Installation

UNIX

Is PerlMagick available from your system RPM repository? For example, on our CentOS system, we install PerlMagick thusly:

```
yum install ImageMagick-perl
```

If not, you must install PerlMagick from the ImageMagick source distribution. Download the latest source release.

Unpack the distribution with this command:

```
tar xvzf ImageMagick.tar.gz
```

Next configure and compile ImageMagick:

```
$ cd ImageMagick-7.0.10
$ ./configure -with-perl
$ make
```

If ImageMagick / PerlMagick configured and compiled without complaint, you are ready to install it on your system. Administrator privileges are required to install. To install, type

```
sudo make install
```

You may need to configure the dynamic linker run-time bindings:

```
sudo ldconfig /usr/local/lib
```

Finally, verify the PerlMagick install worked properly, type

```
perl -MImage::Magick -le 'print Image::Magick->QuantumDepth'
```

Congratulations, you have a working ImageMagick distribution and you are ready to use PerlMagick to convert, compose, or edit your images.

Windows XP / Windows 2000

ImageMagick must already be installed on your system. Also, the ImageMagick source distribution for Windows 2000 is required. You must also have the `nmake` from the Visual C++ or J++ development environment. Copy `\bin\IMagick.dll` and `\bin\X11.dll` to a directory in your dynamic load path such as `c:\perl\site\5.00502`.

Next, type

```
cd PerlMagick
perl Makefile.nt
nmake
nmake install
```

See the PerlMagick Windows HowTo page for further installation instructions.

Running the Regression Tests

To verify a correct installation, type

```
make test
```

Use `nmake test` under Windows. There are a few demonstration scripts available to exercise many of the functions PerlMagick can perform. Type

```
cd demo
make
```

You are now ready to utilize the PerlMagick methods from within your Perl scripts.

Overview

Any script that wants to use PerlMagick methods must first define the methods within its namespace and instantiate an image object. Do this with:

```
use Image::Magick;
```

```
$image = Image::Magick->new;
```

PerlMagick is *quantum* aware. You can request a specific quantum depth when you instantiate an image object:

```
use Image::Magick::Q16;
```

```
$image = Image::Magick::Q16->new;
```

The new() method takes the same parameters as SetAttribute . For example,

```
$image = Image::Magick->new(size=>'384x256');
```

Next you will want to read an image or image sequence, manipulate it, and then display or write it. The input and output methods for PerlMagick are defined in Read or Write an Image. See Set an Image Attribute for methods that affect the way an image is read or written. Refer to Manipulate an Image for a list of methods to transform an image. Get an Image Attribute describes how to retrieve an attribute for an image. Refer to Create an Image Montage for details about tiling your images as thumbnails on a background. Finally, some methods do not neatly fit into any of the categories just mentioned. Review Miscellaneous Methods for a list of these methods.

Once you are finished with a PerlMagick object you should consider destroying it. Each image in an image sequence is stored in virtual memory. This can potentially add up to mebibytes of memory. Upon destroying a PerlMagick object, the memory is returned for use by other Perl methods. The recommended way to destroy an object is with `undef`:

```
undef $image;
```

To delete all the images but retain the `Image::Magick` object use

```
@$image = ();
```

and finally, to delete a single image from a multi-image sequence, use

```
undef $image->[$x];
```

The next section illustrates how to use various PerlMagick methods to manipulate an image sequence.

Some of the PerlMagick methods require external programs such as Ghostscript. This may require an explicit path in your PATH environment variable to work properly. For example (in Unix),

```
$ENV{PATH}' . "='/../bin:/usr/bin:/usr/local/bin';
```

Example Script

Here is an example script to get you started:

```
#!/usr/local/bin/perl
use Image::Magick;

my($image, $x);

$image = Image::Magick->new;
$x = $image->Read('girl.png', 'logo.png', 'rose.png');
warn "$x" if "$x";

$x = $image->Crop(geometry=>'100x100+100+100');
warn "$x" if "$x";

$x = $image->Write('x.png');
warn "$x" if "$x";
```

The script reads three images, crops them, and writes a single image as a GIF animation sequence. In many cases you may want to access individual images of a sequence. The next example illustrates how this done:

```
#!/usr/local/bin/perl
use Image::Magick;

my($image, $p, $q);

$image = new Image::Magick;
$image->Read('x1.png');
$image->Read('j*.jpg');
$image->Read('k.miff[1, 5, 3]');
$image->Contrast();
for ($x = 0; $image->[$x]; $x++)
{
  $image->[$x]->Frame('100x200') if $image->[$x]->Get('magick') eq 'GIF';
  undef $image->[$x] if $image->[$x]->Get('columns') < 100;
}
$p = $image->[1];
$p->Draw(stroke=>'red', primitive=>'rectangle', points=>20,20 100,100');
$q = $p->Montage();
undef $image;
$q->Write('x.miff');
```

Suppose you want to start out with a 100 by 100 pixel white canvas with a red pixel in the center. Try

```
$image = Image::Magick->new;
$image->Set(size=>'100x100');
$image->ReadImage('canvas:white');
$image->Set('pixel[49,49]'=>'red');
```

Here we reduce the intensity of the red component at (1,1) by half:

```
@pixels = $image->GetPixel(x=>1,y=>1);
$pixels[0]*=0.5;
$image->SetPixel(x=>1,y=>1,color=>\@pixels);
```

Or suppose you want to convert your color image to grayscale:

```
$image->Quantize(colorspace=>'gray');
```

Let's annotate an image with a Taipai TrueType font:

```
$text = 'Works like magick!';
$image->Annotate(font=>'kai.ttf', pointsize=>40, fill=>'green', text=>$text);
```

Perhaps you want to extract all the pixel intensities from an image and write them to STDOUT:

```
@pixels = $image->GetPixels(map=>'I', height=>$height, width=>$width,
normalize=>true);
binmode STDOUT;
print pack('B*',join('',@pixels));
```

Other clever things you can do with a PerlMagick objects include

```
$i = $#$p"+1";   # return the number of images associated with object p
push(@$q, @$p);  # push the images from object p onto object q
@$p = ();        # delete the images but not the object p
$p->Convolve([1, 2, 1, 2, 4, 2, 1, 2, 1]);   # 3x3 Gaussian kernel
```

Read or Write an Image

Use the methods listed below to either read, write, or display an image or image sequence:

Read or Write Methods

Method	Parameters	Return Value	Description
Read	one or more filenames	the number of images read	read an image or image sequence
Write	filename	the number of images written	write an image or image sequence
Display	server name	the number of images displayed	display the image or image sequence to an X server
Animate	server name	the number of images animated	animate image sequence to an X server

For convenience, methods Write(), Display(), and Animate() can take any parameter that SetAttribute knows about. For example,

```
$image->Write(filename=>'image.png', compression=>'None');
```

Use `-` as the filename to method Read() to read from standard in or to method Write() to write to standard out:

```
binmode STDOUT;
$image->Write('png:-');
```

To read an image in the GIF format from a PERL filehandle, use:

```
$image = Image::Magick->new;
open(IMAGE, 'image.gif');
$image->Read(file=>\*IMAGE);
close(IMAGE);
```

To write an image in the PNG format to a PERL filehandle, use:

```
$filename = "image.png";
open(IMAGE, ">$filename");
$image->Write(file=>\*IMAGE, filename=>$filename);
close(IMAGE);
```

Note, reading from or writing to a Perl filehandle may fail under Windows due to different versions of the C-runtime libraries between ImageMagick and the ActiveState Perl distributions or if one of the DLL's is linked with the /MT option. See Potential Errors Passing CRT Objects Across DLL Boundaries for an explanation.

If `%0Nd, %0No, or %0Nx` appears in the filename, it is interpreted as a printf format specification and the specification is replaced with the specified decimal, octal, or hexadecimal encoding of the scene number. For example,

```
image%03d.miff
```

converts files image000.miff, image001.miff, etc.

You can optionally add *Image* to any method name. For example, ReadImage() is an alias for method Read().

Manipulate an Image

Once you create an image with, for example, method ReadImage() you may want to operate on it. Below is a list of all the image manipulations methods available to you with PerlMagick. There are examples of select PerlMagick methods. Here is an example call to an image manipulation method:

```
$image->Crop(geometry=>'100x100+10+20');
$image->[$x]->Frame("100x200");
```

And here is a list of other image manipulation methods you can call:

Image Manipulation Methods

Method	Parameters	Description

AdaptiveBlur	geometry=>*geometry*, radius=>*double*, sigma=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	adaptively blur the image with a Gaussian operator of the given radius and standard deviation (sigma). Decrease the effect near edges.
AdaptiveResize	geometry=>*geometry*, width=>*integer*, height=>*integer*, filter=>{Point, Box, Triangle, Hermite, Hanning, Hamming, Blackman, Gaussian, Quadratic, Cubic, Catrom, Mitchell, Lanczos, Bessel, Sinc}, support=>*double*, blur=>*double*	adaptively resize image using data dependant triangulation. Specify `blur` > 1 for blurry or < 1 for sharp
AdaptiveSharpen	geometry=>*geometry*, radius=>*double*, sigma=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	adaptively sharpen the image with a Gaussian operator of the given radius and standard deviation (sigma). Increase the effect near edges.
AdaptiveThreshold	geometry=>*geometry*, width=>*integer*, height=>*integer*, bias=>*double*	local adaptive thresholding.
AddNoise	noise=>{Uniform, Gaussian, Multiplicative, Impulse, Laplacian, Poisson}, attenuate=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	add noise to an image
AffineTransform	affine=>*array of float values*, translate=>*float*, *float*, scale=> *float*, *float*, rotate=>*float*, skewX=>*float*, skewY=>*float*, interpolate={Average, Bicubic, Bilinear, Filter, Integer, Mesh, NearestNeighbor}, background=>*color name*	affine transform image
Affinity	image=>*image-handle*, method=>{None, FloydSteinberg, Riemersma}	choose a particular set of colors from this image
Annotate	text=>*string*, font=>*string*, family=>*string*, style=>{Normal, Italic, Oblique, Any}, stretch=>{Normal, UltraCondensed, ExtraCondensed, Condensed, SemiCondensed, SemiExpanded, Expanded, ExtraExpanded, UltraExpanded}, weight=>*integer*, pointsize=>*integer*, density=>*geometry*, stroke=>*color*	annotate an image with text. See QueryFontMetrics to get font metrics without rendering any text.

	name, strokewidth=>*integer*, fill=>*color name*, undercolor=>*color name*, kerning=>*float*, geometry=>*geometry*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}, antialias=>{true, false}, x=>*integer*, y=>*integer*, affine=>*array of float values*, translate=>*float*, *float*, scale=>*float*, *float*, rotate=>*float*. skewX=>*float*, skewY=> *float*, align=>{Left, Center, Right}, encoding=>{UTF-8}, interline-spacing=>*double*, interword-spacing=>*double*, direction=>{right-to-left, left-to-right}, decorate=>{none, underline, overline, line-through}	
AutoGamma	channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	automagically adjust gamma level of image
AutoLevel	channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	automagically adjust color levels of image
AutoOrient		adjusts an image so that its orientation is suitable for viewing (i.e. top-left orientation)
AutoThreshold	method=>{Kapur, OTSU, Triangle}	automatically perform image thresholding
BlackThreshold	threshold=>*color*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	force all pixels below the threshold intensity into black
BlueShift	factor=>*double*,	simulate a scene at nighttime in the moonlight. Start with a factor of 1.5.
Blur	geometry=>*geometry*, radius=>*double*, sigma=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	reduce image noise and reduce detail levels with a Gaussian operator of the given radius and standard deviation (sigma).
Border	geometry=>*geometry*, width=>*integer*, height=>*integer*,	surround the image with a border of color

	bordercolor=>*color name*, compose=>{Undefined, Add, Atop, Blend, Bumpmap, Clear, ColorBurn, ColorDodge, Colorize, CopyBlack, CopyBlue, CopyCMYK, Cyan, CopyGreen, Copy, CopyMagenta, CopyAlpha, CopyRed, RGB, CopyYellow, Darken, Dst, Difference, Displace, Dissolve, DstAtop, DstIn, DstOut, DstOver, Dst, Exclusion, HardLight, Hue, In, Lighten, Luminize, Minus, Modulate, Multiply, None, Out, Overlay, Over, Plus, ReplaceCompositeOp, Saturate, Screen, SoftLight, Src, SrcAtop, SrcIn, SrcOut, SrcOver, Src, Subtract, Threshold, Xor },	
CannyEdge	geometry=>*geometry*, radius=>*double*, sigma=>*double*, 'lower-percent'=>*double*, 'upper-percent'=>*double*	use a multi-stage algorithm to detect a wide range of edges in the image (e.g. CannyEdge('0x1+10% +40%')).
Charcoal	geometry=>*geometry*, radius=>*double*, sigma=>*double*	simulate a charcoal drawing
Chop	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}	chop an image
CLAHE	geometry=>*geometry*, width=>*integer*, height=>*integer*, number-bins=>*integer*, clip-limit=>*double*	contrast limited adaptive histogram equalization. *width*, *height* divides the image into tiles. *number-bins* is the number of histogram bins per tile (min 2, max 256). *clip-limit* is the contrast limit for localised changes in contrast. A clip-limit of 2 to 3 is a good starting place.
Clamp	channel=>{Red, RGB, All, etc.}	set each pixel whose value is below zero to zero and any the pixel whose value is above the quantum range to the quantum range (e.g. 65535) otherwise the pixel value

		remains unchanged.
Clip	id=>*name*, inside=>*{true, false}*,	apply along a named path from the 8BIM profile.
ClipMask	mask=>*image-handle*	clip image as defined by the image mask
Clut	image=>*image-handle*, interpolate={Average, Bicubic, Bilinear, Filter, Integer, Mesh, NearestNeighbor}, channel=>{Red, RGB, All, etc.}	apply a color lookup table to an image sequence
Color	color=>*color name*	set the entire image to this color.
ColorDecisionList	filename=>*string*,	color correct with a color decision list.
Colorize	fill=>*color name*, blend=>*string*	colorize the image with the fill color
ColorMatrix	matrix=>*array of float values*	apply color correction to the image. Although you can use variable sized matrices, typically you use a 5 x 5 for an RGBA image and a 6x6 for CMYKA. A 6x6 matrix is required for offsets (populate the last column with normalized values).
Colorspace	colorspace=>{RGB, Gray, Transparent, OHTA, XYZ, YCbCr, YCC, YIQ, YPbPr, YUV, CMYK}	set the image colorspace
Comment	string	add a comment to your image
ColorThreshold	start-color=>*color*, stop-color=>*color*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	force all pixels below the threshold intensity into black
CompareLayers	method=>{any, clear, overlay}	compares each image with the next in a sequence and returns the minimum bounding region of any pixel differences it discovers. Images do not have to be the same size, though it is best that all the images are coalesced (images are all the same size, on a flattened canvas, so as to represent exactly how a specific frame should look).

Composite	image=>*image-handle*, compose=>{Undefined, Add, Atop, Blend, Bumpmap, Clear, ColorBurn, ColorDodge, Colorize, CopyBlack, CopyBlue, CopyCMYK, Cyan, CopyGreen, Copy, CopyMagenta, CopyAlpha, CopyRed, RGB, CopyYellow, Darken, Dst, Difference, Displace, Dissolve, DstAtop, DstIn, DstOut, DstOver, Dst, Exclusion, HardLight, Hue, In, Lighten, Luminize, Minus, Modulate, Multiply, None, Out, Overlay, Over, Plus, ReplaceCompositeOp, Saturate, Screen, SoftLight, Src, SrcAtop, SrcIn, SrcOut, SrcOver, Src, Subtract, Threshold, Xor }, mask=>*image-handle*, geometry=>*geometry*, x=>*integer*, y=>*integer*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}, opacity=>*integer*, tile=>{True, False}, rotate=>*double*, color=>*color name*, blend=>*geometry*, interpolate=>{undefined, average, bicubic, bilinear, filter, integer, mesh, nearest-neighbor, spline}, clip-to-self=>{True, False}	composite one image onto another. Use the rotate parameter in concert with the tile parameter.
ConnectedComponents	connectivity=>*integer*,	connected-components uniquely labeled, choose from 4 or 8 way connectivity.
Contrast	sharpen=>{True, False}	enhance or reduce the image contrast
ContrastStretch	levels=>*string*, 'black-point'=>*double*, 'white-point'=>*double*, channel=>{Red, RGB, All, etc.}	improve the contrast in an image by `stretching' the range of intensity values
Convolve	coefficients=>*array of float values*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, bias=>*double*	apply a convolution kernel to the image. Given a kernel *order* , you would supply *order***order* float values (e.g. 3x3 implies 9 values).
CopyPixels	image=>*image-handle*, geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*,	copy pixels from the image as defined by the `widthxheight+x+y` to image at offset `+dx,+dy`.

	offset=>*geometry*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}, dx=>*integer*, dy=>*integer*	
Crop	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*, fuzz=>*double*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}	crop an image
CycleColormap	amount=>*integer*	displace image colormap by amount
Decipher	passphrase=>*string*	convert cipher pixels to plain pixels
Deconstruct		break down an image sequence into constituent parts
Deskew	geometry=>*string*,threshold=>*double*	straighten the image
Despeckle		reduce the speckles within an image
Difference	image=>*image-handle*	compute the difference metrics between two images
Distort	points=>*array of float values*, method=>{Affine, AffineProjection, ScaleRotateTranslate, SRT, Perspective, PerspectiveProjection, BilinearForward, BilinearReverse, Polynomial, Arc, Polar, DePolar, Barrel, BarrelInverse, Shepards, Resize}, 'virtual-pixel'=>{Background Black Constant Dither Edge Gray Mirror Random Tile Transparent White}, 'best-fit'=>{True, False}	distort image
Draw	primitive=>{point, line, rectangle, arc, ellipse, circle, path, polyline, polygon, bezier, color, matte, text, *@filename*}, points=>*string* , method=>*{Point, Replace, Floodfill, FillToBorder, Reset}*, stroke=>*color name*, fill=>*color name*, font=>*string*, pointsize=>*integer*, strokewidth=>*float*, antialias=>{true, false}, bordercolor=>*color name*, x=>*float*, y=>*float*, dash-offset=>*float*, dash-	annotate an image with one or more graphic primitives.

	pattern=>*array of float values*, affine=>*array of float values*, translate=>*float*, *float*, scale=>*float*, *float*, rotate=>*float*, skewX=>*float*, skewY=>*float*, interpolate=>{undefined, average, bicubic, bilinear, mesh, nearest-neighbor, spline}, kerning=>*float*, text=>*string*, vector-graphics=>*string*, interline-spacing=>*double*, interword-spacing=>*double*, direction=>{right-to-left, left-to-right}	
Encipher	passphrase=>*string*	convert plain pixels to cipher pixels
Edge	radius=>*double*	enhance edges within the image with a convolution filter of the given radius.
Emboss	geometry=>*geometry*, radius=>*double*, sigma=>*double*	emboss the image with a convolution filter of the given radius and standard deviation (sigma).
Enhance		apply a digital filter to enhance a noisy image
Equalize	channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	perform histogram equalization to the image
Extent	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*, fuzz=>*double*, background=>*color name*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}	set the image size
Evaluate	value=>*double*, operator=>*{Add, And, Divide, LeftShift, Max, Min, Multiply, Or, Rightshift, RMS, Subtract, Xor}*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	apply an arithmetic, relational, or logical expression to the image
Filter	kernel=>*string*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, bias=>*double*	apply a convolution kernel to the image.
Flip		reflect the image scanlines in the vertical direction
Flop		reflect the image scanlines

		in the horizontal direction
FloodfillPaint	geometry=>*geometry*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, x=>*integer*, y=>*integer* , fill=>*color name*, bordercolor=>*color name*, fuzz=>*double*, invert=>{True, False}	changes the color value of any pixel that matches the color of the target pixel and is a neighbor. If you specify a border color, the color value is changed for any neighbor pixel that is not that color.
ForwardFourierTransform	magnitude=>{True, False}	implements the forward discrete Fourier transform (DFT)
Frame	geometry=>*geometry*, width=>*integer*, height=>*integer*, inner=>*integer*, outer=>*integer*, fill=>*color name*, compose=>{Undefined, Add, Atop, Blend, Bumpmap, Clear, ColorBurn, ColorDodge, Colorize, CopyBlack, CopyBlue, CopyCMYK, Cyan, CopyGreen, Copy, CopyMagenta, CopyAlpha, CopyRed, RGB, CopyYellow, Darken, Dst, Difference, Displace, Dissolve, DstAtop, DstIn, DstOut, DstOver, Dst, Exclusion, HardLight, Hue, In, Lighten, Luminize, Minus, Modulate, Multiply, None, Out, Overlay, Over, Plus, ReplaceCompositeOp, Saturate, Screen, SoftLight, Src, SrcAtop, SrcIn, SrcOut, SrcOver, Src, Subtract, Threshold, Xor },	surround the image with an ornamental border
Function	parameters=>*array of float values*, function=>{Sin}, 'virtual-pixel'=>{Background Black Constant Dither Edge Gray Mirror Random Tile Transparent White}	apply a function to the image
Gamma	gamma=>*string*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	gamma correct the image
GaussianBlur	geometry=>*geometry*, radius=>*double*, sigma=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	reduce image noise and reduce detail levels with a Gaussian operator of the given radius and standard deviation (sigma).
GetPixel	geometry=>*geometry*, channel=>{All, Default, Alpha,	get a single pixel. By default normalized pixel

	Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, normalize=>{true, false}, x=>*integer*, y=>*integer*	values are returned.
GetPixels	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*, map=>*string*, normalize=>{true, false}	get image pixels as defined by the map (e.g. "RGB", "RGBA", etc.). By default non-normalized pixel values are returned.
Grayscale	channel=>{Average, Brightness, Lightness, Rec601Luma, Rec601Luminance, Rec709Luma, Rec709Luminance, RMS}	convert image to grayscale
HaldClut	image=>*image-handle*, channel=>{Red, RGB, All, etc.}	apply a Hald color lookup table to an image sequence
HoughLine	geometry=>*geometry*, width=>*double*, height=>*double*, threshold=>*double*	identify lines in the image (e.g. HoughLine('9x9+195')).
Identify	file=>*file*, features=>*distance*, moments=>{True, False}, unique=>{True, False}	identify the attributes of an image
Implode	amount=>*double*, interpolate=>{undefined, average, bicubic, bilinear, mesh, nearest-neighbor, spline}	implode image pixels about the center
InverseDiscreteFourierTransform	magnitude=>{True, False}	implements the inverse discrete Fourier transform (DFT)
Kmeans	geometry=>*geometry*, 'colors'=>*double*, 'iterations'=>*double*, 'tolerance'=>*double*	K means color reduction.
Kuwahara	geometry=>*geometry*, radius=>*double*, sigma=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	edge preserving noise reduction filter
Label	string	assign a label to an image
Layers	method=>{coalesce, compare-any, compare-clear, compare-over, composite, dispose, flatten, merge, mosaic, optimize, optimize-image, optimize-plus, optimize-trans, remove-dups, remove-zero}, compose=>{Undefined, Add, Atop, Blend, Bumpmap, Clear, ColorBurn, ColorDodge, Colorize, CopyBlack, CopyBlue, CopyCMYK, Cyan, CopyGreen, Copy, CopyMagenta,	compare each image the GIF disposed forms of the previous image in the sequence. From this, attempt to select the smallest cropped image to replace each frame, while preserving the results of the animation.

	CopyAlpha, CopyRed, RGB, CopyYellow, Darken, Dst, Difference, Displace, Dissolve, DstAtop, DstIn, DstOut, DstOver, Dst, Exclusion, HardLight, Hue, In, Lighten, LinearLight, Luminize, Minus, Modulate, Multiply, None, Out, Overlay, Over, Plus, ReplaceCompositeOp, Saturate, Screen, SoftLight, Src, SrcAtop, SrcIn, SrcOut, SrcOver, Src, Subtract, Threshold, Xor }, dither=>{true, false}	
Level	levels=>*string*, 'black-point'=>*double*, 'gamma'=>*double*, 'white-point'=>*double*, channel=>{Red, RGB, All, etc.}	adjust the level of image contrast
LevelColors	invert=>>{True, False}, 'black-point'=>*string*, 'white-point'=>*string*, channel=>{Red, RGB, All, etc.}	level image with the given colors
LinearStretch	levels=>*string*, 'black-point'=>*double*, 'white-point'=>*double*	linear with saturation stretch
LiquidResize	geometry=>*geometry*, width=>*integer*, height=>*integer*, delta-x=>*double*, rigidity=>*double*	rescale image with seam-carving.
Magnify		double the size of the image with pixel art scaling
Mask	mask=>*image-handle*	composite image pixels as defined by the mask
MatteFloodfill	geometry=>*geometry*, x=>*integer*, y=>*integer* , matte=>*integer*, bordercolor=>*color name*, fuzz=>*double*, invert=>{True, False}	changes the matte value of any pixel that matches the color of the target pixel and is a neighbor. If you specify a border color, the matte value is changed for any neighbor pixel that is not that color.
MeanShift	geometry=>*geometry*, width=>*double*, height=>*double*, distance=>*double*	delineate arbitrarily shaped clusters in the image (e.g. MeanShift('7x7+10%')).
MedianFilter	geometry=>*geometry*, width=>*integer*, height=>*integer*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	replace each pixel with the median intensity pixel of a neighborhood.
Minify		half the size of an image
Mode	geometry=>*geometry*,	make each pixel the

	width=>*integer*, height=>*integer*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	*predominant color* of the neighborhood.
Modulate	factor=>*geometry*, brightness=>*double*, saturation=>*double*, hue=>*double*, lightness=>*double*, whiteness=>*double*, blackness=>*double*	vary the brightness, saturation, and hue of an image by the specified percentage
Morphology	kernel=>*string*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, iterations=>*integer*	apply a morphology method to the image.
MotionBlur	geometry=>*geometry*, radius=>*double*, sigma=>*double*, angle=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	reduce image noise and reduce detail levels with a Gaussian operator of the given radius and standard deviation (sigma) at the given angle to simulate the effect of motion
Negate	gray=>{True, False}, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	replace each pixel with its complementary color (white becomes black, yellow becomes blue, etc.)
Normalize	channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	transform image to span the full range of color values
OilPaint	radius=>*integer*	simulate an oil painting
Opaque	color=>*color name*, fill=>*color name*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, invert=>{True, False}	change this color to the fill color within the image
OrderedDither	threshold=>{threshold, checks, o2x2, o3x3, o4x4, o8x8, h4x4a, h6x6a, h8x8a, h4x4o, h6x6o, h8x8o, h16x16o, hlines6x4}, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	order dither image
Perceptible	epsilon=>*double*, channel=>{Red, RGB, All, etc.}	set each pixel whose value is less than |*epsilon*| to -*epsilon* or *epsilon*

		(whichever is closer) otherwise the pixel value remains unchanged..
Polaroid	caption=>*string*, angle=>*double*, pointsize=>*double*, font=>*string*, stroke=> *color name*, strokewidth=>*integer*, fill=>*color name*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}, background=>*color name*	simulate a Polaroid picture.
Posterize	levels=>*integer*, dither=>{True, False}	reduce the image to a limited number of color level
Profile	name=>*string*, profile=>*blob*, rendering-intent=>{Undefined, Saturation, Perceptual, Absolute, Relative}, black-point-compensation=>{True, False}	add or remove ICC or IPTC image profile; name is formal name (e.g. ICC or filename; set profile to `''` to remove profile
Quantize	colors=>*integer*, colorspace=>{RGB, Gray, Transparent, OHTA, XYZ, YCbCr, YIQ, YPbPr, YUV, CMYK, sRGB, HSL, HSB}, treedepth=> *integer*, dither=>{True, False}, dither-method=>{Riemersma, Floyd-Steinberg}, measure_error=>{True, False}, global_colormap=>{True, False}, transparent-color=>*color*	preferred number of colors in the image
Raise	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*, raise=>{True, False}	lighten or darken image edges to create a 3-D effect
RangeThreshold	geometry=>*geometry*, 'low-black'=>*double*, 'low-white'=>*double*, 'high-white'=>*double*, 'high-black'=>*double*	combine soft and hard image thresholding.
ReduceNoise	geometry=>*geometry*, width=>*integer*, height=>*integer*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	reduce noise in the image with a noise peak elimination filter
Remap	image=>*image-handle*, dither=>{true, false}, dither-method=>{Riemersma, Floyd-Steinberg}	replace the colors of an image with the closest color from a reference image.
Resample	density=>*geometry*, x=>*double*, y=>*double*, filter=>{Point, Box,	resample image to desired resolution. Specify `blur`

	Triangle, Hermite, Hanning, Hamming, Blackman, Gaussian, Quadratic, Cubic, Catrom, Mitchell, Lanczos, Bessel, Sinc}, support=>*double*	> 1 for blurry or < 1 for sharp
Resize	geometry=>*geometry*, width=>*integer*, height=>*integer*, filter=>{Point, Box, Triangle, Hermite, Hanning, Hamming, Blackman, Gaussian, Quadratic, Cubic, Catrom, Mitchell, Lanczos, Bessel, Sinc}, support=>*double*, blur=>*double*	scale image to desired size. Specify `blur` > 1 for blurry or < 1 for sharp
Roll	geometry=>*geometry*, x=>*integer*, y=>*integer*	roll an image vertically or horizontally
Rotate	degrees=>*double*, background=>*color name*	rotate an image
RotationalBlur	geometry=>*geometry*, angle=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	radial blur the image.
Sample	geometry=>*geometry*, width=>*integer*, height=>*integer*	scale image with pixel sampling.
Scale	geometry=>*geometry*, width=>*integer*, height=>*integer*	scale image to desired size
Segment	colorspace=>{RGB, Gray, Transparent, OHTA, XYZ, YCbCr, YCC, YIQ, YPbPr, YUV, CMYK}, verbose={True, False}, cluster-threshold=>*double*, smoothing-threshold=*double*	segment an image by analyzing the histograms of the color components and identifying units that are homogeneous
SelectiveBlur	geometry=>*geometry*, radius=>*double*, sigma=>*double*, threshold=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	selectively blur pixels within a contrast threshold.
Separate	channel=>{Red, RGB, All, etc.}	separate a channel from the image into a grayscale image
Shade	geometry=>*geometry*, azimuth=>*double*, elevation=>*double*, gray=>{true, false}	shade the image using a distant light source
SetPixel	geometry=>*geometry*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray,	set the value a single pixel. Normalized pixel values are expected.

	Green, Index, Magenta, Alpha, Red, RGB, Yellow}, color=>*array of float values*, x=>*integer*, y=>*integer*, color=>*array of float values*	
SetPixels	geometry=>*geometry*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, color=>*array of float values*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*, color=>*array of float values*	set the value of one or more pixels. Normalized pixel values are expected.
Shadow	geometry=>*geometry*, opacity=>*double*, sigma=>*double*, x=>*integer*, y=>*integer*	simulate an image shadow
Sharpen	geometry=>*geometry*, radius=>*double*, sigma=>*double*, bias=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	sharpen the image with a Gaussian operator of the given radius and standard deviation (sigma).
Shave	geometry=>*geometry*, width=>*integer*, height=>*integer*	shave pixels from the image edges
Shear	geometry=>*geometry*, x=>*double*, y=>*double* fill=>*color name*	shear the image along the X or Y axis by a positive or negative shear angle
SigmoidalContrast	geometry=>*string*, 'contrast'=>*double*, 'mid-point'=>*double* channel=>{Red, RGB, All, etc.}, sharpen=>{True, False}	sigmoidal non-lineraity contrast control. Increase the contrast of the image using a sigmoidal transfer function without saturating highlights or shadows. *Contrast* indicates how much to increase the contrast (0 is none; 3 is typical; 20 is a lot); *mid-point* indicates where midtones fall in the resultant image (0 is white; 50% is middle-gray; 100% is black). To decrease contrast, set sharpen to False.
Signature		generate an SHA-256 message digest for the image pixel stream
Sketch	geometry=>*geometry*, radius=>*double*, sigma=>*double*, angle=>*double*	sketch the image with a Gaussian operator of the given radius and standard

		deviation (sigma) at the given angle
Solarize	geometry=>*string*, threshold=>*double*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	negate all pixels above the threshold level
SparseColor	points=>*array of float values*, method=>{Barycentric, Bilinear, Shepards, Voronoi}, 'virtual-pixel'=>{Background Black Constant Dither Edge Gray Mirror Random Tile Transparent White}	interpolate the image colors around the supplied points
Splice	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*, fuzz=>*double*, background=>*color name*, gravity=>{NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}	splice an image
Spread	radius=>*double*, interpolate=>{undefined, average, bicubic, bilinear, mesh, nearest-neighbor, spline}	displace image pixels by a random amount
Statistic	geometry=>*geometry*, width=>*integer*, height=>*integer*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}, type=>{Median, Mode, Mean, Maximum, Minimum, ReduceNoise, RMS}	replace each pixel with corresponding statistic from the neighborhood.
Stegano	image=>*image-handle*, offset=>*integer*	hide a digital watermark within the image
Stereo	image=>*image-handle*, x=>*integer*, y=>*integer*	composites two images and produces a single image that is the composite of a left and right image of a stereo pair
Strip		strip an image of all profiles and comments.
Swirl	degrees=>*double*, interpolate=>{undefined, average, bicubic, bilinear, mesh, nearest-neighbor, spline}	swirl image pixels about the center
Texture	texture=>*image-handle*	name of texture to tile onto the image background
Thumbnail	geometry=>*geometry*, width=>*integer*, height=>*integer*	changes the size of an image to the given dimensions and removes

		any associated profiles.
Threshold	threshold=>*string*, channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	threshold the image
Tint	fill=>*color name*, blend=>*string*	tint the image with the fill color.
Transparent	color=>*color name*, invert=>{True, False}	make this color transparent within the image
Transpose		flip image in the vertical direction and rotate 90 degrees
Transverse		flop image in the horizontal direction and rotate 270 degrees
Trim		remove edges that are the background color from the image
UnsharpMask	geometry=>*geometry*, radius=>*double*, sigma=>*double*, gain=>*double*, threshold=>*double*	sharpen the image with the unsharp mask algorithm.
Vignette	geometry=>*geometry*, radius=>*double*, sigma=>*double*, x=>*integer*, y=>*integer*, background=>*color name*	offset the edges of the image in vignette style
Wave	geometry=>*geometry*, amplitude=>*double*, wavelength=>*double*, interpolate=>{undefined, average, bicubic, bilinear, mesh, nearest-neighbor, spline}	alter an image along a sine wave
WaveDenoise	geometry=>*geometry*, threshold=>*double*, threshold=>*double*	removes noise from the image using a wavelet transform
WhiteBalance		applies white balancing to an image according to a grayworld assumption in the LAB colorspace.
WhiteThreshold	threshold=>*string*, , channel=>{All, Default, Alpha, Black, Blue, CMYK, Cyan, Gray, Green, Index, Magenta, Alpha, Red, RGB, Yellow}	force all pixels above the threshold intensity into white

Note, that the `geometry` parameter is a short cut for the `width` and `height` parameters (e.g. `geometry=>'106x80'` is equivalent to `width=>106, height=>80`).

You can specify `@filename` in both Annotate() and Draw(). This reads the text or graphic primitive instructions from a file on disk. For example,

```
image->Draw(fill=>'red', primitive=>'rectangle',
 points=>'20,20 100,100  40,40 200,200  60,60 300,300');
```

Is equivalent to

```
$image->Draw(fill=>'red', primitive=>'@draw.txt');
```

Where `draw.txt` is a file on disk that contains this:

```
rectangle 20, 20 100, 100
rectangle 40, 40 200, 200
rectangle 60, 60 300, 300
```

The *text* parameter for methods, Annotate(), Comment(), Draw(), and Label() can include the image filename, type, width, height, or other image attribute by embedding these special format characters:

```
%b   file size
%c   comment
%d   directory
%e   filename extension
%f   filename
%g   page geometry
%h   height
%i   input filename
%k   number of unique colors
%l   label
%m   magick
%n   number of scenes
%o   output filename
%p   page number
%q   quantum depth
%r   image class and colorspace
%s   scene number
%t   top of filename
%u   unique temporary filename
%w   width
%x   x resolution
%y   y resolution
%z   image depth
%C   image compression type
%D   image dispose method
%H   page height
%Q   image compression quality
%T   image delay
%W   page width
%X   page x offset
%Y   page y offset
%@   bounding box
%#   signature
%%   a percent sign
\n   newline
\r   carriage return
```

For example,

```
text=>"%m:%f %wx%h"
```

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

You can optionally add *Image* to any method name. For example, TrimImage() is an alias for method Trim().

Most of the attributes listed above have an analog in magick. See the documentation for a more detailed description of these attributes.

Set an Image Attribute

Use method Set() to set an image attribute. For example,

```
$image->Set(dither=>'True');
$image->[$x]->Set(delay=>3);
```

Where this example uses 'True' and this document says '{True, False}', you can use the case-insensitive strings 'True' and 'False', or you can use the integers 1 and 0.

When you call Get() on a Boolean attribute, Image::Magick returns 1 or 0, not a string.

And here is a list of all the image attributes you can set:

Image Attributes

Attribute	Values	Description
adjoin	{True, False}	join images into a single multi-image file
alpha	{On, Off, Opaque, Transparent, Copy, Extract, Set}	control of and special operations involving the alpha/matte channel
antialias	{True, False}	remove pixel aliasing
area-limit	*integer*	set pixel area resource limit.
attenuate	*double*	lessen (or intensify) when adding noise to an image.
authenticate	*string*	decrypt image with this password.
background	*color name*	image background color
blue-primary	*x-value, y-value*	chromaticity blue primary point (e.g. 0.15, 0.06)
bordercolor	*color name*	set the image border color
clip-mask	*image*	associate a clip mask with the image.
colormap[*i*]	*color name*	color name (e.g. red) or hex value (e.g. #ccc) at position *i*
comment	*string*	set the image comment
compression	{None, BZip, Fax, Group4, JPEG, JPEG2000, LosslessJPEG, LZW, RLE, Zip}	type of image compression
debug	{All, Annotate, Blob, Cache, Coder, Configure, Deprecate, Draw, Exception, Locale, None, Resource, Transform, X11}	display copious debugging information
delay	*integer*	this many 1/100ths of a second must expire before displaying the next image in a sequence
density	*geometry*	vertical and horizontal resolution in pixels of the image
depth	*integer*	image depth
direction	*{Undefined, right-to-left, left-to-right*	render text right-to-left or left-to-

		right
disk-limit	*integer*	set disk resource limit
dispose	*{Undefined, None, Background, Previous}*	layer disposal method
dither	{True, False}	apply error diffusion to the image
display	*string*	specifies the X server to contact
extract	*geometry*	extract area from image
file	*filehandle*	set the image filehandle
filename	*string*	set the image filename
fill	*color*	The fill color paints any areas inside the outline of drawn shape.
font	*string*	use this font when annotating the image with text
fuzz	*integer*	colors within this distance are considered equal
gamma	*double*	gamma level of the image
Gravity	{Forget, NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast}	type of image gravity
green-primary	*x-value*, *y-value*	chromaticity green primary point (e.g. 0.3, 0.6)
index[*x*, *y*]	*string*	colormap index at position (*x*, *y*)
interlace	{None, Line, Plane, Partition, JPEG, GIF, PNG}	the type of interlacing scheme
iterations	*integer*	add Netscape loop extension to your GIF animation
label	*string*	set the image label
loop	*integer*	add Netscape loop extension to your GIF animation
magick	*string*	set the image format
map-limit	*integer*	set map resource limit
mask	*image*	associate a mask with the image.
matte	{True, False}	enable the image matte channel
mattecolor	*color name*	set the image matte color
memory-limit	*integer*	set memory resource limit
monochrome	{True, False}	transform the image to black and white
option	*string*	associate an option with an image format (e.g. option=>'ps:imagemask'
orientation	{top-left, top-right, bottom-right, bottom-left, left-top, right-top, right-bottom, left-bottom}	image orientation
page	{ Letter, Tabloid, Ledger, Legal, Statement, Executive, A3, A4, A5, B4, B5, Folio, Quarto, 10x14} or *geometry*	preferred size and location of an image canvas
pixel[*x*, *y*]	*string*	hex value (e.g. #ccc) at position (*x*, *y*)
pointsize	*integer*	pointsize of the Postscript or

		TrueType font
precision	*integer*	set the maximum number of significant digits to be printed
quality	*integer*	JPEG/MIFF/PNG compression level
red-primary	*x-value*, *y-value*	chromaticity red primary point (e.g. 0.64, 0.33)
sampling-factor	*geometry*	horizontal and vertical sampling factor
scene	*integer*	image scene number
server	*string*	specifies the X server to contact
size	*string*	width and height of a raw image
stroke	*color*	The stroke color paints along the outline of a shape.
texture	*string*	name of texture to tile onto the image background
tile-offset	*geometry*	image tile offset
time-limit	*integer*	set time resource limit in seconds
type	{Bilevel, Grayscale, GrayscaleMatte, Palette, PaletteMatte, TrueColor, TrueColorMatte, ColorSeparation, ColorSeparationMatte}	image type
units	{ Undefined, PixelsPerInch, PixelsPerCentimeter}	units of image resolution
verbose	{True, False}	print detailed information about the image
virtual-pixel	{Background Black Constant Dither Edge Gray Mirror Random Tile Transparent White}	the virtual pixel method
white-point	*x-value*, *y-value*	chromaticity white point (e.g. 0.3127, 0.329)

Note, that the `geometry` parameter is a short cut for the `width` and `height` parameters (e.g. `geometry=>'106x80'` is equivalent to `width=>106, height=>80`).

SetAttribute() is an alias for method Set().

Most of the attributes listed above have an analog in magick. See the documentation for a more detailed description of these attributes.

Get an Image Attribute

Use method Get() to get an image attribute. For example,

```
($a, $b, $c) = $image->Get('colorspace', 'magick', 'adjoin');
$width = $image->[3]->Get('columns');
```

In addition to all the attributes listed in Set an Image Attribute , you can get these additional attributes:

Image Attributes

Attribute	Values	Description

area	*integer*	current area resource consumed
base-columns	*integer*	base image width (before transformations)
base-filename	*string*	base image filename (before transformations)
base-rows	*integer*	base image height (before transformations)
class	{Direct, Pseudo}	image class
colors	*integer*	number of unique colors in the image
columns	*integer*	image width
copyright	*string*	get PerlMagick's copyright
directory	*string*	tile names from within an image montage
elapsed-time	*double*	elapsed time in seconds since the image was created
error	*double*	the mean error per pixel computed with methods Compare() or Quantize()
bounding-box	*string*	image bounding box
disk	*integer*	current disk resource consumed
filesize	*integer*	number of bytes of the image on disk
format	*string*	get the descriptive image format
geometry	*string*	image geometry
height	*integer*	the number of rows or height of an image
icc	*string*	ICC profile
icc	*string*	ICM profile
id	*integer*	ImageMagick registry id
IPTC	*string*	IPTC profile
mean-error	*double*	the normalized mean error per pixel computed with methods Compare() or Quantize()
map	*integer*	current memory-mapped resource consumed
matte	{True, False}	whether or not the image has a matte channel
maximum-error	*double*	the normalized max error per pixel computed with methods Compare() or Quantize()
memory	*integer*	current memory resource consumed
mime	*string*	MIME of the image format
montage	*geometry*	tile size and offset within an image montage
page.x	*integer*	x offset of image virtual canvas
page.y	*integer*	y offset of image virtual canvas
rows	*integer*	the number of rows or height of an image
signature	*string*	SHA-256 message digest associated with the image pixel stream
taint	{True, False}	True if the image has been modified
total-ink-density	*double*	returns the total ink density for a CMYK image
transparent-color	*color name*	set the image transparent color
user-time	*double*	user time in seconds since the image was created
version	*string*	get PerlMagick's version
width	*integer*	the number of columns or width of an image
XMP	*string*	XMP profile
x-resolution	*integer*	x resolution of the image

y-resolution	*integer*	y resolution of the image

GetAttribute() is an alias for method Get().

Most of the attributes listed above have an analog in magick. See the documentation for a more detailed description of these attributes.

Compare an Image to its Reconstruction

Mathematically and visually annotate the difference between an image and its reconstruction with the Compare() method. The method supports these parameters:

Compare Parameters

Parameter	Values	Description
channel	*double*	select image channels, the default is all channels except alpha.
fuzz	*double*	colors within this distance are considered equal
image	*image-reference*	the image reconstruction
metric	AE, MAE, MEPP, MSE, PAE, PSNR, RMSE	measure differences between images with this metric

In this example, we compare the ImageMagick logo to a sharpened reconstruction:

```
use Image::Magick;

$logo=Image::Magick->New();
$logo->Read('logo:');
$sharp=Image::Magick->New();
$sharp->Read('logo:');
$sharp->Sharpen('0x1');
$difference=$logo->Compare(image=>$sharp, metric=>'rmse');
print $difference->Get('error'), "\n";
$difference->Display();
```

In addition to the reported root mean squared error of around 0.024, a difference image is displayed so you can visually identify the difference between the images.

Create an Image Montage

Use method Montage() to create a composite image by combining several separate images. The images are tiled on the composite image with the name of the image optionally appearing just below the individual tile. For example,

```
$image->Montage(geometry=>'160x160', tile=>'2x2', texture=>'granite:');
```

And here is a list of Montage() parameters you can set:

Montage Parameters

Parameter	Values	Description
background	*color name*	background color name
border	*integer*	image border width
filename	*string*	name of montage image
fill	color name	fill color for annotations

font	*string*	X11 font name
frame	*geometry*	surround the image with an ornamental border
geometry	*geometry*	preferred tile and border size of each tile of the composite image (e.g. 120x120+4+3>)
gravity	NorthWest, North, NorthEast, West, Center, East, SouthWest, South, SouthEast	direction image gravitates to within a tile
label	*string*	assign a label to an image
mode	Frame, Unframe, Concatenate	thumbnail framing options
pointsize	*integer*	pointsize of the Postscript or TrueType font
shadow	{True, False}	add a shadow beneath a tile to simulate depth
stroke	color name	stroke color for annotations
texture	*string*	name of texture to tile onto the image background
tile	*geometry*	the number of tiles per row and page (e.g. 6x4)
title	string	assign a title to the image montage
transparent	*string*	make this color transparent within the image

Note, that the `geometry` parameter is a short cut for the `width` and `height` parameters (e.g. `geometry=>'106x80'` is equivalent to `width=>106, height=>80`).

MontageImage() is an alias for method Montage().

Most of the attributes listed above have an analog in montage. See the documentation for a more detailed description of these attributes.

Working with Blobs

A blob contains data that directly represent a particular image format in memory instead of on disk. PerlMagick supports blobs in any of these image formats and provides methods to convert a blob to or from a particular image format.

Blob Methods

Method	Parameters	Return Value	Description
ImageToBlob	any image attribute	an array of image data in the respective image format	convert an image or image sequence to an array of blobs
BlobToImage	one or more blobs	the number of blobs converted to an image	convert one or more blobs to an image

ImageToBlob() returns the image data in their respective formats. You can then print it, save it to an ODBC database, write it to a file, or pipe it to a display program:

```
@blobs = $image->ImageToBlob();
open(DISPLAY,"| display -") || die;
binmode DISPLAY;
print DISPLAY $blobs[0];
close DISPLAY;
```

Method BlobToImage() returns an image or image sequence converted from the supplied blob:

```
@blob=$db->GetImage();
$image=Image::Magick->new(magick=>'jpg');
$image->BlobToImage(@blob);
```

Direct-access to Image Pixels

Use these methods to obtain direct access to the image pixels:

Direct-access to Image Pixels

Method	Parameters	Description
GetAuthenticPixels	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*	return authentic pixels as a C pointer
GetVirtualPixels	geometry=>*geometry*, width=>*integer*, height=>*integer*, x=>*integer*, y=>*integer*	return virtual pixels as a const C pointer
GetAuthenticIndexQueue		return colormap indexes or black pixels as a C pointer
GetVirtualIndexQueue		return colormap indexes or black pixels as a const C pointer
SyncAuthenticPixels		sync authentic pixels to pixel cache

Miscellaneous Methods

The Append() method append a set of images. For example,

```
$p = $image->Append(stack=>{true,false});
```

appends all the images associated with object `$image`. By default, images are stacked left-to-right. Set `stack` to True to stack them top-to-bottom.

The Clone() method copies a set of images. For example,

```
$q = $p->Clone();
```

copies all the images from object `$p` to `$q`. You can use this method for single or multi-image sequences.

Coalesce() 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. A new image sequence is returned with all images the same size as the first images virtual canvas and composited with the next image in the sequence.. For example,

```
$q = $p->Coalesce();
```

The ComplexImages() method performs complex mathematics on an image sequence. For example,

```
$p = $image->ComplexImages('conjugate');
```

The EvaluateImages() method applies an arithmetic, logical or relational expression to a set of images. For example,

```
$p = $image->EvaluateImages('mean');
```

averages all the images associated with object `$image`.

The Features() method returns features for each channel in the image in each of four directions (horizontal, vertical, left and right diagonals) for the specified distance. The features include the angular second momentum, contrast, correlation, sum of squares: variance, inverse difference moment, sum average, sum varience, sum entropy, entropy, difference variance, difference entropy, information measures of correlation 1, information measures of correlation 2, and maximum correlation coefficient. Values in RGB, CMYK, RGBA, or CMYKA order (depending on the image type).

```
@features = $image->Features(1);
```

The Flatten() method flattens a set of images and returns it. For example,

```
$p = $images->Flatten(background=>'none');
$p->Write('flatten.png');
```

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

The Fx() method applies a mathematical expression to a set of images and returns the results. For example,

```
$p = $image->Fx(expression=>'(g+b)/2.0',channel=>'red');
$p->Write('fx.miff');
```

replaces the red channel with the average of the green and blue channels.

See FX, The Special Effects Image Operator for a detailed discussion of this method.

Histogram() returns the unique colors in the image and a count for each one. The returned values are an array of red, green, blue, opacity, and count values.

The Morph() method morphs a set of images. Both the image pixels and size are linearly interpolated to give the appearance of a meta-morphosis from one image to the next:

```
$p = $image->Morph(frames=>integer);
```

where *frames* is the number of in-between images to generate. The default is 1.

Mosaic() creates an mosaic from an image sequence.

Method Mogrify() is a single entry point for the image manipulation methods (Manipulate an Image). The parameters are the name of a method followed by any parameters the method may require. For example, these calls are equivalent:

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

Method MogrifyRegion() applies a transform to a region of the image. It is similar to Mogrify() but begins with the region geometry. For example, suppose you want to brighten a 100x100 region of your image at location (40, 50):

```
$image->MogrifyRegion('100x100+40+50', 'modulate', brightness=>50);
```

PerceptualHash() maps visually identical images to the same or similar hash-- useful in image retrieval, authentication, indexing, or copy detection as well as digital watermarking. For each channel and for the sRGB and the HCLp colorspaces, 7 hash values are returned For an sRGB images, for example, expect 42 perceptual hashes.

```
@phash = $image->PerceptualHash();
```

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

This a more efficient and less memory intensive way to query if an image exists and what its characteristics are.

Poly() builds a polynomial from the image sequence and the corresponding terms (coefficients and degree pairs):

```
$p = $image->Poly([0.5,1.0,0.25,2.0,1.0,1.0]);
```

PreviewImage() tiles 9 thumbnails of the specified image with an image processing operation applied at varying strengths. This may be helpful pin-pointing an appropriate parameter for a particular image processing operation. Choose from these operations: `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`. Here is an example:

```
$preview = $image->Preview('Gamma');
$preview->Display();
```

To have full control over text positioning you need font metric information. Use

```
($x_ppem, $y_ppem, $ascender, $descender, $width, $height, $max_advance) =
  $image->QueryFontMetrics(parameters);
```

Where *parameters* is any parameter of the Annotate method. The return values are:

1. character width
2. character height
3. ascender
4. descender

5. text width
6. text height
7. maximum horizontal advance
8. bounds: x1
9. bounds: y1
10. bounds: x2
11. bounds: y2
12. origin: x
13. origin: y

Use QueryMultilineFontMetrics() to get the maximum text width and height for multiple lines of text.

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) = $image->QueryColor('cyan');
($red, $green, $blue, $alpha) = $image->QueryColor('#716baeff');
```

QueryColorname() accepts a color value and returns its respective name or hex value;

```
$name = $image->QueryColorname('rgba(80,60,0,0)');
```

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];
```

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, decoder, encoder, description, and module.

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

Call MagickToMime() with the image format name to get its MIME type such as `image/tiff` from `tif`.

```
$mime = $image->MagickToMime('tif');
```

Use RemoteCommand() to send a command to an already running display or animate application. The only parameter is the name of the image file to display or animate.

```
$image->RemoteCommand('image.jpg');
```

The Smush() method smushes a set of images together. For example,

```
$p = $image->Smush(stack=>{true,false},offset=>integer);
```

smushes together all the images associated with object `$image`. By default, images are smushed left-to-right. Set `stack` to True to smushed them top-to-bottom.

Statistics() returns the image statistics for each channel in the image. The returned values are an array of depth, minima, maxima, mean, standard deviation, kurtosis, skewness, and entropy values in RGB, CMYK, RGBA, or CMYKA order (depending on the image type).

```
@statistics = $image->Statistics();
```

Finally, the Transform() method accepts a fully-qualified geometry specification for cropping or resizing one or more images. For example,

```
$p = $image->Transform(crop=>'100x100+0+0');
```

You can optionally add *Image* to any method name above. For example, PingImage() is an alias for method Ping().

Handling Exceptions

All PerlMagick methods return an undefined string context upon success. If any problems occur, the error is returned as a string with an embedded numeric status code. A status code less than 400 is a warning. This means that the operation did not complete but was recoverable to some degree. A numeric code greater or equal to 400 is an error and indicates the operation failed completely. Here is how exceptions are returned for the different methods:

Methods which return a number (e.g. Read(), Write()):

```
$x = $image->Read(...);
warn "$x" if "$x";      # print the error message
$x =~ /(\d+)/;
print $1;               # print the error number
print 0+$x;             # print the number of images read
```

Methods which operate on an image (e.g. Resize(), Crop()):

```
$x = $image->Crop(...);
warn "$x" if "$x";      # print the error message
$x =~ /(\d+)/;
print $1;               # print the error number
```

Methods which return images (EvaluateSequence(), Montage(), Clone()) should be checked for errors this way:

```
$x = $image->Montage(...);
warn "$x" if !ref($x);  # print the error message
$x =~ /(\d+)/;
print $1;               # print the error number
```

Here is an example error message:

```
Error 400: Memory allocation failed
```

Review the complete list of error and warning codes.

The following illustrates how you can use a numeric status code:

```
$x = $image->Read('rose.png');
$x =~ /(\d+)/;
die "unable to continue" if ($1 == ResourceLimitError);
```

Constants

PerlMagick includes these constants:

```
BlobError
BlobWarning
CacheError
CacheWarning
CoderError
CoderWarning
ConfigureError
ConfigureWarning
CorruptImageError
CorruptImageWarning
DelegateError
DelegateWarning
DrawError
DrawWarning
ErrorException
FatalErrorException
FileOpenError
FileOpenWarning
ImageError
ImageWarning
MissingDelegateError
MissingDelegateWarning
ModuleError
ModuleWarning
Opaque
OptionError
OptionWarning
QuantumDepth
QuantumRange
RegistryError
RegistryWarning
ResourceLimitError
ResourceLimitWarning
StreamError
StreamWarning
Success
Transparent
TypeError
TypeWarning
WarningException
XServerError
XServerWarning
```

You can access them like this:

```
Image::Magick->QuantumDepth
```