Convert, Edit, and Compose Images

John Cristy

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

Copyright

Contents

Preface . . . . **vii**

Part 1: Quick Start Guide 1

1 Introduction **3**

1.1 What is ImageMagick 3

1.2 Getting Help 3

2 Image Primer **4**

2.1 What is an Image 4

2.2 Image Depth 4

2.3 Colormapped Images 4

2.4 Compression 4

2.5 Colorspace 4

2.6 Meta-Information 4

2.7 Image Formats 4

3 Image Tools **5**

4 Image Transformations **6**

4.1 How to specify an image 7

4.2 Convert from one Image Format to Another 7

4.3 Colormap Manipulation 7

4.4 Resize an Image 7

4.5 Crop 7

4.6 Enhance 7

4.7 Effects 7
4.8 Decorate 7
4.9 Annotate 7
4.10 Draw 7
4.11 Composite 7
4.12 Meta-Information 7
4.13 Miscellanious Transforms 7

5 Advanced ImageMagick Features 8

5.1 Working with Multi-resolution Images 9
5.2 Working with an Image Sequence 9
5.3 Working with a Group of Images 9
5.4 Working with Raw Images 9
5.5 Using ImageMagick from a Web Browser 9

Part 2: Application Programming Interface 11

6 C Application Programming Interface 13

6.1 Working with Blobs 13
6.2 Working with Threads 13

7 C++ Application Programming Interface 14

7.1 Working with Blobs 14
7.2 Working with Threads 14

8 Perl Application Programming Interface 15

8.1 Background 15

9 PHP Application Programming Interface 16

9.1 Background 16

10 Other Application Programming Interfaces 17

10.1 Java 17
10.2 Python 17

10.3 ImageMagick Integration Project 17

Part 3: User's Guide 19

11 Image Channels 21

11.1 Working with Image Channels 21

12 Image Painting 22

12.1 Image Painting 22

13 Color Profiles 23

13.1 Working with Color Profiles 23

14 Image Drawing 24

14.1 SVG 24
14.2 MVG 24

Part 4: Installation And Administration Guide 25

15 Installing from Binary 27

15.1 Downloading 27
15.2 Linux RPM 27
15.3 Windows 27
15.4 VMS 27
15.5 Unix 27
15.6 Other 27

16 Installing from Source 28

16.1 Downloading 28
16.2 Unix 28
16.3 Windows 28
16.4 Macintosh 28
16.5 VMS 28

17 Customizing ImageMagick **29**

17.1 Image Depth 29

17.2 Image Cache 29

17.3 Delegates 29

17.4 magic.mgk 29

17.5 fontmap.mgk 29

Part 5: Reference Manual **31**

18 Supported Image Formats **33**

19 Commandline options **38**

20 API Structures and Enumerations **70**

20.1 API Structures 70

20.2 API Enumerations 81

21 C API Methods **91**

21.1 Methods to Constitute an Image 91

21.2 ImageMagick Image Methods 93

21.3 Working With Image Lists 103

21.4 Methods to Count the Colors in an Image 107

21.5 Methods to Reduce the Number of Unique Colors in an Image . 109

21.6 Methods to Segment an Image with Thresholding Fuzzy c-Means 112

21.7 Methods to Resize an Image 112

21.8 Methods to Transform an Image 114

21.9 Methods to Shear or Rotate an Image by an Arbitrary Angle . . . 117

21.10 Methods to Enhance an Image 119

21.11 ImageMagick Image Effects Methods 120

21.12 ImageMagick Image Decoration Methods 130

21.13 Methods to Annotate an Image 131

21.14 Methods to Draw on an Image 132

21.15 Methods to Create a Montage 136

21.16 Image Text Attributes Methods 137
21.17 Methods to Compute a Digital Signature for an Image 138
21.18 Methods to Interactively Animate an Image Sequence 139
21.19 Methods to Interactively Display and Edit an Image 139
21.20 Methods to Get or Set Image Pixels 140
21.21 ImageMagick Cache Views Methods 142
21.22 Image Pixel FIFO 144
21.23 Methods to Read or Write Binary Large Objects 144
21.24 ImageMagick Registry Methods 145
21.25 Methods to Read or List ImageMagick Image formats 147
21.26 ImageMagick Error Methods 149
21.27 ImageMagick Memory Allocation Methods 152
21.28 ImageMagick Progress Monitor Methods 153

22 C++ API Methods 154

23 Perl API Methods 155
23.1 Image::Magick Attributes 155
23.2 Image::Magick Methods 170
23.3 Image::Magick Errors 196

24 Recognized Color Keyword Names 198

References 202

A Appendix A 203

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.

4.1 How to specify an image

4.1.1 Implicitly

4.1.2 Explicitly

4.1.3 By URL

4.2 Convert from one Image Format to Another

4.3 Colormap Manipulation

4.4 Resize an Image

4.5 Crop

4.6 Enhance

4.7 Effects

4.7.1 Special Effects

4.7.2 Image Preview

4.8 Decorate

4.9 Annotate

4.10 Draw

4.11 Composite

4.12 Meta-Information

4.12.1 Comment

4.13 Miscellanious Transforms

4.13.1 Append

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 fontmap.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 Hallo
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
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
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
PM	rw-	X Windows system pixmap (color)
PNG	rw-	Portable Network Graphics

Supported Image Formats (continued)

Format	**Mode**	**Description**
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
WBMP	rw-	Wireless Bitmap (level 0) image
WPG	r–	Word Perfect Graphics

Supported Image Formats (continued)

Format	**Mode**	**Description**
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*).

-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 transform matrix

-antialias remove pixel aliasing

-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.

-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 in the “Color Names” section of *X(1)*. 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 in the “Color Names” section of *X(1)*.

-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 in the “Color Names” section of *X(1)*.

-borderwidth <geometry> the border width

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

The color is specified using the format described in the “Color Names” section of *X(1)*.

See **-draw** for further details.

-cache <threshold> megabytes of memory available to the pixel cache

Image pixels are stored in memory until 80 megabytes of memory have been consumed. Subsequent pixel operations are cached on disk. Operations to memory are significantly faster but if your computer does not have a sufficient amount of free memory you may want to adjust this threshold value.

-channel <type> the type of channel

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

Use this option to extract a particular *channel* from the image. **Matte**, for example, is useful for extracting the opacity values from an image.

-charcoal <factor> simulate a charcoal drawing

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

The width and height give the number of columns and rows to remove, and the offsets 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_cockatoo.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 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 delineated 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. You can include the image filename, type, width, height, or other image attribute by embedding special format characters:

```
%b   file size
%c   comment
%d   directory
%e   filename extention
%f   filename
%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
%#   signature
\n   newline
\r   carriage return
```

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 [This option is not used by *convert* but this section is included because it describes the composite operators that are used by the *-draw* option of *convert*.]

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 255 (no overflow). This operation is independent of the matte channels.

Minus
: The result of *composite image* - *image*, with underflow cropped to zero. The matte channel is ignored (set to 255, full coverage).

Add
: The result of *composite image* + *image*, with overflow wrapping around (*mod* 256).

Subtract
: The result of *composite image* - *image*, with underflow wrapping around (*mod* 256). 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 255 (full coverage) for pixels inside the shape, zero outside, and between zero and 255 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 255 (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.

-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.

-crop <***width***>***x***<***height***>{**+-**}<***x offset***>{**+-**}<***y offset***>{***%***} 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 the offsets 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.

To specify a percentage width or height to be removed instead, append **%**. For example to crop the image by ten percent (five percent on each side of the image), use **-crop 10%**.

The x and y 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.

Omit the x and y offset to generate one or more subimages of a uniform size.

-cycle <amount> displace image colormap by amount

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

-debug enable debug printout

-deconstruct break down an image sequence into constituent parts 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

Here are the valid methods:

```
    0     No disposal specified.
    1     Do not dispose between frames.
    2     Overwrite frame with background color from header.
    3     Overwrite with previous frame.
```

-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

```
point           x,y
line            x0,y0 x1,y1
rectangle       x0,y0 x1,y1
roundRectangle  x0,y0  w,h  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
color           x0,y0 method
matte           x0,y0 method
text            x0,y0 string
image           operator x0,y0 w,h filename
```

Point requires a single coordinate. **Line** requires a start and end coordinate, while **rectangle** expects an upper left and lower right coordinate. **roundRectangle** has a center coordinate, a width and height, and the width and height of the corners. **Circle** has a center coordinate and a coordinate for the outer edge. Use **Arc** to circumscribe an 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’
```

See Paths. 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 **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.

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.

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’
```

See **-compose** for a description of the composite operators.

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

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 <order> detect edges within an image

Good order values are odd numbers from 3 to 31.

-emboss emboss an image

-enhance apply a digital filter to enhance a noisy image

-equalize perform histogram equalization to the image

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

The color is specified using the format described in the “Color Names” section of *X(1)*.

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 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 in the “Color Names” section of *X(1)*.

-format <type> the image format type

This option will convert any image to the image format you specify. See **convert(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

Use this option to print information about the image in a format of your choosing. You can include the image filename, type, width, height, or other image attributes by embedding special format characters:

```
%b   file size
%c   comment
%d   directory
%e   filename extention
%f   filename
%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
%#   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.

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

-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**.

You can apply separate gamma values to the red, green, and blue channels of the image with a gamma value list delineated 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 offset>{+-}<y offset>{%}{@}{!}{<}{>} 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 size exceeds the geometry specification. < resizes the image *only* if its dimensions is less than the geometry specification. For example, if you specify `'640x480>'` and the image size is 512x512, the image size does not change. However, if the image is 1024x1024, it is resized to 640x480. 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 offset* 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 offset* is measured between the bottom edges.

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 offset 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, the y offset is measured between the bottom edges. Accordingly, a positive offset will never point in the direction outside of the image.

When used as a *convert*, *import* or *mogrify* option, **-geometry** specifies the size of the output image and 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*.

-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 or other graphic primitive 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.

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_offset**> and <**y_offset**> 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.

-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. 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,

```
-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**.

-level <value> adjust the level of image contrast

Give three point values delineated with commas: black, mid, and white (e.g. 10,1.0,65000). The white and black points range from 0 to MaxRGB and mid ranges from 0 to 10.

-linewidth the line width for subsequent draw operations

-list <type> the type of list

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

This option lists entries from the ImageMagick configuration files.

-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 opacity 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 matte color

The color is specified using the format described in the “Color Names” section of *X(1)*.

-median <order> apply a median filter to the image

Good order values are odd numbers from 3 to 31

-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 sequence

The **-page** option is used 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 <value> 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 **order** 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 in the “Color Names” section of *X(1)*.

See **-fill** for more details.

-page <***width***>***x***<***height***>{**+-**}<***x offset***>{**+-**}<***y offset***>{***%***}{***!***}{<}{>}

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>`).

To position a GIF image, use **-page**{+-}<*x offset*>{+-}<*y offset*> (e.g. -page +100+200).

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**.

-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> specify the pen color for drawing operations

The color is specified using the format described in the “Color Names” section of *X(1)*.

-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 -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 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.

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

For the JPEG image format, quality is 0 (worst) to 100 (best). The default quality is 75.

Quality for the MIFF and PNG image format sets the amount of image compression (quality / 10) and filter-type (quality % 10). Compression quality values range from 0 (worst) to 100 (best). 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 or more, adaptive filtering with *minimum-sum-of-absolute-values* is used.

The default is quality is 75. Which means nearly the best compression with adaptive filtering.

For further information, see the PNG specification.

-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**.

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

Negative 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.

-resize <width>x<height>{+-}<x offset>{+-}<y offset>{%}{@}{!}{<}{>} resize and locate 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 and location 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 and location of the montage, while **-geometry** conveys information about the tiles.

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

See **-geometry** for details the geometry specification. The 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**). See *X(1)* for details.

-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.

-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 **%0Nd, %0No, 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.

-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.

-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

-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 <radius>x<sigma> 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**). See *X(1)* for details.

-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).

-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 in the "Color Names" section of *X(1)*.

See **-draw** for further details.

-strokewidth <value> set the stroke width

See **-draw** for further details.

-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> threshold the image

Create a bi-level image such that any pixel intensity that is equal or exceeds the threshold is reassigned the maximum intensity otherwise the minimum intensity.

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

-tile <geometry> layout of images

-title <string> assign a title to the displayed image

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, or other image attribute by embedding special format characters:

```
%b   file size
%c   comment
%d   directory
%e   filename extention
%f   filename
%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
%#   signature
\n   newline
\r   carriage return
```

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.

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

The color is specified using the format described in the “Color Names” section of *X(1)*.

-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**.

-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> sharpen the image with an unsharp mask operator

Use the given radius and standard deviation (sigma).

-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.

-view <string> FlashPix viewing parameters

-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* to effect the characteristics of the wave.

-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 write the image to a file [*display*]

If *file* 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 *file* 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 with compress or **gzip** respectively. Precede the image file name with — to pipe to a system command. If *file* already exists, you will be prompted as to whether it should be overwritten. 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: Matrix Structure

Matrix 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.

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().

Table20.2: DrawInfo Structure

DrawInfo Structure

Member	**Type**	**Description**
affine	*AffineMatrix*	Coordinate transformation (rotation, scaling, and translation).
border_color	*PixelPacket*	Border color.
box	*PixelPacket*	Text solid background color.
decorate	*DecorationType*	Text decoration type.
density	*char **	Text rendering density in DPI (effects scaling font according to pointsize). E.g. “72x72”.
fill	*PixelPacket*	Object internal fill (within outline) color.
font	*char **	Font to use when rendering text.
gravity	*GravityType*	Text placement preference (e.g. NorthWestGravity).
linewidth	*double*	Stroke (outline) drawing width in pixels.
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.
stroke	*PixelPacket*	Object stroke (outline) color.
stroke_antialias	*unsigned int*	Set to True (non-zero) to obtain anti-aliased stroke rendering.
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.

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

Table20.3: ImageInfo Structure

ExceptionInfo Structure

Member	**Type**	**Description**
severity	*ExceptionType*	warning or error severity.
reason	*char **	warning or error message.
description	*char **	warning or error description.

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:

Table20.4: ImageInfo Structure

ImageInfo 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).
background_color	*PixelPacket*	Image background color
blur	*double*	Blur factor to apply to the image when zooming
border_color	*PixelPacket*	Image border color
chromaticity	*ChromaticityInfo*	Red, green, blue, and white-point chromaticity values.
color_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.
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.
columns	*unsigned long*	Image width
comments	*char **	Image comments
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 int*	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.

ImageInfo Structure (continued)

Member	**Type**	**Description**
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.
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.
fuzz	*int*	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.
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.
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.
list	*struct _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)

ImageInfo Structure (continued)

Member	**Type**	**Description**
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	*unsigned long*	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	*struct _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	*int*	Number of initial bytes to skip over when reading raw image.
page	*RectangleInfo*	size of Postscript page and offsets. Offsets are measured from the lower left corner of the page, regardless of their sign.
pipe	*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	*struct _Image* *	Previous image frame in sequence.
rendering_intent	*RenderingIntent*	The type of rendering intent.
rows	*unsigned long*	Image height
scene	*unsigned long*	Image frame scene number.
tainted	*int*	Set to non-zero (True) if the image pixels have been modified.
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.
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().

ImageInfo Structure (continued)

Member	**Type**	**Description**
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:

Table20.5: ImageAttribute Structure

ImageAttribute Structure

Member	**Type**	**Description**
key	*char **	key.
value	*char **	value.
compression	*unsigned int*	compression.

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 structure. 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:

Table20.6: ImageInfo Structure

ImageInfo Structure

Member	**Type**	**Description**
adjoin	*unsigned int*	Join images into a single multi-image file.
antialias	*unsigned int*	Control antialiasing of rendered graphic primitives and text fonts.
background_color	*PixelPacket*	Image background color.
border_color	*PixelPacket*	Image border color.
box	*char **	Base color that annotation text is rendered on.
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.
delay	*char **	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.
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).
dispose	*char **	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.
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.
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.
fill	*PixelPacket*	Drawing object fill color.

ImageInfo Structure (continued)

Member	**Type**	**Description**
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	*int*	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.
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.
iterations	*char **	Number of iterations to loop an animation (e.g. Netscape loop extension) for.
linewidth	*unsigned long*	Line width for drawing lines, circles, ellipses, etc.
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.
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/PNG compression level (default 75).
server_name	*char **	X11 display to display to obtain fonts from, or to capture image from.
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.
stroke	*PixelPacket*	Drawing object outline color.
subimage	*unsigned long*	Subimage of an image sequence.
subrange	*unsigned long*	Number of images relative to the base image.
texture	*char **	Image filename to use as background texture.

ImageInfo Structure (continued)

Member	**Type**	**Description**
tile	*char **	Tile name.
units	*ResolutionType*	Units of image resolution.
verbose	*unsigned int*	Print detailed information about the image if True.
view	*char **	FlashPix viewing parameters.

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 run-time, 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.

Table20.7: MagickInfo Structure

MagickInfo Structure

Member	**Type**	**Description**
tag	*const char **	Magick string (e.g. “GIF”) to call this format.
decoder	*Image **	*(*decoder)(const ImageInfo *)* Function to decode image data and return ImageMagick Image.
encoder	*unsigned int*	*(*encoder)(const ImageInfo, Image *)* Function to encode image data with options passed via ImageInfo and image represented by Image.
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).
raw	*unsigned int*	Image format does not contain size (must be specified in ImageInfo).

MagickInfo Structure (continued)

Member	**Type**	**Description**
description	*char **	Long form image format description (e.g. “CompuServe graphics interchange format”).
module	*char **	Name of module (e.g. “GIF”) which registered this format. Set to NULL if format is not registered by a module.
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.
previous	*MagickInfo*	Previous MagickInfo struct in linked-list. NULL if none.
next	*MagickInfo*	Next MagickInfo struct in linked-list. NULL if none.

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:

Table20.8: PixelPacket Structure

PixelPacket Structure

Member	**Type**	**Description**
red	*Quantum*	red.
green	*Quantum*	green.
blue	*Quantum*	blue.
opacity	*Quantum*	opacity.

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:

Table20.9: ProfileInfo Structure

ProfileInfo Structure

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

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

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

Table20.10: RectangleInfo Structure

RectangleInfo Structure

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

20.2 API Enumerations

CacheType

ChannelType ChannelType is used as an argument when doing color separations. Use ChannelType when extracting a channel from an image. MatteChannel is useful for extracting the opacity values from an image.

Table20.11: ChannelType Enumeration

ChannelType Enumeration

Enumeration	**Description**
UndefinedChannel	Unset value.
RedChannel	Select red channel.
GreenChannel	Select green channel.
BlueChannel	Select blue channel.
MatteChannel	Select matte (opacity values) channel.

ClassType ClassType specifies the image storage class.

Table20.12: 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.

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.

Table20.13: 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	
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	

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.

Table20.14: 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.

CompositeOperator Enumeration (continued)

Enumeration	**Description**
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 255 (no overflow). This operation is independent of the matte channels.
MinusCompositeOp	The result of composite image - image, with overflow cropped to zero. The matte chanel is ignored (set to 255, full coverage).
AddCompositeOp	The result of composite image + image, with overflow wrapping around (mod 256).
SubtractCompositeOp	The result of composite image - image, with underflow wrapping around (mod 256). 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.
BumpmapCompositeOp	The result image shaded by composite image.
ReplaceCompositeOp	The resulting image is image replaced with composite image. Here the matte information is ignored.
ReplaceRedCompositeOp	The resulting image is the red channel in image replaced with the red channel in composite image. The other channels are copied untouched.
ReplaceGreenCompositeOp	The resulting image is the green channel in image replaced with the green channel in composite image. The other channels are copied untouched.
ReplaceBlueCompositeOp	The resulting image is the blue channel in image replaced with the blue channel in composite image. The other channels are copied untouched.

CompositeOperator Enumeration (continued)

Enumeration	Description
ReplaceMatteCompositeOp	The resulting image is the matte channel in image replaced with the matte channel in composite image. The other channels 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 255 (full coverage) for pixels inside the shape, zero outside, and between zero and 255 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 255 (to work properly borderWidth must be 0).

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.

Table20.15: 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

ExceptionType

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.

Table20.16: 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

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.

Table20.17: 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.
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.

Table20.18: 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.

Table20.19: InterlaceType Enumeration

InterlaceType Enumeration

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

LineCap

LineJoin

MapMode

MontageMode

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

Table20.20: 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.

Table20.21: 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.

ProfileType

PreviewType

PrimitiveType

PrimitiveType

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.”

Table20.22: RenderingIntent Enumeration

RenderingIntent Enumeration

Enumeration	**Description**
UndefinedIntent	Unset value.
SaturationIntent	A rendering intent that specifies 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 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.

Table20.23: 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).

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. 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 Write the image defined by the `file` or `filename` members of this structure.
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	InCompositeOP	OutCompositeOP
AtopCompositeOP	XorCompositeOP	PlusCompositeOP
MinusCompositeOP	AddCompositeOP	SubtractCompositeOP
DifferenceCompositeOP	BumpmapCompositeOP	CopyCompositeOP
DisplaceCompositeOP		

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.

DestroyImages() destroy an image sequence.

```
void DestroyImages(Image image)
```

DestroyImages() is a convenience method. It calls DestroyImage() for each image in the sequence.

A description of each parameter follows:

image The image sequence.

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, pixels 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.

A small normalized mean square error, accessed as `image->normalized_mean_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 oapcity 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

CloneImageList() duplicate an image list.

```
Image CloneImageList(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.

DeleteImageList() delete an image from the list.

```
unsigned int DeleteImageList(Image images, const unsigned long offset)
```

DeleteImageList() 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.

GetImageList() get an image from an image list.

> Image GetImageList(Image images, const unsigned long offset, ExceptionInfo exception)

GetImageList() returns an image at the specified position in the image 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.

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

> size_t GetImageListSize(const Image images)

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

A description of each parameter follows:

images The image list.

GetNextImage() the next image in a sequence.

> Image GetNextImage(Image imageis)

GetNextImage() returns the next image in an image sequence.

A description of each parameter follows:

images The image list.

ImageListToGroup() convert an image list to an array.

> Image ListToGroupImage(const Image image, ExceptionInfo exception)

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

```
group = ListToGroupImage(images, exception);
for (i=0; i < n; i++)
  puts(group[i]->filename);
```

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.

PopImageList() remove the first image from an image list.

    Image PopImageList(Image *images)

PopImageList() removes the first image in the list.

A description of each parameter follows:

images The image list.

PushImageList() add an image to the end of an image list.

    unsigned int PushImageList(Image images, const Image image, Exception-
          Info exception)

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

A description of each parameter follows:

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

ReverseImageList() reverse an image list.

    Image CloneImageList(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.

SetImageList() add an image to the end of an image list.

> unsigned int SetImageList(Image images, const Image image, ExceptionInfo exception)

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

A description of each parameter follows:

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

ShiftImageList() reverse an image list.

> Image CloneImageList(Image images, ExceptionInfo exception)

ShiftImageList() 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.

SpliceImageList() splice an image list.

> Image SpliceImageList(Image images, const unsigned long offset, const unsigned long length, const Image splices, ExceptionInfo exception)

SplicemageList() removes the images designated by offset and length from the list and replaces them with the specified list.

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 eplace the removed image list with this list.
exception Return any errors or warnings in this structure.

UnshiftImageList() reverse an image list.

> Image CloneImageList(Image images, ExceptionInfo exception)

UnshiftImageList() 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.

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 any 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)

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.

QueryColorname() return a color name for the corresponding numerical values.

unsigned int QueryColorname(const Image *image, const PixelPacket $\times$ 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.

QuantizationError()

unsigned int QuantizationError(Image image)

Method QuantizationError 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 Thsi 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 $\times$ 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 $\times$ 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 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.

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, ExceptionInfo 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 offset `x` is negative, it specifies the distance from the right edge of the region to the right edge of the chopping region. Similarly, if the offset `y` is negative, the distance is 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 with the next image in the sequence.

Regardless of their signs, offsets are measured from the lower left corner of the composition to the lower left corner of each image. Positive offsets represent a location of the image to the right and upward 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 offset `x` is negative, it specifies the distance from the right edge of the region to the right edge of the chopping region. Similarly, if the offset `y` is negative, the distance is between the bottom edges.
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.

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 point values delineated with commas: black, mid, and white (e.g. 10/1.0/65000). The white and black points range from 0 to MaxRGB and mid ranges from 0 to 10.

A description of each parameter follows:

image The image.
gamma Define the image contrast levels.

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, ExceptionInfo $\times$
    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. texttOffset 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 $\times$ 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 operatorof 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 in pixels needed to apply the diffence 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 $\times$ 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.

StoreImageAttribute() store an image attribute.

```
StoreImageAttribute(Image image, char text)
```

StoreImageAttribute() is used to store an image attribute from a text string with the syntax: NAME=VALUE.

A description of each parameter follows:

image The image.
text The text string that is parsed and used to determine the name and value of the new attribute.

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, ExceptionInfo *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.

```
void MagickMonitor(const char text, const off_t quantum, const off_t span)
```

MagickMonitor() calls the monitor handler method with a text string that describes the task and a measure of completion.

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.

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

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.

clip-mask associate a clip mask with the image.

> $image->Set('clip-mask'=>*image*)

`Clip-mask` associates a clip mask with the image.

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.

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')

Set or 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 vertical and horizontal 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('filesize')

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’
```

compression type of gravity.

$image->Set(gravity=>*string*)
$image->Get(’compression’)

`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(label=>*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('magick')

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, 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 and PNG formats. The higher the quality, the smaller the resulting image size but with a requisite increase in compute time. 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	

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(units=>*boolean*)
$image->Get('units')

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.

y-resolution vertical resolution.

$image->Get('y-resolution')

Returns the *y* 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.

23.2 Image::Magick Methods

AddNoise() add noise to an image.

$image->AddNoise(noise=>*string*)

This method adds random noise to the image, where *string* specifies one of the following types:

Uniform	Gaussian	Multiplicative
Impulse	Laplacian	Poisson

Animate() animate an image sequence.

$image->Animate()

Animate() repeatedly displays an image sequence to any X window screen. This method accepts the same parameters as Set() as described in section 23.1.

Annotate() annotate an image with text.

> $image->Annotate(text=>*string*, antialias=>*boolean*, box=>*color-name*, density=>*geometry*, fill=>*color-name*, family=>*string*, font=>*string*, geometry=>*geometry*, gravity=>*string*, pointsize=>*integer*, rotate=>*rotate-angle*, scale=>*sx, sy*, skewX=>*skew-angle*, skewY=>*skew-angle*, stroke=>*color-name*, strokewidth=>*integer*, stretch=>*string*, style=>*string*, translate=>*tx, ty*, weight=>*string*, x=>*integer*, y=>*integer*)

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. You can affect how text is drawn by specifying one or more of the following parameters:

antialias The visible effect of antialias is to smooth out the rounded corners of text characters. Set to 0 to keep crisp edges.
box By default text is blended with the image background. Set the box color to give a uniform background to your text of the color you choose.
density Set the vertical and horizontal resolution of the font. The default is 72 pixels/inch.
family font family.
fill The fill color paints any areas inside the outline of the text.
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 text is rendered (e.g. +100+50).
gravity Gravity affects how the text is rendered relative to the (*x*, *y*) baseline position. By default gravity is NorthWest which renders text above the baseline position. Choose from these gravities:

NorthWest	North	NorthEast
West	Center	East
SouthWest	South	SouthEast

pointsize The font pointsize. The default is 12.
rotate Specifies a rotation by the specified number of 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.
stretch font stretch. Choose from these stretches:

Normal	UltraCondensed	ExtraCondensed
Condensed	SemiCondensed	SemiExpanded
Expanded	ExtraExpanded	UltraExpanded

stroke A stroke color paints along the outline of the text.

strokewidth The width of the stroke on the text. A zero value causes no stroke to be painted.

style font style. Choose from these styles:

Normal	Italic	Oblique
Any		

translate Specifies a translation by *tx* and *ty*.

x Specifies the *x* baseline position of the text.

y Specifies the *y* baseline position of the text.

Append() append a set of images.

$image->Append(stack=>*boolean*)

The Append() method takes a set of images and appends them to each other. Append() returns a single image where each image in the original set is side-by-side. If the stack parameter is True, the images are stacked top-to-bottom.

```
$append = $image->Append();
```

Average() average a set of images.

$image->Average()

The Average() method takes a set of images and averages them together. Each image in the set must have the same width and the same height. Average() returns a single image with each corresponding pixel component of each image averaged.

BlobToImage() return an image from a Binary Large OBject.

$image->BlobToImage(*blob*)

Read() returns an image from a file on disk, whereas, BlobToImage() performs the same function if the image format is stored in memory:

```
@blob = $db->GetImage();    # get blob from database
$image = Image::Magick->New(magick=>’jpg’);
                            # the blob is a JPEG image
$image->BlobToImage(@blob); # convert blob to Image::Magick object
```

Blur() blur the image.

$image->Blur(geometry=>*geometry*, radius=>*float*, sigma=>*float*)

Blur() 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 Blur() selects a suitable radius for you. `Geometry` represents radius x sigma as one parameter (e.g. 0x1).

Border() frame the image with a border.

$image->Border(geometry=>*geometry*, width=>*integer*, height=>*integer*, fill=>*color-name*)

This method surrounds the image with a border of the specified color. `Geometry` represents *width x height* as one parameter (e.g. 10x5).

Channel() extract a channel from the image.

$image->Channel(channel=>*string*);

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

Red
Cyan
Green
Magenta
Blue
Yellow
Opacity
Black

Charcoal() special effect filter.

$image->Charcoal(geometry=>*geometry*, radius=>*float*, sigma=>*float*)

Charcoal() is a special effect filter that simulates a charcoal drawing. 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 Charcoal() selects a suitable radius for you. `Geometry` represents radius x sigma as one parameter (e.g. 0x1).

Chop() chop an image.

$image->Chop(geometry=>geometry, width=>integer, height=>integer, x=>integer, y=>integer)

Chop() removes a region of an image and collapses the image to occupy the removed portion. Columns `x` through `x+width` and the rows `y` through `y+height` are chopped. Use `Geometry` as a shortcut for *width x height + x + y* (e.g. 100x50+10+20).

Clone() create a new copy of an image.

$image->Clone()

The Clone() method copies a set of images and returns the copy as a new image object. For example

```
$clone = $image=>Clone();
```

copies all of the images from `$image` to `$clone`.

Coalesce() coalesce a set of images.

$image->Coalesce()

This method 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 with the next image in the sequence.

ColorFloodfill() floodfill the designed area with color.

$image->ColorFloodfill(geometry=>*geometry*, x=>*integer*, y=>*integer*, fill=>*color-name*, bordercolor=>*color-name*, fuzz=>*float*)

ColorFloodfill() 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 `bordercolor`. Use `Geometry` as a shortcut for *x + y* (e.g. +10+20).

By default `fill` 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 for the purposes of the floodfill.

Colorize() colorize an image.

$image->Colorize(fill=>*color-name*, opacity=>*string*)

Colorize() 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).

Comment() add a comment to an image.

$image->Comment(comment=>*string*)

Add a comment to an image. Optionally you can include any of the following bits of information about the image by embedding the appropriate 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 the 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.
%#	SHA-256 message digest.

Given an image whose filename is `logo.gif` and dimensions of 640 pixels in width and 480 pixels in height, this statement:

```
$image->Comment('%f %m %wx%h')
```

generates a comment that reads: `logo.gif GIF 640x480`.

Composite composite one image to another.

$image->Composite(image=>*image-handle*, compose=>*string*, geometry=>*geometry*, x=>*integer*, y=>*integer*, gravity=>*string*, opacity=>=*integer*, rotate=>*float*, tile=>*image-handle*)

Composite() allows you to overlay one image to another. You can affect how and where the composite is overlaid by specifying one or more of the following options:

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

Over	In	Out
Atop	Xor	Plus
Minus	Add	Subtract
Difference	Bumpmap	Copy
Displace		

geometry Geometry defines the baseline position where the composite is placed (e.g. +100+50).
x Specifies the *x* baseline position of the composite.
y Specifies the *y* baseline position of the composite.
gravity Gravity affects how the image is placed relative to the (*x, y*) baseline position. By default gravity is NorthWest which renders the image just below the baseline position. Choose from these gravities:

NorthWest	North	NorthEast
West	Center	East
SouthWest	South	SouthEast

opacity Blend composite with the image background. `Opacity` is expressed as percent transparency.
rotate Rotate image before it is composited, expressed in degrees.
tile A value other than 0 tiles the composite repeatedly across and down the image.

Contrast() enhance or reduce the image contrast.

$image->Contrast(sharpen=>*boolean*)

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.

Convolve() apply a convolution kernel to the image.

$image->Convolve(coefficients=>*array of float values*)

Apply a custom convolution kernel to the image. Given a particular kernel *order*, you must supply *order x order* float values. For example, a kernel of order 3 implies 9 values (3x3):

```
$image->Convolve([1, 2, 1, 2, 4, 2, 1, 2, 1]);
```

Crop crop an image.

$image->Crop(geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*)

Crop() extracts a region of the image starting at the offset defined by `x` and `y` and extending for `width` and `height`. `Geometry` is a shorthard method to define a region. To crop 100 x 50 region that begins at position (10, 20), use

```
$image->Crop('100x50+10+20');
```

CycleColormap displace a colormap.

$image->CycleColormap(display=>*integer*)

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.

Deconstruct return the constituent parts of an image sequence.

$image->Deconstruct()

Deconstruct() returns a new sequence that consists of the first image in the sequence followed by the maximum bounding region of any differences in subsequent images. This method can undo a coalesced sequence returned by Coalesce().

Despeckle filter speckles.

$image->Despeckle()

Despeckle() reduces the *speckle* noise in an image while perserving the edges of the original image.

Display() display image.

> $image->Display(server=>*server-name*)

Display() displays the image to any X window screen.

Draw annotate an image with a graphic primitive.

> $image->Draw(primitive=>*string*, antialias=>*boolean*, bordercolor=>*color-name*, density=>*geometry*, fill=>*color-name*, font=>*string*, geometry=>*geometry*, method=>*string*, points=>*string*, pointsize=>*integer*, rotate=>*rotate-angle*, scale=>*sx, sy*, skewX=>*skew-angle*, skewY=>*skew-angle* stroke=>*color-name*, strokewidth=>*integer*, translate=>*tx, ty*

Draw() allows you 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 specifying one or more of the following parameters:

primitive The primitive describes the type of graphic to draw. Choose from these primitives:

Point	Line	Rectangle
roundRectangle	Arc	Ellipse
Circle	Polyline	Polygon
Bezier	Path	Color
Matte	Text	Image

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.
strokewidth The width of the stroke of the shape. A zero value means no stroke is painted.
translate Specifies a translation by *tx* and *ty*.

Edge detect edges within an image.

$image->Edge(radius=>*float*)

Edge() 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.

Emboss emboss the image.

`$image->Emboss(geometry=>`*geometry*`, radius=>`*float*`, sigma=>`*float*`)`

Emboss() 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. `Geometry` represents radius x sigma as one parameter (e.g. 0x1).

Enhance filter a noisy image.

$image->Enhance()

Enhance() applies a digital filter that improves the quality of a noisy image.

Equalize equalize an image.

$image->Equalize()

Perform a histogram equalization on the image.

Flatten() flatten a sequence of images.

$image->Coalesce()

This method composites a sequence of images while respecting any page offsets. A Photoshop image typically starts with an image background and each subsequent layer varies in size and offset. Flatten() returns a single image with all the layers composited onto the first image in the sequence.

Flip reflect an image vertically.

$image->Flip()

Flip() creates a vertical mirror image by reflecting the pixels around the central x-axis.

Flop reflect an image horizontally.

$image->Flop()

Flop() creates a horizontal mirror image by reflecting the pixels around the central y-axis.

Frame surround the image with a decorative border.

$image->Frame(geometry=>*geometry*, width=>*integer*, height=>*integer*,
inner=>=*integer*, outer=>*integer*, fill=>*color-name*)

Frame() adds a simulated three-dimensional border around the image. The color of the border is defined by `fill`. `Width` and `height` specify the border width of the vertical and horizontal sides of the frame. The `inner` and `outer` parameters indicate the width of the inner and outer *shadows* of the frame. Use `Geometry` as a shortcut for `width`, `height`, `inner`, and `outer` (e.g. 10x10+3+3).

Gamma gamma-correct the image.

$image->Gamma(gamma=>*string*, red=>*float*, green=>*float*, blue=>*float*)

Use Gamma() 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.

Get() get an image attribute.

$image->Get(*attribute*, ...)

Get() accepts one or more image attributes listed in section 23.1 and return their value.

ImageToBlob() return image as a Perl variable.

$image->ImageToBlob()

ImageToBlob() behaves just like Write() except the image is returned as a Perl variable rather than written to disk. This method accepts the same parameters as Set() as described in section 23.1.

Implode() apply an implosion/explosion filter.

$image->Implode(amount=>*double*)

Implode() applies a special effects filter to the image where `amount` determines the amount of implosion. Use a negative amount for an explosive effect.

Label() add a label to an image.

$image->Label(label=>*string*)

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.

Level adjust the level of image contrast.

$image->Level(levels=>*string*, 'black-point'=>*float*, 'mid-point'=>*float*, 'white-point'=>*float*)

The white and black points range from 0 to MaxRGB and mid ranges from 0 to 10.

Magnify() scale the image to twice its size.

$image->Magnify()

Magnify() is a convenience method that scales an image proportionally to twice its size.

Map() choose a set of colors from another image.

$image->Map(image=>*image-handle*, dither=>*boolean*)

Map() changes the colormap of the image to that of the image given by `image`. Use this method to change the colormap in an image or image sequence to a set of predetermined colors. Set `dither` to a value other than zero to helps smooth out the apparent contours produced when sharply reducing colors.

One useful example of mapping is to convert an image to the Netscape 216-color web safe palette:

```
$safe = new Image::Magick;
$safe->Read('Netscape:');
$image->Map(image=>$safe, dither=>'True');
```

MatteFloodfill() floodfill an area with transparency.

$image->MatteFloodfill(geometry=>*geometry*, x=>*integer*, y=>*integer*, opacity=>*integer*, bordercolor=>*color-name*, fuzz=>*float*)

MatteFloodfill() changes the transparency value of any pixel that matches `opacity` and is an immediate neighbor. If `bordercolor` is specified, the transparency value is changed for any neighbor pixel that is not `bordercolor`. Use `Geometry` as a shortcut for *x* + *y* (e.g. +10+20).

By default `opacity` must match a particular pixel transparency exactly. However, in many cases two transparency values may differ by a small amount. `Fuzz` 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.

MedianFilter() filter a noisy image.

$image->MedianFilter(radius=>*float*)

MedianFilter() 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`.

Minify() scale the image to half its size.

$image->Magnify()

Minify() is a convenience method that scales an image proportionally to half its size.

Modulate adjust the brightness, saturation, and hue.

$image->Modulate(factor=>*string*, brightness=>*float*, saturation=>*float*, hue=>*float*)

Modulate() lets you control the brightness, saturation, and hue of an image. Each parameter is in the form of a percentage relative to 100. For example, to decrease the brightness by 10

```
$image->Modulate(brightness=$>$90, saturation=$>$150);
```

`Factor` represents the brightness, saturation, and hue as one parameter (e.g. 90/150/100).

Mogrify() alternative calling scheme.

$image->Mogrify(method, ...)

The Mogrify() method is convenience function that allows you to call any image manipulation method by giving a method name followed by one or parameters to pass to the method. The following calls have the same result:

```
$image->Crop('340x256+0+0')
$image->Mogrify('Crop', '340x256+0+0')
```

MogrifyRegion() apply method to a region.

$image->MogrifyRegion(geometry, method, ...)

MogrifyRegion() applies an image manipulation method to a region of the image as defined by *geometry*. For example if you want to sharpen a 100 x 100 region starting at position (20, 20), use: result:

```
$image->MogrifyRegion('100x100+20+20', Sharpen, '0x1')
```

Montage() uniformly tile thumbnails across an image canvas.

$image->Montage(background=>*color-name*, bordercolor=>*color-name*, borderwidth=>*integer*, compose=>*string*, fill=>*color-name*, font=>*string*, frame=>*geometry*, geometry=>*geometry*, gravity=>*string*, label=>*string*, mattecolor=>*color-name*, mode=>*string*, pointsize=>*integer*, shadow=>*boolean*, stroke=>*color-name*, texture=>*string*, tile=>*geometry*, title=>*string*, transparent=>*color-name*)

The Montage() method is a layout manager that lets you tile one or more thumbnails across an image canvas. Use these parameters to control how the layout manager places the thumbnails:

background The color name for the montage background.
bordercolor The color name for the thumbnail border.
borderwidth The width of the thumbnail border.
compose This operator affects how the thumbnail is composited on the image canvas. The default is Over. Choose from these operators:

Over,	In	Out
Atop	Xor	Plus
Minus	Add	Subtract
Difference	Bumpmap	Copy
Displace		

fill The fill color paints any areas inside the outline of the thumbnail label.
font A font can be a Truetype (arial.ttf), Postscript (Helvetica), or a fully-qualified X11 font (-*-helvetica-medium-r-*-*-12-*-*-*-*-*-iso8859-*).
frame Adds a simulated three-dimensional border around each thumbnail. The color of the border is defined by `mattecolor`. Specify the border width of the vertical and horizontal sides of the frame and the inner and outer *shadows* of the frame as a geometry (e.g. 10x10+3+3).
geometry Geometry defines the baseline position where a thumbnail is composited (e.g. +100+50).

gravity Gravity affects how the thumbnail is placed relative to the (x, y) baseline position. By default gravity is South which positions the thumbnail centered south of the baseline position. Choose from these gravities:

NorthWest	North	NorthEast
West	Center	East
SouthWest	South	SouthEast

label A label optionally appears just below each thumbnail. Use this parameter to customize the label. See Comment() for a list of embedded formatting options for the thumbnail label.

mode Define one of three thumbnail framing options:

Frame	Unframe	Concatentate

The default is `Frame` which adds a simulated three-dimensional border around each thumbnail. `Unframe` tiles thumbnails without any border or frame, and `Concatentate` causes each image to be tightly packed without any border, frame, or space between them.

pointsize The font pointsize. The default is 12.

shadow Any value other than 0 will add a simulated shadow beneath and to the right side of each thumbnail.

stroke The stroke color paints along the outline of any text labels.

texture Tile this image across and down the image canvas before compositing the image thumbnails.

tile Give the number of thumbnails across and down the canvas as a geometry string. The default is 5 x 4. If the number of thumbnails exceed this maximum, more then one image canvas is created.

title Give a title to the montage. The title is centered near the top of the montage image.

transparent Make this color transparent.

Mosaic() form a single coherent picture.

$image->Mosiac()

The Mosaic() method takes a set of images and inlays them to form a single coherent pictiure. Mosaic() returns a single image with each image in the sequence inlayed in the image canvas at an offset as defined in the image.

```
$mosaic = $image->Mosaic();
```

MotionBlur() simulate motion blur.

$image->MotionBlur(geometry=>*geometry*, radius=>*float*, sigma=>*float*,
angle=>*float*)

MotionBlur() 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 MotionBlur() selects a suitable radius for you. `Geometry` represents radius x sigma as one parameter (e.g. 0x1). `Angle` gives the angle of the blurring motion.

Morph() morph a set of images.

$image->Morph(frames=>*integer*)

The Morph() 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`. The result is returned as a new image sequence, for example:

```
$morph = $image->Morph(30);
```

Negate apply color inversion.

$image->Negate(gray=>*boolean*)

Negate() negates the intensities of each pixel in the image. If `gray` is a value other than 0, only the grayscale pixels are inverted.

New() create an image object.

$image = new Image::Magick;
$image = Image::Magick-¿New()

New() instantiates an image object. As a convenience, you can set any image attribute that Set() knows about. See section **??** for a list of known image attributes. Here is an example:

```
$image = Image::Magick->New(size=>'160x120');
$image->Read('gray:protein');
```

Normalize() enhance image contrast.

$image->Normalize()

The Normalize() method enhances the contrast of a color image by adjusting the pixels color to span the entire range of colors available.

OilPaint() simulate an oil painting.

$image->OilPaint(radius=>*integer*)

OilPaint() 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`.

Opaque() globally change a color.

$image->Opaque(color=>*color-name*, fill=>*color-name*, fuzz=>*float*)

Opaque() 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.

OrderedDither() reduce the image to black and white.

$image->OrderedDither()

The OrderedDither() method reduces the image to black and white.

Ping() get information about an image.

$image->Ping(filename=>*string*, file=>*file-handle*, blob=>*blob*)

Ping() is a convenience method that returns information about an image without having to read the image into memory. It returns the width, height, file size in bytes, and the file format of the image. You can specify more than one filename but only one filehandle:

```
($width, $height, $size, $format) = $image->Ping('logo.gif');
($width, $height, $size, $format) = $image->Ping(file=>\*IMAGE);
($width, $height, $size, $format) = $image->Ping(blob=>\$blob);
```

Profile() add, remove, or apply an image profile.

$image->Profile(name=>*variable*, profile=>*blob*)

The Profile() method adds, removes, or applies an image profile. The two most common profiles are ICC, a color management option, IPTC, a newswire profile, and APP1, which is a JPEG marker that can contain EXIF data. `Profile` is a Perl variable representing the binary profile information. To remove all profiles from the image, use an asterick as the profile name:

```
$image->Profile('*');
```

Quantize() set the maximum number of colors in an image.

$image->Quantize(colors=>*integer*, colorspace=>*string*, dither=>*boolean*, global_colormap=>*boolean* measure_error=>*boolean*, tree_depth=>*integer*)

The Quantize() method sets the maximum number of colors in an image. If the number of colors in the image exceeds `colors`, a color reduction algorithm repeatly merges pixels of similar color until the total number of unique colors is less or equal to the maximum. Here is a description of the color reduction parameters:

colors Set the maximum number of colors in the image.

colorspace By default, color merging is performed in the RGB colorspace. However, RGB is not perceptually uniform like YCbCr for example. You may get better results by trying one of the following colorspaces:

CMYK	Gray	OHTA
RGB	sRGB	Transparent
XYZ	YCbCr	YCC
YIQ	YPbPr	YUV

dither Images which suffer from severe contouring when reducing colors can be improved with this option.

global_colormap A value other than 0 creates one global colormap for a sequence of images.

measure_error A value other than 0 returns a measure of how closely the color reduced image matches the original. The mean error, normalized mean error, and normalized maximum mean error per pixel are computed. Obtain these values with the Get() method.

tree_depth By default, the color reduction uses a Oct-tree algorithm whose depth ranges from 1-8 which is optimally determined to allow the best representation of the image with the fastest computational speed and least amount of memory consumption. You can override the default with this parameter.

QueryColor() return numerical values corresponding to a color name.

$image->QueryColor(...)

Call QueryColor() with no parameters to return a list of known colors names or specify one or more color names to get these attributes: red, green, blue, and opacity value.

```
@colors = $image->QueryColor();
($red, $green, $blue, $opacity) = $image->QueryColor('red');
($red, $green, $blue, $opacity) = $image->QueryColor('#716bae');
```

QueryColorName() return a color name corresponding to the numerical values.

$image->QueryColorName(...)

QueryColorName() accepts one or more numerical values and returns their respective color name:

```
$color = $image->QueryColorName('rgba(65535,0,0,0)');
```

QueryFont() get font attributes.

$image->QueryFont(...)

Call QueryFont() with no parameters to return a list of known fonts or specify one or more font names to get these attributes: font name, description, family, style, stretch, weight, encoding, foundry, format, metrics, and glyphs values.

```
@fonts = $image->QueryFont();
$weight = ($image->QueryFont('Helvetica'))[5];
```

QueryFontMetric() query font metrics.

$image->QueryFontMetrics(font=¿*string*, ...)

QueryFontMetrics() accepts a font name and any parameter acceptable to Annotate(). The method returns these attributes associated with the given font:

- character width
- character height
- ascender

- descender
- text width
- text height
- maximum horizontal advance

For example,

```
@metrics = $image->QueryFontMetrics(font=>'arial.ttf', pointsize=>24);
```

QueryFormat() get image format attributes.

$image->QueryFormat(...)

Call QueryFormat() with no parameters to return a list of known image formats or specify one or more format names to get these attributes: adjoin, blob support, raw, format, decoder, encoder, description, and module.

```
@formats = $image->QueryFormat();
($adjoin, $blob_support, $raw, $decoder, $encoder, $description, $module) = $image->QueryFormat(
```

Raise() lighten or darken edges to create a 3-D effect.

$image->Raise(geometry=>*geometry*, width=>*integer*, height=>*integer*, raise=>*boolean*)

Raise() creates a simulated three-dimensional button-like effect by lightening and darkening the edges of the image. `Width` and `height` define the width of the vertical and horizontal edge of the effect. Use `Geometry` as a shortcut for `width` and `height` (e.g. 10x10).

A value other than 0 for `raise` simulates a raised button-like effect otherwise a sunken button-like effect is applied to the image.

Read() read one or more image files.

$image->Read(filename=>*float*, file=>*file-handle*)

filename the name of an image file.
file-handle read the image from an open filehandle.

The Read() method reads an image or image sequence from one or more filenames or the filehandle you specify. You can specify more than one filename but only one filehandle:

```
$image->Read(filename=$>$’logo.gif’);    # read a single GIF into
                                         # $image object.
$image->Read(’logo.jpg’, ’button.gif’);  # read two images.
$image->Read(’*.png’);                   # read all the PNG files in the
                                         # current directory.
$image->Read(file=$>$\*IMAGE);           # read from open Perl filehandle.
```

Read() returns the number of images that were successfully read.

ReduceNoise() smooth an image.

$image->ReduceNoise(radius=>*float*)

The ReduceNoise() method 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.

Resize() scale an image with a filter.

$image->Resize(geometry=>*geometry*, width=>*integer*, height=>*integer*, filter=>*string*, blur=>*float*)

Resize() 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	

The default is Lanczos.

Use `width` and `height` to specify the image size, or use `geometry` as a shortcut (e.g. 640x480).

Set `Blur` to a value greater than 1 to blur the image as it is scaled. A value less than 1 sharpens as the image is scaled.

Roll() offset and roll over an image.

$image->Roll(geometry=>*geometry*, x=>*integer*, y=>*integer*)

Roll() offsets an image as defined by `x` and `y`. `Geometry` represents + x + y as one parameter (e.g. +10+20).

Rotate() rotate an image.

$image->Rotate(degrees=>*float*, color=>*color-name*)

Rotate() rotates an image around the x axis by the number of degrees by `degrees`. Any empty spaces are filled with `color`.

Sample() sample an image.

$image->Sample(geometry=>*geometry*, width=>*integer*, height=>*integer*)

Sample() 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.

Use `width` and `height` to specify the image size, or use `geometry` as a shortcut (e.g. 640x480).

Scale() scale an image to given dimensions.

$image->Scale(geometry=>*geometry*, width=>*integer*, height=>*integer*)

Scale() changes the size of an image to the given dimensions. Use `width` and `height` to specify the image size, or use `geometry` as a shortcut (e.g. 640x480).

Segment() segment an image.

$image->Segment(geometry=>*geometry*, cluster_threshold=>*float*, smoothing_threshold=>*float*, colorspace=>*string*, verbose=>*boolean*)

Segment() segments an image by by analyzing the histograms of the color components and identifying units that are homogeneous. The default value for `cluster_threshold` is 1.0 and `smoothing_threshold` is 1.5. This can be represented with a shortcut geometry of 1.0x1.5.

Set() set an image attribute.

$image->Set(*attribute*, ...)

Set() accepts one or more image attributes listed in section 23.1 and sets their value.

Shade() shade the image with light source.

$image->Shade(geometry=>*geometry*, azimuth=>*float*, elevation=>*float*, color=>*boolean*)

Shade() 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. The geometry parameter is a shortcut for azimuth x elevation (e.g. 30x30).

Sharpen() sharpen an image.

$image->Sharpen(geometry=>*geometry*, radius=>*float*, sigma=>*float*)

Sharpen() 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 Sharpen() selects a suitable radius for you. `Geometry` represents radius x sigma as one parameter (e.g. 0x1).

Shave() shave pixels from the image edges.

$image->Border(geometry=>*geometry*, width=>*integer*, height=>*integer*)

This method shaves pixels from the image edges. `Geometry` represents *width x height* as one parameter (e.g. 10x5).

Shear() shear an image.

$image->Shear(geometry=>*geometry*, x=>*float*, y=>*float*, color=>*color-name*)

Shear() transforms an image by shearing it along the x or y axis. The `x` and `y` parameters specify the degree of shear and ranges from -179.9 to 179.9. `Geometry` represents x x y as one parameter (e.g. 30x60). Any empty spaces created when shearing are filled with `color`.

Signature() generate an SHA-256 message digest.

$image->Signature()

Signature() generates an SHA-256 message digest across all the image pixels. The signature can later be used to verify the color integrity of the image. Two images with the same signature are identical.

Solarize() apply solorization special effect.

$image->Solarize(threshold=>*float*)

Solarize() 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.

Spread() randomly displace pixels.

$image->Spread(amount=>*integer*)

Spead() is a special effects method that randomly displaces each pixel in a block defined by the `amount` parameter.

Stereo() create a stereo special effect.

$image->Stereo(image=>*image-handle*)

Stereo() 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.

Stegano() hide a digital watermark.

$image->Stegano(image=>*image-handle*, offset=>*integer*)

Use Stegano() 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.

Swirl() swirl pixels about image center.

$image->Swirl(degrees=>*float*)

The Swirl() method 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.

Texture() tile a texture on image background.

$image->Texture(texture=>*image-handle*)

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

Threshold() divide pixels based on intensity values.

$image->Threshold(threshold=>*integer*)

Threshold() 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.

Transform() resize or crop an image.

$image->Transform(geometry=>*string*, crop=>*string*)

Transform() behaves like Resize() or Crop() but rather than acting on the image, it returns a new image handle:

```
$slices = $image->Transform(crop=>'100x100')
```

Transparent() make color transparent.

$image->Transparent(color=>*color-name*, opacity=>*integer* fuzz=>*float*)

Transparent() 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.

Trim() remove background color from edges of image.

$image->Trim(fuzz=>*float*)

Trim() crops a rectangular box around the image to remove edges that are the background color.

By default the edge pixels must match in color exactly to be trimmed. 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.

UnsharpMask() sharpen an image.

$image->UnsharpMask(geometry=>*geometry*, radius=>*float*, sigma=>*float*, amount=>*float*, threshold=>*float*)

UnsharpMask() 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 UnsharpMask() selects a suitable radius for you. `Geometry` represents radius x sigma as one parameter (e.g. 0x1).

Wave() special effects filter.

$image->Wave(geometry=>*string*, amplitude=>*float*, wavelength=>*float*)

The Wave() 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. `Geometry` represents amplitude x wavelength as one parameter (e.g. 30x30).

Write() write one or more image files.

$image->Write(filename=>*float*, file=>*file-handle*)

Write() allows you to write a single or image or a sequence to a file or filehandle. You can specify more than one filename but only one filehandle:

```
$image->Write(filename=>’logo.gif’);     # write a single GIF image.
$image->Write(’logo.jpg’, ’button.gif’); # write two images.
$image->Write(’gif:-’);                  # write to STDOUT.
$image->[0]->Write(’logo.png’);          # write only first image
                                         # in a sequence.
$image->Write(file=>\*IMAGE);            # write to a open Perl filehandle.
```

Write() returns the number of images that were written.

23.3 Image::Magick Errors

Most Image::Magick methods return an undefined value if the operation was successful. When an error occurs, a message is returned with an embedded numeric status code. Look up the status code in table **??** to determine the reason the operation failed. The mnemonics are aliases for the the corresponding numeric codes.

Table23.1: Error and Warning Codes

Error and Warning Codes

Code	**Mnemonic**	**Description**
0	Success	Method completed without an error or warning.
300	ResourceLimitWarning	A program resource is exhausted (e.g. not enough memory).
305	TypeWarning	A font is unavailable; a substitution may have occured.
310	OptionWarning	An option parameter was malformed.
315	DelegateWarning	An ImageMagick *delegate* returned a warning.
320	MissingDelegateWarning	The image type can not be read or written because the required *delegate* is missing.
325	CorruptImageWarning	The image file may be corrupt.
330	FileOpenWarning	The image file could not be opened.
335	BlobWarning	A Binary Large OBject could not be allocated.
340	StreamWarning	There was a problem reading or writing from a stream.
345	CacheWarning	Pixels could not be saved to the pixel cache.
385	XServerWarning	An X resource is unavailable.
390	RegistryWarning	There was a problem getting or setting the registry.
395	ConfigurationWarning	There was a problem getting a configuration file.
400	ResourceLimitError	A program resource is exhausted (e.g. not enough memory).
405	TypeError	A font is unavailable; a substitution may have occured.
410	OptionError	An option parameter was malformed.
415	DelegateError	An ImageMagick *delegate* returned a warning.
420	MissingDelegateError	The image type can not be read or written because the required *delegate* is missing.
425	CorruptImageError	The image file may be corrupt.
430	FileOpenError	The image file could not be opened.
435	BlobError	A Binary Large OBject could not be allocated.
440	StreamError	There was a problem reading or writing from a stream.
445	CacheError	Pixels could not be saved to the pixel cache.
485	XServerError	An X resource is unavailable.
490	RegistryError	There was a problem getting or setting the registry.
495	ConfigurationError	There was a problem getting a configuration file.

24 Recognized Color Keyword Names

The following is the list of recognized color keywords that can be used whenever a color is needed for the ImageMagick command-line utilities or API methods. The color keyword names follow the W3C SVG 1.0 Specification with the addition of gray color names *gray1* thru *gray100*.

Table24.1: Color Names

ImageMagick Colors

Color	**Name**	**Color**	**Name**
aliceblue	rgba(240, 248, 255, 0)	gray71	rgba(181, 181, 181, 0)
antiquewhite	rgba(250, 235, 215, 0)	gray72	rgba(184, 184, 184, 0)
aqua	rgba(0, 255, 255, 0)	gray73	rgba(186, 186, 186, 0)
aquamarine	rgba(127, 255, 212, 0)	gray74	rgba(189, 189, 189, 0)
azure	rgba(240, 255, 255, 0)	gray75	rgba(191, 191, 191, 0)
beige	rgba(245, 245, 220, 0)	gray76	rgba(194, 194, 194, 0)
bisque	rgba(255, 228, 196, 0)	gray77	rgba(196, 196, 196, 0)
black	rgba(0, 0, 0, 0)	gray78	rgba(199, 199, 199, 0)
blanchedalmond	rgba(255, 235, 205, 0)	gray79	rgba(201, 201, 201, 0)
blue	rgba(0, 0, 255, 0)	gray8	rgba(20, 20, 20, 0)
blueviolet	rgba(138, 43, 226, 0)	gray80	rgba(204, 204, 204, 0)
brown	rgba(165, 42, 42, 0)	gray81	rgba(207, 207, 207, 0)
burlywood	rgba(222, 184, 135, 0)	gray82	rgba(209, 209, 209, 0)
cadetblue	rgba(95, 158, 160, 0)	gray83	rgba(212, 212, 212, 0)
chartreuse	rgba(127, 255, 0, 0)	gray84	rgba(214, 214, 214, 0)
chocolate	rgba(210, 105, 30, 0)	gray85	rgba(217, 217, 217, 0)
coral	rgba(255, 127, 80, 0)	gray86	rgba(219, 219, 219, 0)
cornflowerblue	rgba(100, 149, 237, 0)	gray87	rgba(222, 222, 222, 0)
cornsilk	rgba(255, 248, 220, 0)	gray88	rgba(224, 224, 224, 0)

ImageMagick Colors (continued)

Color	Name	Color	Name
crimson	rgba(220, 20, 60, 0)	gray89	rgba(227, 227, 227, 0)
cyan	rgba(0, 255, 255, 0)	gray9	rgba(23, 23, 23, 0)
darkblue	rgba(0, 0, 139, 0)	gray90	rgba(229, 229, 229, 0)
darkcyan	rgba(0, 139, 139, 0)	gray91	rgba(232, 232, 232, 0)
darkgoldenrod	rgba(184, 134, 11, 0)	gray92	rgba(235, 235, 235, 0)
darkgray	rgba(169, 169, 169, 0)	gray93	rgba(237, 237, 237, 0)
darkgreen	rgba(0, 100, 0, 0)	gray94	rgba(240, 240, 240, 0)
darkgrey	rgba(169, 169, 169, 0)	gray95	rgba(242, 242, 242, 0)
darkkhaki	rgba(189, 183, 107, 0)	gray96	rgba(245, 245, 245, 0)
darkmagenta	rgba(139, 0, 139, 0)	gray97	rgba(247, 247, 247, 0)
darkolivegreen	rgba(85, 107, 47, 0)	gray98	rgba(250, 250, 250, 0)
darkorange	rgba(255, 140, 0, 0)	gray99	rgba(252, 252, 252, 0)
darkorchid	rgba(153, 50, 204, 0)	green	rgba(0, 128, 0, 0)
darkred	rgba(139, 0, 0, 0)	greenyellow	rgba(173, 255, 47, 0)
darksalmon	rgba(233, 150, 122, 0)	grey	rgba(128, 128, 128, 0)
darkseagreen	rgba(143, 188, 143, 0)	honeydew	rgba(240, 255, 240, 0)
darkslateblue	rgba(72, 61, 139, 0)	hotpink	rgba(255, 105, 180, 0)
darkslategray	rgba(47, 79, 79, 0)	indianred	rgba(205, 92, 92, 0)
darkslategrey	rgba(47, 79, 79, 0)	indigo	rgba(75, 0, 130, 0)
darkturquoise	rgba(0, 206, 209, 0)	ivory	rgba(255, 255, 240, 0)
darkviolet	rgba(148, 0, 211, 0)	khaki	rgba(240, 230, 140, 0)
deeppink	rgba(255, 20, 147, 0)	lavender	rgba(230, 230, 250, 0)
deepskyblue	rgba(0, 191, 255, 0)	lavenderblush	rgba(255, 240, 245, 0)
dimgray	rgba(105, 105, 105, 0)	lawngreen	rgba(124, 252, 0, 0)
dimgrey	rgba(105, 105, 105, 0)	lemonchiffon	rgba(255, 250, 205, 0)
dodgerblue	rgba(30, 144, 255, 0)	lightblue	rgba(173, 216, 230, 0)
firebrick	rgba(178, 34, 34, 0)	lightcoral	rgba(240, 128, 128, 0)
floralwhite	rgba(255, 250, 240, 0)	lightcyan	rgba(224, 255, 255, 0)
forestgreen	rgba(34, 139, 34, 0)	lightgoldenrodyellow	rgba(250, 250, 210, 0)
fractal	rgba(128, 128, 128, 0)	lightgray	rgba(211, 211, 211, 0)
fuchsia	rgba(255, 0, 255, 0)	lightgreen	rgba(144, 238, 144, 0)
gainsboro	rgba(220, 220, 220, 0)	lightgrey	rgba(211, 211, 211, 0)
ghostwhite	rgba(248, 248, 255, 0)	lightpink	rgba(255, 182, 193, 0)
gold	rgba(255, 215, 0, 0)	lightsalmon	rgba(255, 160, 122, 0)
goldenrod	rgba(218, 165, 32, 0)	lightseagreen	rgba(32, 178, 170, 0)
gray	rgba(126, 126, 126, 0)	lightskyblue	rgba(135, 206, 250, 0)
gray0	rgba(0, 0, 0, 0)	lightslategray	rgba(119, 136, 153, 0)
gray1	rgba(3, 3, 3, 0)	lightslategrey	rgba(119, 136, 153, 0)
gray10	rgba(26, 26, 26, 0)	lightsteelblue	rgba(176, 196, 222, 0)
gray100	rgba(255, 255, 255, 0)	lightyellow	rgba(255, 255, 224, 0)
gray11	rgba(28, 28, 28, 0)	lime	rgba(0, 255, 0, 0)

ImageMagick Colors (continued)

Color	Name	Color	Name
gray12	rgba(31, 31, 31, 0)	limegreen	rgba(50, 205, 50, 0)
gray13	rgba(33, 33, 33, 0)	linen	rgba(250, 240, 230, 0)
gray14	rgba(36, 36, 36, 0)	magenta	rgba(255, 0, 255, 0)
gray15	rgba(38, 38, 38, 0)	maroon	rgba(128, 0, 0, 0)
gray16	rgba(41, 41, 41, 0)	mediumaquamarine	rgba(102, 205, 170, 0)
gray17	rgba(43, 43, 43, 0)	mediumblue	rgba(0, 0, 205, 0)
gray18	rgba(46, 46, 46, 0)	mediumorchid	rgba(186, 85, 211, 0)
gray19	rgba(48, 48, 48, 0)	mediumpurple	rgba(147, 112, 219, 0)
gray2	rgba(5, 5, 5, 0)	mediumseagreen	rgba(60, 179, 113, 0)
gray20	rgba(51, 51, 51, 0)	mediumslateblue	rgba(123, 104, 238, 0)
gray21	rgba(54, 54, 54, 0)	mediumspringgreen	rgba(0, 250, 154, 0)
gray22	rgba(56, 56, 56, 0)	mediumturquoise	rgba(72, 209, 204, 0)
gray23	rgba(59, 59, 59, 0)	mediumvioletred	rgba(199, 21, 133, 0)
gray24	rgba(61, 61, 61, 0)	midnightblue	rgba(25, 25, 112, 0)
gray25	rgba(64, 64, 64, 0)	mintcream	rgba(245, 255, 250, 0)
gray26	rgba(66, 66, 66, 0)	mistyrose	rgba(255, 228, 225, 0)
gray27	rgba(69, 69, 69, 0)	moccasin	rgba(255, 228, 181, 0)
gray28	rgba(71, 71, 71, 0)	navajowhite	rgba(255, 222, 173, 0)
gray29	rgba(74, 74, 74, 0)	navy	rgba(0, 0, 128, 0)
gray3	rgba(8, 8, 8, 0)	none	rgba(0, 0, 0, 255)
gray30	rgba(77, 77, 77, 0)	oldlace	rgba(253, 245, 230, 0)
gray31	rgba(79, 79, 79, 0)	olive	rgba(128, 128, 0, 0)
gray32	rgba(82, 82, 82, 0)	olivedrab	rgba(107, 142, 35, 0)
gray33	rgba(84, 84, 84, 0)	orange	rgba(255, 165, 0, 0)
gray34	rgba(87, 87, 87, 0)	orangered	rgba(255, 69, 0, 0)
gray35	rgba(89, 89, 89, 0)	orchid	rgba(218, 112, 214, 0)
gray36	rgba(92, 92, 92, 0)	palegoldenrod	rgba(238, 232, 170, 0)
gray37	rgba(94, 94, 94, 0)	palegreen	rgba(152, 251, 152, 0)
gray38	rgba(97, 97, 97, 0)	paleturquoise	rgba(175, 238, 238, 0)
gray39	rgba(99, 99, 99, 0)	palevioletred	rgba(219, 112, 147, 0)
gray4	rgba(10, 10, 10, 0)	papayawhip	rgba(255, 239, 213, 0)
gray40	rgba(102, 102, 102, 0)	peachpuff	rgba(255, 218, 185, 0)
gray41	rgba(105, 105, 105, 0)	peru	rgba(205, 133, 63, 0)
gray42	rgba(107, 107, 107, 0)	pink	rgba(255, 192, 203, 0)
gray43	rgba(110, 110, 110, 0)	plum	rgba(221, 160, 221, 0)
gray44	rgba(112, 112, 112, 0)	powderblue	rgba(176, 224, 230, 0)
gray45	rgba(115, 115, 115, 0)	purple	rgba(128, 0, 128, 0)
gray46	rgba(117, 117, 117, 0)	red	rgba(255, 0, 0, 0)
gray47	rgba(120, 120, 120, 0)	rosybrown	rgba(188, 143, 143, 0)
gray48	rgba(122, 122, 122, 0)	royalblue	rgba(65, 105, 225, 0)
gray49	rgba(125, 125, 125, 0)	saddlebrown	rgba(139, 69, 19, 0)

ImageMagick Colors (continued)

Color	Name	Color	Name
gray5	rgba(13, 13, 13, 0)	salmon	rgba(250, 128, 114, 0)
gray50	rgba(127, 127, 127, 0)	sandybrown	rgba(244, 164, 96, 0)
gray51	rgba(130, 130, 130, 0)	seagreen	rgba(46, 139, 87, 0)
gray52	rgba(133, 133, 133, 0)	seashell	rgba(255, 245, 238, 0)
gray53	rgba(135, 135, 135, 0)	sienna	rgba(160, 82, 45, 0)
gray54	rgba(138, 138, 138, 0)	silver	rgba(192, 192, 192, 0)
gray55	rgba(140, 140, 140, 0)	skyblue	rgba(135, 206, 235, 0)
gray56	rgba(143, 143, 143, 0)	slateblue	rgba(106, 90, 205, 0)
gray57	rgba(145, 145, 145, 0)	slategray	rgba(112, 128, 144, 0)
gray58	rgba(148, 148, 148, 0)	slategrey	rgba(112, 128, 144, 0)
gray59	rgba(150, 150, 150, 0)	snow	rgba(255, 250, 250, 0)
gray6	rgba(15, 15, 15, 0)	springgreen	rgba(0, 255, 127, 0)
gray60	rgba(153, 153, 153, 0)	steelblue	rgba(70, 130, 180, 0)
gray61	rgba(156, 156, 156, 0)	tan	rgba(210, 180, 140, 0)
gray62	rgba(158, 158, 158, 0)	teal	rgba(0, 128, 128, 0)
gray63	rgba(161, 161, 161, 0)	thistle	rgba(216, 191, 216, 0)
gray64	rgba(163, 163, 163, 0)	tomato	rgba(255, 99, 71, 0)
gray65	rgba(166, 166, 166, 0)	turquoise	rgba(64, 224, 208, 0)
gray66	rgba(168, 168, 168, 0)	violet	rgba(238, 130, 238, 0)
gray67	rgba(171, 171, 171, 0)	wheat	rgba(245, 222, 179, 0)
gray68	rgba(173, 173, 173, 0)	white	rgba(255, 255, 255, 0)
gray69	rgba(176, 176, 176, 0)	whitesmoke	rgba(245, 245, 245, 0)
gray7	rgba(18, 18, 18, 0)	yellow	rgba(255, 255, 0, 0)
gray70	rgba(179, 179, 179, 0)	yellowgreen	rgba(154, 205, 50, 0)
gray71	rgba(181, 181, 181, 0)		

References

[1] Dalrymple, F., Pringle, S. (1999) Cognitive Disfunction. **49**, 581–623

A Appendix A