	const Geometry &imageGeometry_	Transform image based on image and crop geometries. Crop geometry is optional.
	const Geometry &imageGeometry_, const Geometry &cropGeometry_	
transparentImage	const Color &color_	Add matte image to image, setting pixels matching color to transparent.
trimImage	void	Trim edges that are the background color from the image.
waveImage	double amplitude_ = 25.0, double wavelength_ = 150.0	Alter an image along a sine wave.
zoomImage	const Geometry &geometry_	Zoom image to specified size.

Function objects are available to set attributes on image frames which are equivalent to methods in the Image object. These function objects allow setting an option across a range of image frames using for_each().

The following code is an example of how the color 'red' may be set to transparent in a GIF animation:

```
list<image> images;
readImages( &images, "animation.gif" );
```

```
for_each ( images.begin(), images.end(), transparentImage( "red" )  );
writeImages( images.begin(), images.end(), "animation.gif" );
```

The available function objects for setting image attributes are

Image Image Attributes

Attribute	Type	Constructor Signature(s)	Description
adjoinImage	bool	bool flag_	Join images into a single multi-image file.
antiAliasImage	bool	bool flag_	Control antialiasing of rendered Postscript and Postscript or TrueType fonts. Enabled by default.
animation-DelayImage	unsigned int (0 to 65535)	unsigned int delay_	Time in 1/100ths of a second (0 to 65535) which must expire before displaying the next image in an animated sequence. This option is useful for regulating the animation of a sequence of GIF images within Netscape.
animation-IterationsImage	unsigned int	unsigned int iterations_	Number of iterations to loop an animation (e.g. Netscape loop extension) for.
background-ColorImage	Color	const Color &color_	Image background color
background-TextureImage	std::string	const string &texture_	Image to use as background texture.
borderColor-Image	Color	const Color &color_	Image border color
boxColorImage	Color	const Color &boxColor_	Base color that annotation text is rendered on.
chroma-BluePrimaryImage	float x & y	float x_, float y_	Chromaticity blue primary point (e.g. x=0.15, y=0.06)
chroma-GreenPrimaryImage	float x & y	float x_, float y_	Chromaticity green primary point (e.g. x=0.3, y=0.6)
chroma-RedPrimaryImage	float x & y	float x_, float y_	Chromaticity red primary point (e.g. x=0.64, y=0.33)

chroma-WhitePointImage	float x & y	float x_, float y_	Chromaticity white point (e.g. x=0.3127, y=0.329)
colorFuzzImage	double	double fuzz_	Colors within this distance are considered equal. A number of algorithms search for a target color. By default the color must be exact. Use this option to match colors that are close to the target color in RGB space.
colorMapImage	Color	unsigned int index_, const Color &color_	Color at color-pallet index.
colorSpaceImage	ColorspaceType	ColorspaceType colorSpace_	The colorspace (e.g. CMYK) used to represent the image pixel colors. Image pixels are always stored as RGB(A) except for the case of CMY(K).
compressType-Image	CompressionType	CompressionType compressType_	Image compresion type. The default is the compression type of the specified image file.
densityImage	Geometry (default 72x72)	const Geometry &density_	Vertical and horizontal resolution in pixels of the image. This option specifies an image density when decoding a Postscript or Portable Document page. Often used with psPageSize.
depthImage	unsigned int (8 or 16)	unsigned int depth_	Image depth. Used to specify the bit depth when reading or writing raw images or thwn the output format supports multiple depths. Defaults to the quantum depth that ImageMagick is compiled with.
endianImage	EndianType	EndianType endian_	Specify (or obtain) endian option for formats which support it.
		const	

fileNameImage	std::string	std::string &fileName_	Image file name.
fillColorImage	Color	const Color &fillColor_	Color to use when filling drawn objects
filterTypeImage	FilterTypes	FilterTypes filterType_	Filter to use when resizing image. The reduction filter employed has a sigificant effect on the time required to resize an image and the resulting quality. The default filter is Lanczos which has been shown to produce good results when reducing images.
fontImage	std::string	const std::string &font_	Text rendering font. If the font is a fully qualified X server font name, the font is obtained from an X server. To use a TrueType font, precede the TrueType filename with an @. Otherwise, specify a Postscript font name (e.g. "helvetica").
fontPointsize-Image	unsigned int	unsigned int pointSize_	Text rendering font point size
gifDispose-MethodImage	unsigned int { 0 = Disposal not specified, 1 = Do not dispose of graphic, 3 = Overwrite graphic with background color, 4 = Overwrite graphic with previous graphic. }	unsigned int disposeMethod_	GIF disposal method. This option is used to control how successive frames are rendered (how the preceding frame is disposed of) when creating a GIF animation.
			The type of interlacing scheme (default NoInterlace). This option is used to specify the type of interlacing scheme for raw image formats such as RGB or YUV. NoInterlace means do not interlace, LineInterlace uses

interlace-TypeImage	InterlaceType	InterlaceType interlace_	scanline interlacing, and PlaneInterlace uses plane interlacing. PartitionInterlace is like PlaneInterlace except the different planes are saved to individual files (e.g. image.R, image.G, and image.B). Use LineInterlace or PlaneInterlace to create an interlaced GIF or progressive JPEG image.
isValidImage	bool	bool isValid_	Set image validity. Valid images become empty (inValid) if argument is false.
labelImage	std::string	const std::string &label_	Image label
lineWidthImage	double	double lineWidth_	Line width for drawing lines, circles, ellipses, etc. See Drawable.
magickImage	std::string	const std::string &magick_	Get image format (e.g. "GIF")
matteImage	bool	bool matteFlag_	True if the image has transparency. If set True, store matte channel if the image has one otherwise create an opaque one.
matteColorImage	Color	const Color &matteColor_	Image matte (transparent) color
monochrome-Image	bool	bool flag_	Transform the image to black and white
pageImage	Geometry	const Geometry &pageSize_	Preferred size and location of an image canvas. Use this option to specify the dimensions and position of the Postscript page in dots per inch or a TEXT page in pixels. This option is typically used in concert with density.

			Page may also be used to position a GIF image (such as for a scene in an animation)
penColorImage	Color	const Color &penColor_	Pen color to use when annotating on or drawing on image.
penTextureImage	Image	const Image & penTexture_	Texture image to paint with (similar to penColor).
pixelColorImage	Color	unsigned int x_, unsigned int y_, const Color &color_	Get/set pixel color at location x & y.
psPageSizeImage	Geometry	const Geometry &pageSize_	Postscript page size. Use this option to specify the dimensions of the Postscript page in dots per inch or a TEXT page in pixels. This option is typically used in concert with density.
qualityImage	unsigned int (0 to 100)	unsigned int quality_	JPEG/MIFF/PNG compression level (default 75).
quantize-ColorsImage	unsigned int	unsigned int colors_	Preferred number of colors in the image. The actual number of colors in the image may be less than your request, but never more. Images with less unique colors than specified with this option will have any duplicate or unused colors removed.
quantize-ColorSpaceImage	ColorspaceType	ColorspaceType colorSpace_	Colorspace to quantize colors in (default RGB). Empirical evidence suggests that distances in color spaces such as YUV or YIQ correspond to perceptual color differences more closely than do distances in RGB space. These color spaces may give better results when

			color reducing an image.
quantizeDitherImage	bool	bool flag_	Apply Floyd/Steinberg error diffusion to the image. The basic strategy of dithering is to trade intensity resolution for spatial resolution by averaging the intensities of several neighboring pixels. Images which suffer from severe contouring when reducing colors can be improved with this option. The quantizeColors or monochrome option must be set for this option to take effect.
quantizeTreeDepthImage	unsigned int (0 to 8)	unsigned int treeDepth_	Depth of the quantization color classification tree. Values of 0 or 1 allow selection of the optimal tree depth for the color reduction algorithm. Values between 2 and 8 may be used to manually adjust the tree depth.
renderingIntentImage	RenderingIntent	RenderingIntent render_	The type of rendering intent
resolutionUnitsImage	ResolutionType	ResolutionType units_	Units of image resolution
sceneImage	unsigned int	unsigned int scene_	Image scene number
sizeImage	Geometry	const Geometry &geometry_	Width and height of a raw image (an image which does not support width and height information). Size may also be used to affect the image size read from a multi-resolution format (e.g. Photo CD, JBIG, or JPEG.
strokeColorImage	Color	const Color &strokeColor_	Color to use when drawing object outlines

subImageImage	unsigned int	unsigned int subImage_	Subimage of an image sequence
subRangeImage	unsigned int	unsigned int subRange_	Number of images relative to the base image
tileNameImage	std::string	const std::string &tileName_	Tile name
typeImage	ImageType	ImageType type_	Image storage type.
verboseImage	bool	bool verboseFlag_	Print detailed information about the image
viewImage	std::string	const std::string &view_	FlashPix viewing parameters.
x11DisplayImage	std::string (e.g. "hostname:0.0")	const std::string &display_	X11 display to display to, obtain fonts from, or to capture image from

Query Image Format Support

Magick++ provides the coderInfoList() function to support obtaining information about the image formats supported by ImageMagick. Support for image formats in ImageMagick is provided by modules known as "coders". A user-provided container is updated based on a boolean truth-table match. The truth-table supports matching based on whether ImageMagick can read the format, write the format, or supports multiple frames for the format. A wildcard specifier is supported for any "don't care" field. The data obtained via coderInfoList() may be useful for preparing GUI dialog boxes or for deciding which output format to write based on support within the ImageMagick build.

The definition of coderInfoList is:

```
  class CoderInfo
  {
  public:

    enum MatchType {
      AnyMatch,  // match any coder
      TrueMatch, // match coder if true
      FalseMatch // match coder if false
    };

    [ remaining CoderInfo methods ]

   }

  template <class Container >
  void coderInfoList( Container *container_,
                      CoderInfo::MatchType isReadable_   =
CoderInfo::AnyMatch,
                      CoderInfo::MatchType isWritable_   =
```

```
CoderInfo::AnyMatch,
                     CoderInfo::MatchType isMultiFrame_ =
CoderInfo::AnyMatch
                     );
```

The following example shows how to retrieve a list of all of the coders which support reading images and print the coder attributes (all listed formats will be readable):

```
  list<CoderInfo> coderList;
  coderInfoList( &coderList,           // Reference to output list
                 CoderInfo::TrueMatch, // Match readable formats
                 CoderInfo::AnyMatch,  // Don't care about writable formats
                 CoderInfo::AnyMatch); // Don't care about multi-frame
support
  list<CoderInfo>::iterator entry = coderList.begin();
  while( entry != coderList.end() )
  {
    cout << entry->name() << ": (" << entry->description() << ") : ";
    cout << "Readable = ";
    if ( entry->isReadable() )
      cout << "true";
    else
      cout << "false";
    cout << ", ";
    cout << "Writable = ";
    if ( entry->isWritable() )
      cout << "true";
    else
      cout << "false";
    cout << ", ";
    cout << "Multiframe = ";
    if ( entry->isMultiframe() )
      cout << "true";
    else
      cout << "false";
    cout << endl;
   }
```

22.10 Magick::TypeMetric

The TypeMetric class provides the means to pass data from the Image class's TypeMetric method to the user. It provides information regarding font metrics such as ascent, descent, text width, text height, and maximum horizontal advance. The units of these font metrics are in pixels, and that the metrics are dependent on the current Image font (default Ghostscript's "Helvetica"), pointsize (default 12 points), and x/y resolution (default 72 DPI) settings.

The pixel units may be converted to points (the standard resolution-independent measure used by the typesetting industry) via the following equation:

```
size_points = (size_pixels * 72)/resolution
```

where resolution is in dots-per-inch (DPI). This means that at the default image resolution, there is one pixel per point.

Note that a font's pointsize is only a first-order approximation of the font height (ascender + descender) in points. The relationship between the specified pointsize and the rendered font height is determined by the font designer.

See FreeType Glyph Conventions for a detailed description of font metrics related issues.

The methods available in the TypeMetric class are shown in the following table:

TypeMetric Methods

Method	Returns	Units	Signature	Description
ascent	double	Pixels	void	Returns the distance in pixels from the text baseline to the highest/upper grid coordinate used to place an outline point. Always a positive value.
descent	double	Pixels	void	Returns the the distance in pixels from the baseline to the lowest grid coordinate used to place an outline point. Always a negative value.
textWidth	double	Pixels	void	Returns text width in pixels.
textHeight	double	Pixels	void	Returns text height in pixels.
maxHorizontalAdvance	double	Pixels	void	Returns the maximum horizontal advance (advance from the beginning of a character to the beginning of the next character) in pixels.

22.11 Special Format Characters

The Magick::Image methods annotate, draw, label, and the template function montageImages support special format characters contained in the argument text. These format characters work similar to C's printf. Whenever a format character appears in the text, it is replaced with the equivalent attribute text. The available format characters are shown in the following table.

Format Characters

Format	Description
%b	file size
%c	comment
%d	directory
%e	filename extension
%f	filename
%g	page geometry
%h	height
%i	input filename
%k	number of unique colors
%l	label
%m	magick
%n	number of scenes
%o	output filename
%p	page number
%q	quantum depth
%s	scene number
%t	top of filename
%u	unique temporary filename
%w	width
%x	x resolution
%y	y resolution
%z	image depth
%#	signature
\n	newline
\r	carriage return

23 Perl API Methods

23.1 Image::Magick Attributes

An image has certain attributes associated with it such as width, height, number of colors in the colormap, page geometry, and others. Many of the image methods allow you to set relevant attributes directly in the method call, or you can use Set(), as in:

```
$image->Set(loop=>100);
$image->[$x]->Set(dither=>1);
```

To get an imageattribute, use Get():

```
($width, $height, $depth) = $image->Get('width', 'height', 'depth');
$colors = $image->[2]->Get('colors');
```

The methods GetAttribute() and SetAttribute() are aliases for Get() and Set() and may be used interchangeably.

Following is a list of image attributes acceptable to either Set() or Get() as noted.

adjoin join images into a single multi-image file.

$image->Set(adjoin=>*boolean*)
$image->Get('adjoin')

Certain file formats accept multiple images within a single file (e.g. a GIF animation). If `adjoin` is value other than 0 and the image is a multi-image format, multiple reads to the same image object will join the images into a single file when you call the Write() method. Set `adjoin` to 0 if you do not want the images output to a single file.

antialias remove pixel aliasing.

$image->Set(antialias=>*boolean*)
$image->Get('antialias')

The visible effect of antialias is to blend the edges of any text or graphics with the image background. This attribute affects how text and graphics are rendered when certain image formats are read (e.g. Postscript or SVG) or when certain Image::Magick methods are called (e.g. Annotate() or Draw()).

background image background color.

$image->Set(background=>*color-name*)
$image->Get('background')

This attribute sets (or gets) the background color of an image. Image formats such as GIF, PICT, PNG, and WMF retain the background color information.

base-filename base image filename (before transformations).

$image->Get('base-filename")

The original filename is returned as a string.

base-height base image height (before transformations).

$image->Get('base-height')

This attribute returns the original height of image before any resizing operation.

base-width base image width (before transformations).

$image->Get('base-width')

This attribute returns the original width of image before any resizing operation.

blue-primary chromaticity blue primary point.

$image->Set(blue-primary=>*x-value,y-value*)
$image->Get('blue-primary')

This attribute sets or returns the chromaticity blue primary point. This is a color management option.

cache-threshold cache threshold.

$image->Set(cache-threshold=>*integer*)
$image->Get('cache-threshold')

Image pixels are stored in your computer's memory until it has been consumed or the cache threshold is exceeded. Subsequent pixel operations are cached to disk. Operations to memory are significantly faster, but if your computer does not have a sufficient amount of free memory to read or transform an image, you may need to set this threshold to a small megabyte value (e.g. 32). Use 0 to cache all images to disk.

class image class.

$image->Get('class')

A `Direct` class image is a continuous tone image and is stored as a sequence of red-green-blue and optional opacity intensity values. A `Pseudo` class image is an image with a colormap, where the image is stored as a map of colors and a sequence of indexes into the map.

clip-mask associate a clip mask with the image.

$image->Set('clip-mask'=>*image*)
$image->Get('clip-mask')

`Clip-mask` associates a clip mask with the image.

colormap color of a particular colormap entry.

$image->Set('colormap[*$i*]'=>*color-name*)
$image->Get('colormap[*$i*]')

This attribute returns the red, green, blue, and opacity values at colormap position *$i*. You can set the color with a colorname (e.g. red) or color hex value (e.g. #ccbdbd).

colors number of distinct colors in the image.

$image->Get('colors')

This attribute returns the number of distinct colors in the image.

comment image comment.

$image->Get('comment')

Return the image comment.

compression type of compression.

$image->Set(compression=>*string*)
$image->Get('compression')

`Compression` defaults to the compression type of the image when it was first read. The value of `compression` can be one of the following:

None	BZip	Fax
Group4	JPEG	LosslessJPEG
LZW	RLE	Zip

If you set a compression type that is incompatible with the output file type, a compatible compression value is used instead (e.g. a PNG image ignores a `compression` value of JPEG and saves with Zip compression).

delay interframe delay.

$image->Set(delay=>*integer*)
$image->Get('delay')

`Delay` regulates the playback speed of a sequence of images. The value is the number of hundredths of a second that must pass before displaying the next image. The default is 0 which means there is no delay and the animation will play as fast as possible.

density image resolution.

$image->Set(density=>*geometry*)
$image->Get('density')

This attribute to set the horizontal and vertical resolution of an image. Use attribute `units` to define the units of resolution. The default is 72 dots-per-inch.

depth color component depth.

$image->Get('depth')

Return the color component depth of the image, either 8 or 16. A depth of 8 represents color component values from 0 to 255 while a depth of 16 represents values from 0 to 65535.

directory thumbnail names of an image montage.

$image->Get('directory')

A montage is one or more image thumbnails regularly spaced across a color or textured background created by the Montage() method or *montage* program. `Directory` returns the filenames associated with each thumbnail.

dispose GIF disposal method.

$image->Set(dispose=>*0, 1, 2, 3*)
$image->Get('dispose')

The `dispose` attribute sets the GIF disposal method that defines how an image is refreshed when flipping between scenes in a sequence. The disposal methods are defined as:

0	replace one full-size, non-transparent frame with another
1	any pixels not covered up by the next frame continue to display
2	background color or background tile shows through transparent pixels
3	restore to the state of a previous, undisposed frame

dither apply dithering to the image.

$image->Set(dither=>*boolean*)
$image->Get('dither')

Color reduction is performed implicitly when an image is converted from a file format that allows many colors to one that allows fewer (e.g. JPEG to GIF). Dithering helps smooth out the apparent contours produced when sharply reducing colors. The default is to dither an image during color reduction.

error mean error per pixel.

$image->Get('error')

This value reflects the mean error per pixel introduced when reducing the number of colors in an image either implicitedly or explicitly:

1. Explicitly, when you use the Quantize() method.
2. Implicitly, when an image is converted from a file format that allows many colors to one that allows fewer (e.g. JPEG to GIF).

The mean error gives one measure of how well the color reduction algorithm performed and how similiar the color reduced image is to the original.

file Perl filehandle.

$image->Set(file=>*filehandle*)
$image->Get('file')

The Read() and Write() methods accept an already opened Perl filehandle and the image is read or written directly from or to the specified filehandle.

filename filename of image.

$image->Set(filename=>*string*)
$image->Get('filename')

The default filename is the name of the file from which the image was read. Write() accepts a filename as a parameter, however, if you do not specify one, it uses the name defined by the `filename` attribute. For example:

```
$image->Read('logo.gif');
$image->Write();                    # write image as logo.gif
$image->Set(filename=>'logo.png');
$image->Write();                    # write image as logo.png
```

filesize size of file in bytes.

$image->Get('filesize')

Returns the number of bytes the image consumes in memory or on disk.

font text font.

$image->Set(font=>*string*)
$image->Get('font')

Both Annotate() and Draw() require a font to render text to an image. A font can be Truetype (Arial.ttf), Postscript (Helvetica), or a fully-qualified X11 font (-*-helvetica-medium-r-*-*-12-*-*-*-*-*-iso8859-*) name.

format descriptive image format.

$image->Get('format')

Attribute `magick` returns the abbreviated image format (e.g. JPEG) while `format` returns more descriptive text about the format (e.g. Joint Photographic Experts Group JFIF format).

fuzz close colors are treated as equal.

$image->Set(fuzz=>*integer*)
$image->Get('fuzz')

A number of image methods (e.g. ColorFloodfill()) compare a target color to a color within the image. By default these colors must match exactly. However, in many cases two colors may differ by a small amount. `Fuzz` defines how much tolerance is acceptable to consider two different colors as the same. For example, set `fuzz` to 10 and the color red at intensities of 100 and 102 respectively are now interpreted as the same color.

gamma image gamma.

$image->Set(gamma=>*float*)
$image->Get('gamma')

Set or return the image gamma value. Unlike Gamma() that actually applies the gamma value to the image pixels, here we just set the value. This is useful if the correct gamma is already known about a particular image.

geometry shortcut for specifying width and height.

$image->Set(geometry=>*geometry*)
$image->Get('geometry')

The `geometry` attribute is a convenient way to specify the width, height, and any offset of an image region as a single string. For example,

```
geometry=>'640x80'
```

is equivalent to:

```
width=>640, height=>480
```

To refer to a 20 x 20 region of pixels starting at coordinate (100, 150), use:

```
geometry=>'20x20+100+150'
```

gravity type of gravity.

$image->Set(gravity=>*string*)
$image->Get('gravity')

`Gravity` defaults to NorthWest. The value of `gravity` can be one of the following:

NorthWest	North	NorthEast
West	Center	East
SouthWest	South	SouthEast

green-primary chromaticity green primary point.

$image->Set(green-primary=>*x-value*,*y-value*)
$image->Get('green-primary')

This attribute sets or returns the chromaticity green primary point. This is a color management option.

height image height.

$image->Get('height')

This attribute returns the height (in pixel rows) of the image.

index colormap index at a particular pixel location.

$image->Set('index[*$x, $y*]'=>*color-name*)
$image->Get('index[*$x, $y*]')

This attribute sets or returns the colormap index at position (*$x, $y*). The result is undefined if the image does not have a colormap or the specified location lies outside the the image area.

ICM color information profile.

$image->Get('ICM')

This attribute returns the color information profile.

id ImageMagick registry ID.

$image->Get('id')

This attribute returns the ImageMagick registry ID. The registry allows for persistent images that can later be referenced as a filename (e.g. `registry:0xbd`).

interlace type of interlacing scheme.

$image->Set(interlace=>*string*)
$image->Get('interlace')

The `interlace` attribute allows you to specify the interlacing scheme used by certain image formats such as GIF, JPEG, RGB, and CMYK. The default is `None` but can be any of the following:

None	no interlacing
Line	scanline interlacing
Plane	plane interlacing
Partition	partition interlacing

IPTC newswire information profile.

$image->Get('IPTC')

This attribute returns the newswire information profile.

label image label.

$image->Set(label=>*string*)
$image->Get('label')

Use labels to optionally annotate a Postscript or PDF image or the thumbnail images of a montage created by the Montage() method or *montage* program. A label can include any of the special formatting characters described in the Comment() method description.

loop add loop extension to your image sequence.

$image->Set(loop=>*integer*)
$image->Get('loop')

The `loop` attribute adds the Netscape looping extension to an image sequence. A value of 0 causes the animation sequence to loop continuously. Any other value results in the animation being repeated for the specified number of times. The default value is 1.

magick image file format.

$image->Set(magick=>*string*)
$image->Get('magick')

The default image format is whatever format the image was in when it was read. Write() accepts an image format as a parameter, however, if you do not specify one, it uses the format defined by the `magick` attribute. For example:

```
$image->Read('logo.gif');
$image->Write();                    # write image as GIF
$image->Set(magick=>'PNG');
$image->Write();                    # write image as PNG
```

matte transparency boolean.

$image->Set(matte=>*boolean*)
$image->Get('matte')

Some images have a transparency mask associated with each pixel ranging from opaque (pixel obscures background) to fully transparent (background shows thru). The transparency mask, if it exists, is ignored if the `matte` attribute is 0 and all pixels are treated as opaque.

maximum-error normalized maximum mean error per pixel.

$image->Get('maximum-error')

This value reflects the normalized maximum per pixel introduced when reducing the number of colors in an image either implicitedly or explicitly:

1. Explicitly, when you use the Quantize() method.
2. Implicitly, when an image is converted from a file format that allows many colors to one that allows fewer (e.g. JPEG to GIF).

The normalized maximum error gives one measure of how well the color reduction algorithm performed and how similiar the color reduced image is to the original.

mean-error normalized mean mean error per pixel.

$image->Get('mean-error')

This value reflects the normalized mean per pixel introduced when reducing the number of colors in an image either implicitedly or explicitly:

1. Explicitly, when you use the Quantize() method.
2. Implicitly, when an image is converted from a file format that allows many colors to one that allows fewer (e.g. JPEG to GIF).

The normalized mean error gives one measure of how well the color reduction algorithm performed and how similiar the color reduced image is to the original.

montage tile size and offset within an image montage.

$image->Get('montage')

A montage is one or more image thumbnails regularly spaced across a color or textured background returned by the Montage() method or *montage* program. The `montage` attribute returns the geometry of the region associated with each image thumbnail (e.g. 160x120+10+10). This information is useful for creating image maps for dynamic web pages.

page perferred size and location of the image canvas.

$image->Set(page=>*string*)
$image->Get('page')

`Page` declares the image canvas size and location. Typically this is only useful for the Postscript, text, and GIF formats. The value of `string` can be:

Letter	Tabloid	Ledger
Legal	Statement	Executive
A3	A4	A5
B4	B5	Folio
Quarto	10x14	

or a geometry (612x792). The default value is `Letter`.

pointsize pointsize of a font.

$image->Set(pointsize=>*integer*)
$image->Get('pointsize')

The `pointsize` attribute determines how large to draw a Postscript or TrueType font with the Annotate() or Draw() methods. The default is 12.

preview type of image preview.

$image->Set(preview=>*string*)
$image->Get('preview')

Set or get the type of preview for the Preview image format.

Rotate	Shear	Roll
Hue	Saturation	Brightness
Gamma	Spiff	Dull
Grayscale	Quantize	
Despeckle	ReduceNoise	
AddNoise	Sharpen	Blur
Threshold	EdgeDetect	
Spread	Solarize	Shade
Raise	Segment	Swirl
Implode	Wave	OilPaint
CharcoalDrawing	JPEG	

Suppose we want to determine an ideal gamma setting for our image:

```
$image->Write(filename=>'model.png',preview=>'Gamma');
$image->Display();
```

quality compression level.

$image->Set(quality=>*integer*)
$image->Get('quality')

The quality attribute sets the JPEG, MIFF, MNG, or PNG compression level. The range is 0 (worst) to 100 (best). The default is 75.

Quality is a trade-off between image size and compression speed for the MIFF, MNG, and PNG formats. The higher the quality, the smaller the resulting image size but with a requisite increase in compute time. The quality value is used as two decimal digits. The "tens" digit conveys the zlib compression level and the "ones" digit conveys the PNG filter method. When the compression level is 0, the Huffman compression strategy is used, which is fast but does not necessarily obtain the worst compression. The MIFF encoder ignores the PNG filter method conveyed by the "ones" digit.

The JPEG trade-off is between image size and image appearance. A high quality returns an image nearly free of compression artifacts but with a larger image size. If you can accept a lower quality image appearance, the resulting image size would be considerably less.

red-primary chromaticity red primary point.

$image->Set(red-primary=>*x-value*,*y-value*)
$image->Get('red-primary')

This attribute sets or returns the chomaticity red primary point. This is a color management option.

rendering-intent intended rendering model.

$image->Set(rendering-intent=>*string*)
$image->Get('rendering-intent')

This is a color management option. Choose from these models:

Undefined	Saturation	Perceptual
Absolute	Relative	

sampling-factor image sampling factor.

$image->Set('sampling-factor'=>*geometry*)
$image->Get('sampling-factor')

Use this attribute to set the horizontal and vertical sampling factor for use by the JPEG encoder.

scene image scene number.

$image->Set(scene=>*integer*)
$image->Get('scene')

By default each image in a sequence has a scene number that starts at 0 and each subsequent image in the sequence increments by 1. Use `scene` to reset this value to whatever is appropriate for your needs.

signature SHA-256 message digest.

$image->Get('signature')

Retrieves the SHA-256 message digest associated with the image. A signature is generated across all the image pixels. If a single pixel changes, the signature will change as well. The signature is mostly useful for quickly determining if two images are identical or if an image has been modified.

size width and height of a raw image.

$image->Set(size=>*geometry*)
$image->Get('size')

Set the `size` attribute before reading an image from a raw data file format such as RGB, GRAY, TEXT, or CMYK (e.g. 640x480) or identify a desired resolution for Photo CD images (e.g. 768x512).

```
$image->Set(size=>'640x480');
$image->Read('gray:protein');
```

server X server to contact.

$image->Set(server=>*string*)
$image->Get('server')

Display(), Animate(), or any X11 font use with Annotate() require contact with an X server. Use `server` to specify which X server to contact (e.g. `mysever:0`).

taint pixel change boolean.

$image->Get('taint')

`Taint` returns a value other than 0 if any image pixel has modified since it was first read.

texture name of texture to tile.

$image->Set(texture=>*string*)
$image->Get('texture')

The `texture` attribute assigns a filename of a texture to be tiled onto the image background when any TXT or WMF image formats are read.

type image type.

$image->Set(type=>*string*)
$image->Get('type')

The image type can be any of the following

Bilevel	Grayscale	GrayscaleMatte
Palette	PaletteMatte	TrueColor
TrueColorMatte	ColorSeparation	ColorSeparationMatte
Optimize		

When getting this attribute, the value reflects the type of image pixels. For example a colormapped GIF image would most likely return Palette as the image type. You can also force a particular type with Set(). For example if you want to force your color image to black and white, use:

```
$image->Set(type=>'Bilevel');
```

units units of resolution.

$image->Set(units=>*string*)
$image->Get('units')

Return or set the units in which the image's resolution are defined. Values may be:

Undefined
pixels/inch
pixels/centimeter

verbose print details.

$image->Set(verbose=>*boolean*)

When set, `verbose` causes some image operations to print details about the operation as it progresses.

white-point chromaticity white point.

$image->Set(white-point=>*x-value,y-value*)
$image->Get('white-point')

This attribute sets or returns the chomaticity white point. This is a color management option.

width image width.

$image->Get('width')

Returns the width (integer number of pixel columns) of the image.

x-resolution horizontal resolution.

$image->Get('x-resolution')

Returns the *x* resolution of the image in the units defined by the `units` attribute (e.g. 72 pixels/inch). Use the `density` attribute to change this value.