Table of Contents

Features and Capabilities • News • Community

Use ImageMagick® to create, edit, compose, or convert bitmap images. It can read and write images in a variety of formats (over 200) including PNG, JPEG, GIF, HEIC, TIFF, DPX, EXR, WebP, Postscript, PDF, and SVG. Use ImageMagick to resize, flip, mirror, rotate, distort, shear and transform images, adjust image colors, apply various special effects, or draw text, lines, polygons, ellipses and Bézier curves.

ImageMagick is free software delivered as a ready-to-run binary distribution or as source code that you may use, copy, modify, and distribute in both open and proprietary applications. It is distributed under a derived Apache 2.0 license.

ImageMagick utilizes multiple computational threads to increase performance and can read, process, or write mega-, giga-, or tera-pixel image sizes.

The current release is ImageMagick 7.0.8-33. It runs on Linux, Windows, Mac Os X, iOS, Android OS, and others.

The authoritative ImageMagick web site is https://imagemagick.org. The authoritative source code repository is https://github.com/ImageMagick. We maintain a source code mirror at https://gitlab.com/ImageMagick. We continue to maintain the legacy release of ImageMagick, version 6, at https://legacy.imagemagick.org.

Features and Capabilities

Here are just a few examples of what ImageMagick can do for you:

Animation	create a GIF animation sequence from a group of images.
Color management	accurate color management with color profiles or in lieu of-- built-in gamma compression or expansion as demanded by the colorspace.
Command-line processing	utilize ImageMagick from the command-line.
Complex text layout	bidirectional text support and shaping.
Composite	overlap one image over another.
Connected component	uniquely label connected regions in an image.

labeling	
Decorate	add a border or frame to an image.
Delineate image features	Canny edge detection, Hough lines.
Discrete Fourier transform	implements the forward and inverse DFT.
Distributed pixel cache	offload intermediate pixel storage to one or more remote servers.
Draw	add shapes or text to an image.
Encipher or decipher an image	convert ordinary images into unintelligible gibberish and back again.
Format conversion	convert an image from one format to another (e.g. PNG to JPEG).
Generalized pixel distortion	correct for, or induce image distortions including perspective.
Heterogeneous distributed processing	certain algorithms are OpenCL-enabled to take advantage of speed-ups offered by executing in concert across heterogeneous platforms consisting of CPUs, GPUs, and other processors.
High dynamic-range images	accurately represent the wide range of intensity levels found in real scenes ranging from the brightest direct sunlight to the deepest darkest shadows.
Histogram Equalization	Use adaptive histogram equalization to improve contrast in images.
Image calculator	apply a mathematical expression to an image or image channels.
Image gradients	create a gradual blend of two colors whose shape is horizontal, vertical, circular, or elliptical.
Image identification	describe the format and attributes of an image.
ImageMagick on the iPhone	convert, edit, or compose images on your iOS device such as the iPhone or iPad.
Large image support	read, process, or write mega-, giga-, or tera-pixel image sizes.
Montage	juxtapose image thumbnails on an image canvas.
Morphology of shapes	extract features, describe shapes, and recognize patterns in images.
Motion picture support	read and write the common image formats used in digital film work.
Noise and color reduction	Kuwahara Filter, mean-shift.
Perceptual hash	map visually identical images to the same or similar hash-- useful in image retrieval, authentication, indexing, or copy detection as well as digital watermarking.
Special effects	blur, sharpen, threshold, or tint an image.
Text & comments	insert descriptive or artistic text in an image.
Threads of execution support	ImageMagick is thread safe and most internal algorithms execute in parallel to take advantage of speed-ups offered by multicore processor chips.

Transform	resize, rotate, deskew, crop, flip or trim an image.
Transparency	render portions of an image invisible.
Virtual pixel support	convenient access to pixels outside the image boundaries.

Examples of ImageMagick Usage shows how to use ImageMagick from the command-line to accomplish any of these tasks and much more. Also, see Fred's ImageMagick Scripts: a plethora of command-line scripts that perform geometric transforms, blurs, sharpens, edging, noise removal, and color manipulations. With Magick.NET, use ImageMagick without having to install ImageMagick on your server or desktop.

News

Now that ImageMagick version 7 is released, we continue to maintain the legacy release of ImageMagick, version 6, at https://legacy.imagemagick.org. Learn how ImageMagick version 7 differs from previous versions with our porting guide.

ImageMagick best practices strongly encourages you to configure a security policy that suits your local environment.

The ImageMagick development process ensures a stable API and ABI. Before each ImageMagick release, we perform a comprehensive security assessment that includes memory error, thread data race detection, and continuous fuzzing to help prevent security vulnerabilities.

As an analog to linear (RGB) and non-linear (sRGB) color colorspaces, as of ImageMagick 7.0.7-17, we introduce the LinearGray colorspace. Gray is non-linear grayscale and LinearGray is linear (e.g. -colorspace linear-gray).

Want more performance from ImageMagick? Try these options:

- Add more memory to your system, see the pixel cache;
- Add more cores to your system, see threads of execution support;
- push large images to a solid-state drive, see large image support.

</ul>

If these options are prohibitive, you can reduce the quality of the image results. The default build is Q16 HDRI. If you disable HDRI, you use half the memory and instead of predominately floating point operations, you use the typically more efficient integer operations. The tradeoff is reduced precision and you cannot process out of range pixel values (e.g. negative). If you build the Q8 non-HDRI version of ImageMagick, you again reduce the memory requirements in half-- and once again there is a tradeoff, even less precision and no out of range pixel values. For a Q8 non-HDRI build of ImageMagick, use these `configure` script options: `--with-quantum-depth=8 --disable-hdri`.

Community

To join the ImageMagick community, try the discourse server. You can review questions or comments (with informed responses) posed by ImageMagick users or ask your own questions. If you want to contribute image processing algorithms, other enhancements, or bug fixes, open an issue.

</div>

Unix Binary Release • Mac OS X Binary Release • iOS Binary Release • Windows Binary Release

You can install ImageMagick from source. However, if you don't have a proper development environment or if you're anxious to get started, download a ready-to-run Unix or Windows executable. Before you download, you may want to review recent changes to the ImageMagick distribution.

ImageMagick source and binary distributions are available from a variety of FTP and Web mirrors around the world.

Unix Binary Release

These are the Unix variations that we support. If your system is not on the list, try installing from source. Although ImageMagick runs fine on a single core computer, it automagically runs in parallel on dual and quad-core systems reducing run times considerably.

Version	HTTP	FTP	Description
ImageMagick-7.0.8-33.x86_64.rpm	download	download	Redhat / CentOS 7.1 x86_64 RPM
ImageMagick-libs-7.0.8-33.x86_64.rpm	download	download	Redhat / CentOS 7.1 x86_64 RPM
ImageMagick RPM's	download	download	Development, Perl, C++, and documentation RPM's.
ImageMagick-i386-pc-solaris2.11.tar.gz	download	download	Solaris Sparc 2.11
ImageMagick-i686-pc-cygwin.tar.gz	download	download	Cygwin
ImageMagick-i686-pc-mingw32.tar.gz	download	download	MinGW

Verify its message digest.

ImageMagick RPM's are self-installing. Simply type the following command and you're ready to start using ImageMagick:

```
$ rpm -Uvh ImageMagick-7.0.8-33.x86_64.rpm
```

You'll need the libraries as well:

```
$ rpm -Uvh ImageMagick-libs-7.0.8-33.x86_64.rpm
```

For other systems, create (or choose) a directory to install the package into and change to that directory, for example:

```
cd $HOME
```

Next, extract the contents of the package. For example:

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

Set the `MAGICK_HOME` environment variable to the path where you extracted the ImageMagick files. For example:

```
$ export MAGICK_HOME="$HOME/ImageMagick-7.0.8"
```

If the `bin` subdirectory of the extracted package is not already in your executable search path, add it to your `PATH` environment variable. For example:

```
export PATH="$MAGICK_HOME/bin:$PATH
```

On Linux and Solaris machines add `$MAGICK_HOME/lib` to the `LD_LIBRARY_PATH` environment variable:

```
LD_LIBRARY_PATH="${LD_LIBRARY_PATH:+$LD_LIBRARY_PATH:}$MAGICK_HOME/lib
export LD_LIBRARY_PATH
```

Finally, to verify ImageMagick is working properly, type the following on the command line:

```
magick logo: logo.gif
identify logo.gif
display logo.gif
```

Congratulations, you have a working ImageMagick distribution under Unix or Linux and you are ready to use ImageMagick to convert, compose, or edit your images or perhaps you'll want to use one of the Application Program Interfaces for C, C++, Perl, and others.

Mac OS X Binary Release

We recommend MacPorts which custom builds ImageMagick in your environment (some users prefer Homebrew). Download MacPorts and type:

```
sudo port install ImageMagick
```

The `port` command downloads ImageMagick and many of its delegate libraries (e.g. JPEG, PNG, Freetype, etc.) and configures, builds, and installs ImageMagick automagically. Alternatively, you can download the ImageMagick Mac OS X distribution we provide:

Version	HTTP	FTP	Description
ImageMagick-x86_64-apple-darwin17.7.0.tar.gz	download	download	macOS High Sierra

Verify its message digest.

Create (or choose) a directory to install the package into and change to that directory, for example:

```
cd $HOME
```

Next, extract the contents of the package. For example:

```
tar xvzf ImageMagick-x86_64-apple-darwin17.2.0.tar.gz
```

Set the `MAGICK_HOME` environment variable to the path where you extracted the ImageMagick files. For example:

```
$ export MAGICK_HOME="$HOME/ImageMagick-7.0.8"
```

If the `bin` subdirectory of the extracted package is not already in your executable search path, add it to your `PATH` environment variable. For example:

```
export PATH="$MAGICK_HOME/bin:$PATH"
```

Set the `DYLD_LIBRARY_PATH` environment variable:

```
export DYLD_LIBRARY_PATH="$MAGICK_HOME/lib/"
```

Finally, to verify ImageMagick is working properly, type the following on the command line:

```
magick logo: logo.gif
identify logo.gif
display logo.gif
```

Note, the display program requires the X11 server available on your Mac OS X installation DVD. Once that is installed, you will also need to set `export DISPLAY=:0`.

The best way to deal with all the exports is to put them at the end of your .profile file

Congratulations, you have a working ImageMagick distribution under Mac OS X and you are ready to use ImageMagick to convert, compose, or edit your images or perhaps you'll want to use one of the Application Program Interfaces for C, C++, Perl, and others.

iOS Binary Release

~Claudio provides iOS builds of ImageMagick.

Download iOS Distribution

You can download the iOS distribution directly from ImageMagick's repository.

There are always 2 packages for the compiled ImageMagick:

- iOSMagick-VERSION-libs.zip
- iOSMagick-VERSION.zip

The first one includes headers and compiled libraries that have been used to compile ImageMagick. Most users would need this one.

ImageMagick compiling script for iOS OS and iOS Simulator

To run the script:

```
./imagemagick_compile.sh VERSION
```

where *VERSION* is the version of ImageMagick you want to compile (i.e.: 7.0.8-33, svn, ...)

This script compiles ImageMagick as a static library to be included in iOS projects and adds support for

- png
- jpeg
- tiff

Upon successful compilation a folder called `IMPORT_ME` is created on your `~/Desktop`. You can import it into your Xcode project.

Xcode project settings

After including everything into Xcode please also make sure to have these settings (Build tab of the project information):

- Other Linker Flags: -lMagickCore-Q16 -lMagickWand-Q16 -ljpeg -lpng -lbz2 -lz
- Header Search Paths: $(SRCROOT) - make it Recursive
- Library Search Paths: $(SRCROOT) - make it Recursive

On the lower left click on the small-wheel and select: Add User-Defined Setting

- Key: OTHER_CFLAGS
- Value: -Dmacintosh=1

Sample project

A sample project is available for download. It is not updated too often, but it does give an idea of all the settings and some ways to play around with ImageMagick in an iOS application.

Windows Binary Release

ImageMagick runs on Windows 10 (x86 & x64), Windows 8 (x86 & x64), Windows 7 (x86 & x64), Windows Server 2012, Windows XP (x86) with Service Pack 3, Windows Vista (x86 & x64) with Service Pack 2, Windows Server 2003 (x86 & x64) with Service Pack 2 (verify MSXML6 is present), Windows Server 2003 R2 (x86 & x64), Windows Server 2008 (x86 & x64) with Service Pack 2, and Windows Server 2008 R2 (x64).

The amount of memory can be an important factor, especially if you intend to work on large images. A minimum of 512 MB of RAM is recommended, but the more RAM the better. Although ImageMagick runs well on a single core computer, it automagically runs in parallel on multi-core systems reducing run times considerably.

The Windows version of ImageMagick is self-installing. Simply click on the appropriate version below and it will launch itself and ask you a few installation questions. Versions with *Q8* in the name are 8 bits-per-pixel component (e.g. 8-bit red, 8-bit green, etc.), whereas, *Q16* in the filename are 16 bits-per-pixel component. A Q16 version permits you to read or write 16-bit images without losing precision but requires twice as much resources as the Q8 version. Versions with *dll* in the filename include ImageMagick libraries as dynamic link libraries. Unless you have a Windows 32-bit OS, we recommend this version of ImageMagick for 64-bit Windows:

Version	HTTP	FTP	Description
ImageMagick-7.0.8-33-Q16-x64-dll.exe	download	download	Win64 dynamic at 16 bits-per-pixel component

Or choose from these alternate Windows binary distributions:

Version	HTTP	FTP	Description
ImageMagick-7.0.8-33-Q16-x64-static.exe	download	download	Win64 static at 16 bits-per-pixel component
ImageMagick-7.0.8-33-Q8-x64-dll.exe	download	download	Win64 dynamic at 8 bits-per-pixel component
ImageMagick-7.0.8-33-Q8-x64-static.exe	download	download	Win64 static at 8 bits-per-pixel component
ImageMagick-7.0.8-33-Q16-HDRI-x64-dll.exe	download	download	Win64 dynamic at 16 bits-per-pixel component with high dynamic-range imaging enabled
			Win64 static at 16 bits-per-pixel component with high

			dynamic-range imaging enabled
ImageMagick-7.0.8-33-Q16-x86-dll.exe	download	download	Win32 dynamic at 16 bits-per-pixel component
ImageMagick-7.0.8-33-Q16-x86-static.exe	download	download	Win32 static at 16 bits-per-pixel component
ImageMagick-7.0.8-33-Q8-x86-dll.exe	download	download	Win32 dynamic at 8 bits-per-pixel component
ImageMagick-7.0.8-33-Q8-x86-static.exe	download	download	Win32 static at 8 bits-per-pixel component
ImageMagick-7.0.8-33-Q16-HDRI-x86-dll.exe	download	download	Win32 dynamic at 16 bits-per-pixel component with high dynamic-range imaging enabled
ImageMagick-7.0.8-33-Q16-HDRI-x86-static.exe	download	download	Win32 static at 16 bits-per-pixel component with high dynamic-range imaging enabled
ImageMagick-7.0.8-33-portable-Q16-x86.zip	download	download	Portable Win32 static at 16 bits-per-pixel component. Just copy to your host and run (no installer, no Windows registry entries).
ImageMagick-7.0.8-33-portable-Q16-x64.zip	download	download	Portable Win64 static at 16 bits-per-pixel component. Just copy to your host and run (no installer, no Windows registry entries).

Verify its message digest.

To verify ImageMagick is working properly, type the following in an Command Prompt window:

```
magick logo: logo.gif
magick identify logo.gif
magick logo.gif win:
```

If you have any problems, you likely need `vcomp120.dll`. To install it, download Visual C++ 2013 Redistributable Package.

Note, use a double quote (`"`) rather than a single quote (`'`) for the ImageMagick command line under Windows:

```
magick "e:/myimages/image.png" "e:/myimages/image.jpg"
```

Use two double quotes for VBScript scripts:

```
Set objShell = wscript.createobject("wscript.shell")
objShell.Exec("magick ""e:/myimages/image.png"" ""e:/myimages/image.jpg""")
```

Congratulations, you have a working ImageMagick distribution under Windows and you are ready to use ImageMagick to convert, compose, or edit your images or perhaps you'll want to use one of the Application Program Interfaces for C, C++, Perl, and others.

</div>

ImageMagick includes a number of command-line utilities for manipulating images. Most of you are probably accustomed to editing images one at a time with a graphical user interface (GUI) with such programs as Gimp or Photoshop. However, a GUI is not always convenient. Suppose you want to process an image dynamically from a web script or you want to apply the same operations to many images or repeat a specific operation at different times to the same or different image. For these types of operations, the command-line image processing utility is appropriate.

The ImageMagick command-line tools exit with a status of 0 if the command line arguments have a proper syntax and no problems are encountered. Expect a descriptive message and an exit status of 1 if any exception occurs such as improper syntax, a problem reading or writing an image, or any other problem that prevents the command from completing successfully.

Here is a short description for each command-line tool. Click on the program name to get details about the program usage and a list of command-line options that alters how the program behaves. If you are just getting acquainted with ImageMagick, start with the magick program. Be sure to peruse Anthony Thyssen's tutorial on how to use ImageMagick utilities to create, edit, compose, or convert images from the command-line.

magick
convert between image formats as well as resize an image, blur, crop, despeckle, dither, draw on, flip, join, re-sample, and much more.

magick-script
use this scripting language interpreter to convert between image formats as well as resize an image, blur, crop, despeckle, dither, draw on, flip, join, re-sample, and much more.

We also support tools for compatibility with ImageMagick version 6:

animate
animate an image sequence on any X server.

compare
mathematically and visually annotate the difference between an image and its reconstruction.

composite
overlap one image over another.

conjure
interpret and execute scripts written in the Magick Scripting Language (MSL).

convert
convert between image formats as well as resize an image, blur, crop, despeckle, dither, draw on, flip, join, re-sample, and much more.

display
display an image or image sequence on any X server.

identify
describe the format and characteristics of one or more image files.

import
save any visible window on an X server and outputs it as an image file. You can capture a single window, the entire screen, or any rectangular portion of the screen.

mogrify
resize an image, blur, crop, despeckle, dither, draw on, flip, join, re-sample, and much more. Mogrify overwrites the original image file, whereas, convert writes to a different image file.

montage

create a composite image by combining several separate images. The images are tiled on the composite image optionally adorned with a border, frame, image name, and more.

stream

a lightweight tool to stream one or more pixel components of the image or portion of the image to your choice of storage formats. It writes the pixel components as they are read from the input image a row at a time making `stream` desirable when working with large images or when you require raw pixel components.

If these tools are not available on your computer, you can instead utilize them as a subcommand of the `magick` command. For example,

```
magick identify -verbose myImage.png
```

The Anatomy of the Command-line • Input Filename • Command-line Options • Output Filename

The ImageMagick command-line tools can be as simple as this:

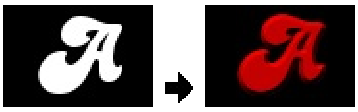
```
magick image.jpg image.png
```

Or it can be complex with a plethora of options, as in the following:

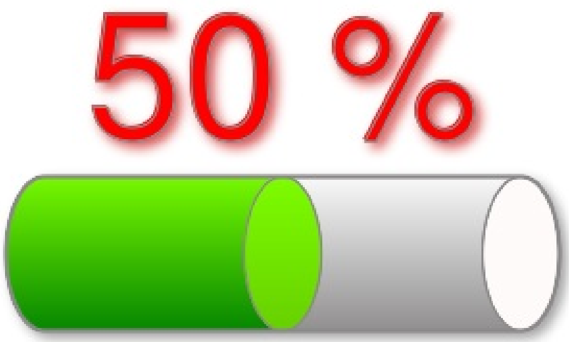
```
magick label.gif +matte \
  \( +clone  -shade 110x90 -normalize -negate +clone  -compose Plus -composite \) \
  \( -clone 0 -shade 110x50 -normalize -channel BG -fx 0 +channel -matte \) \
  -delete 0 +swap  -compose Multiply -composite  button.gif");
```

This example command is long enough that the command must be written across several lines, so we formatted it for clarity by inserting backslashes (`\`). The backslash is the Unix *line-continuation* character. In the Windows shell, use a carat character (`^`) for line-continuation. We use the Unix style on these web pages, as above. Sometimes, however, the lines are wrapped by your browser if the browser window is small enough, but the command-lines, shown in white, are still intended to be typed as one line. Line continuation characters need not be entered. The *parentheses* that are *escaped* above using the backslash are not escaped in Windows. There are some other differences between Windows and Unix (involving quotation marks, for instance), but we'll discuss some of those issues later, as they arise.

Without knowing much about the ImageMagick command-line, you can probably surmise that the first command above converts an image in the JPEG format to one in the PNG format. However, very few may realize the second, more complex command, gives a flat two-dimensional label a three-dimensional look with rich textures and simulated depth:



Here we show percent completion of a task as a shaded cylinder:



Given the complexity of the rendering, you might be surprised it is accomplished by a single command-line:

```
magick -size 320x90 canvas:none -stroke snow4 -size 1x90 -tile gradient:white-snow4 \
  -draw 'roundrectangle 16, 5, 304, 85 20,40' +tile -fill snow \
  -draw 'roundrectangle 264, 5, 304, 85  20,40' -tile gradient:chartreuse-green \
  -draw 'roundrectangle 16,  5, 180, 85  20,40' -tile gradient:chartreuse1-chartreuse3 \
  -draw 'roundrectangle 140, 5, 180, 85  20,40' +tile -fill none \
  -draw 'roundrectangle 264, 5, 304, 85 20,40' -strokewidth 2 \
  -draw 'roundrectangle 16, 5, 304, 85 20,40' \( +clone -background snow4 \
  -shadow 80x3+3+3 \) +swap -background none -layers merge \( +size -pointsize 90 \
  -strokewidth 1 -fill red label:'50 %' -trim +repage \( +clone -background firebrick3 \
  -shadow 80x3+3+3 \) +swap -background none -layers merge \) -insert 0 -gravity center \
  -append -background white -gravity center -extent 320x200 cylinder_shaded.png
```

In the next sections we dissect the anatomy of the ImageMagick command-line. Hopefully, after carefully reading and better understanding how the command-line works, you should be able to accomplish complex image-processing tasks without resorting to the sometimes daunting program interfaces.

See Examples of ImageMagick Usage for additional help when using ImageMagick from the command-line.

The Anatomy of the Command-line

The ImageMagick command-line consists of

1. one or more required input filenames.
2. zero, one, or more image settings.
3. zero, one, or more image operators.
4. zero, one, or more image sequence operators.
5. zero, one, or more image stacks.
6. zero or one output image filenames (required by convert, composite, montage, compare, import, conjure).

You can find a detailed explanation of each of the constituent parts of the command-line in the sections that follow.

Input Filename

ImageMagick extends the concept of an input filename to include:

- filename globbing
- an explicit image format
- using built-in images and patterns
- STDIN, STDOUT, and file descriptors
- selecting certain frames from an image
- selecting a region of an image
- forcing an inline image resize
- forcing an inline image crop
- using filename references

These extensions are explained in the next few paragraphs.

Filename Globbing

In Unix shells, certain characters such as the asterisk (`*`) and question mark (`?`) automagically cause lists of filenames to be generated based on pattern matches. This feature is known as globbing. ImageMagick supports filename globbing for systems, such as Windows, that does not natively support it. For example, suppose you want to convert `1.jpg` , `2.jpg` , `3.jpg` , `4.jpg` , and `5.jpg` in your current directory to a GIF animation. You can conveniently refer to all of the JPEG files with this command:

```
magick *.jpg images.gif
```

Explicit Image Format

Images are stored in a myriad of image formats including the better known JPEG, PNG, TIFF and others. ImageMagick must know the format of the image before it can be read and processed. Most formats have a signature within the image that uniquely identifies the format. Failing that, ImageMagick leverages the filename extension to determine the format. For example, `image.jpg` or `image.JPG` tells ImageMagick it is reading an image in the JPEG format.

In some cases the image may not contain a signature and/or the filename does not identify the image format. In these cases an explicit image format must be specified. For example, suppose our image is named `image` and contains raw red, green, and blue intensity values. ImageMagick has no way to automagically determine the image format so we explicitly set one:

```
magick -size 640x480 -depth 8 rgb:image image.png
```

Built-in Images and Patterns

ImageMagick has a number of built-in images and patterns. To utilize the checkerboard pattern, for example, use:

```
magick -size 640x480 pattern:checkerboard checkerboard.png
```

STDIN, STDOUT, and file descriptors

Unix and Windows permit the output of one command to be piped to the input of another. ImageMagick permits image data to be read and written from the standard streams STDIN (*standard in*) and STDOUT (*standard out*), respectively, using a pseudo-filename of `-`. In this example we pipe the output of convert to the display program:

```
magick logo: gif:- | display gif:-
```

The second explicit format "`gif:`" is optional in the preceding example. The GIF image format has a unique signature within the image so ImageMagick's display command can readily recognize the format as GIF. The convert program also accepts STDIN as input in this way:

```
magick rose: gif:- | magick - -resize "200%" bigrose.jpg'
```

Other pipes can be accessed via their *file descriptors* (as of version 6.4.9-3). The file descriptors 0, 1, and 2 are reserved for the standard streams STDIN, STDOUT, and STDERR, respectively, but a pipe associated with a file descriptor number *N*>2 can be accessed using the pseudonym `fd:` *N*. (The pseudonyms `fd:0` and `fd:1` can be used for STDIN and STDOUT.) The next example shows how to append image data piped from files with descriptors 3 and 4 and direct the result to the file with descriptor number 5.

```
magick fd:3 fd:4 -append fd:5
```

When needed, explicit image formats can be given as mentioned earlier, as in the following.

```
magick gif:fd:3 jpg:fd:4 -append tif:fd:5
```

Selecting Frames

Some images formats contain more than one image frame. Perhaps you only want the first image, or the last, or some number of images in-between. You can specify which image frames to read by appending the image filename with the frame range enclosed in brackets. Here our image (an animated GIF) contains more than one frame but we only want the first:

```
magick 'images.gif[0]' image.png
```

Unix shells generally interpret brackets so we enclosed the filename in quotes above. In a Windows command shell the brackets are not interpreted but using quotes doesn't hurt. However, in most cases the roles of single-quotes and double-quotes are reversed with respect to Unix and Windows, so Windows users should usually try double-quotes where we display single-quotes, and vice versa.

You can read more than one image from a sequence with a frame range. For example, you can extract the first four frames of an image sequence:

```
magick 'images.gif[0-3]' images.mng
```

Finally, you can read more than one image from a sequence, out-of-order. The next command gets the third image in the sequence, followed by the second, and then the fourth:

```
magick 'images.gif[3,2,4]' images.mng
```

Notice that in the last two commands, a single image is written. The output in this case, where the image type is MNG, is a multi-frame file because the MNG format supports multiple frames. Had the output format been JPG, which only supports single frames, the output would have consisted of separate frames. More about that below, in the section about the Output Filename.

Selecting an Image Region

Raw images are a sequence of color intensities without additional meta information such as width, height, or image signature. With raw image formats, you must specify the image width and height but you can also specify a region of the image to read. In our example, the image is in the raw 8-bit RGB format and is 6000 pixels wide and 4000 pixels high. However, we only want a region of 600 by 400 near the center of the image:

```
magick -size 6000x4000 -depth 8 'rgb:image[600x400+1900+2900]' image.jpg
```

You can get the same results with the -extract option:

```
magick -size 6000x4000 -depth 8 -extract 600x400+1900+2900 rgb:image image.jpg
```

Inline Image Resize

It is sometimes convenient to resize an image as they are read. Suppose you have hundreds of large JPEG images you want to convert to a sequence of PNG thumbails:

```
magick '*.jpg' -resize 120x120 thumbnail%03d.png
```

Here *all* the images are read and subsequently resized. It is faster and less resource intensive to resize each image as it is read:

```
magick '*.jpg[120x120]' thumbnail%03d.png
```

Inline Image Crop

It is sometimes convenient to crop an image as they are read. Suppose you have hundreds of large JPEG images you want to convert to a sequence of PNG thumbails:

```
magick '*.jpg' -crop 120x120+10+5 thumbnail%03d.png
```

Here *all* the images are read and subsequently cropped. It is faster and less resource-intensive to crop each image as it is read:

```
magick '*.jpg[120x120+10+5]' thumbnail%03d.png
```

Filename References

There are two methods to use a filename to reference other image filenames. The first is with ' `@` ' which reads image filenames separated by white space from the specified file. Assume the file `myimages.txt` consists of a list of filenames, like so:

```
frame001.jpg
frame002.jpg
frame003.jpg
```

We then expect this command:

```
magick @myimages.txt mymovie.gif
```

to read the images `frame001.jpg` , `frame002.jpg` , and `frame003.jpg` and convert them to a GIF image sequence.

If the image path includes one or more spaces, enclose the path in quotes:

```
'my title.jpg'
```

Some ImageMagick command-line options may exceed the capabilities of your command-line processor. Windows, for example, limits command-lines to 8192 characters. If, for example, you have a draw option with polygon points that exceed the command-line length limit, put the draw option instead in a file and reference the file with the `@` (e.g. `@mypoly.txt`).

Another method of referring to other image files is by embedding a formatting character in the filename with a scene range. Consider the filename `image-%d.jpg[1-5]` . The command

```
magick image-%d.jpg[1-5]
```

causes ImageMagick to attempt to read images with these filenames:

```
image-1.jpg
image-2.jpg
image-3.jpg
image-4.jpg
image-5.jpg
```

Stream Buffering

By default, the input stream is buffered. To ensure information on the source file or terminal is read as soon as its available, set the buffer size to 0:

```
magick logo: gif:- | display -define stream:buffer-size=0 gif:-
```

Command-line Options

You can direct the behavior of ImageMagick utilities with these command-line options. The behavior of an option falls into one of these categories:

- Image Setting
- Image Operator
- Image Channel Operator
- Image Sequence Operator
- Image Geometry
- Image Stack

Image Setting

An image setting persists as it appears on the command-line and may affect subsequent processing such as reading an image, an image operator, or when writing an image as appropriate. An image setting stays in effect until it is reset or the command-line terminates. The image settings include:

−adjoin • −affine • −alpha • −antialias • −authenticate • −background • −bias • −black−point−compensation • −blue−primary • −bordercolor • −caption • −channel • −comment • −compress • −debug • −define • −delay • −density • −depth • −direction • −display • −dispose • −dither • −encoding • −endian • −extract • −family • −fill • −filter • −font • −format • −fuzz • −geometry • −gravity • −green−primary • −interlace • −intent • −interpolate • −label • −limit • −linewidth • −log • −loop • −mattecolor • −monitor • −orient • −page • −pointsize • −preview • −quality • −quiet • −read−mask • −red−primary • −region • −render • −repage • −sampling−factor • −scene • −seed • −size • −stretch • −stroke • −strokewidth • −style • −texture • −tile • −transparent−color • −treedepth • −type • −undercolor • −units • −verbose • −virtual−pixel • −weight • −write−mask

In this example, *-channel* applies to each of the images, since, as we mentioned, settings persist:

```
magick -channel RGB wand.png wizard.png images.png
```

Image Operator

An image operator differs from a setting in that it affects the image immediately as it appears on the command-line. An operator is any command-line option not listed as a image setting or image sequence operator. Unlike an image setting, which persists until the command-line terminates, an operator is applied to the current image set and forgotten. The image operators include:

−annotate • −black−threshold • −blur • −border • −charcoal • −chop • −clip • −clip−path • −clip−mask • −colors • −colorize • −colorspace • −compose • −contrast • −convolve • −crop • −cycle • −despeckle • −draw • −edge • −emboss • −enhance • −equalize • −evaluate • −extent • −flip • −flop • −floodfill • −frame • −gamma • −gaussian−blur • −grayscale • −implode • −lat • −level • −map • −median • −modulate • −monochrome • −negate • −noise • −normalize • −opaque • −ordered−dither • −paint • −posterize • −raise • −profile • −radial−blur • −raise • −random−threshold • −resample • −resize • −roll • −rotate • −sample • −scale • −sepia−tone • −segment • −shade • −shadow • −sharpen • −shave • −shear • −sigmoidal−contrast • −solarize • −splice • −spread • −strip • −swirl • −threshold • −transparent • −thumbnail • −tint • −transform • −trim • −unsharp • −version • −wave • −white−point • −white−threshold

In this example, *-negate* negates the wand image but not the wizard:

```
magick wand.png -negate wizard.png images.png
```

Note that an image operator will be applied to each images in an image sequence. For example, if you use −resize option to resize a GIF image, each frames will be resized to the given size. However, some frames may be smaller than the whole image and resizing all the frames into the same size may result in an unexpected output. In such a case, −coalesce should be used to prepare those frames.

Image Channel Operator

Operate directly on image channels:

−channel−fx • −separate

Image Sequence Operator

An image sequence operator differs from a setting in that it affects an image sequence immediately as it appears on the command-line. Choose from these image sequence operators:

−append • −affinity • −average • −clut • −coalesce • −combine • −compare • −complex • −composite • −copy • −crop • −debug • −deconstruct • −delete • −evaluate−sequence • −fft • −flatten • −fx • −hald−clut • −ift • −identify • −insert • −layers • −limit • −map • −maximum • −minimum • −morph • −mosaic • −optimize • −print • −process • −quiet • −swap • −write

In this example, *-append* appends three images into one:

```
magick mikayla.png picnic.png beach.png -append vacation.png
```

Image Geometry

Many command-line options take a *geometry* argument to specify such things as the desired width and height of an image and other dimensional quantities. Because users want so many variations on the resulting dimensions, sizes, and positions of images (and because ImageMagick wants to provide them), the *geometry* argument can take many forms. We describe many of these in this section.

The image options and settings that take some form of a *geometry* argument include the following. Keep in mind that some of these parse their arguments in slightly different ways. See the documentation for the individual option or setting for more specifics.

−adaptive−resize • −border • −borderwidth • −chop • −crop • −density • −extent • −extract • −frame • −geometry • −iconGeometry • −liquid−rescale • −page • −region • −repage • −resize • −sample • −scale • −shave • −splice • −thumbnail • −window

The *geometry* argument might take any of the forms listed in the table below. These will described in more detail in the subsections following the table. The usual form is *size*[*offset*], meaning *size* is required and *offset* is optional. Occasionally, [*size*]*offset* is possible. In no cases are spaces permitted within the *geometry* argument.

size	General description (actual behavior can vary for different options and settings)
scale%	Height and width both scaled by specified percentage.
scale-x%x*scale-y*%	Height and width individually scaled by specified percentages. (Only one % symbol needed.)
width	Width given, height automagically selected to preserve aspect ratio.
x*height*	Height given, width automagically selected to preserve aspect ratio.
*width*x*height*	Maximum values of height and width given, aspect ratio preserved.
*width*x*height*^	Minimum values of width and height given, aspect ratio preserved.

widthxheight!	Width and height emphatically given, original aspect ratio ignored.
widthxheight>	Shrinks an image with dimension(s) larger than the corresponding *width* and/or *height* argument(s).
widthxheight<	Enlarges an image with dimension(s) smaller than the corresponding *width* and/or *height* argument(s).
area@	Resize image to have specified area in pixels. Aspect ratio is preserved.
x:*y*	Here x and y denotes an aspect ratio (e.g. 3:2 = 1.5).
{*size*}{*offset*}	Specifying the *offset* (default is `+0+0`). Below, {*size*} refers to any of the forms above.
{*size*}{+-}*x*{+-}*y*	Horizontal and vertical offsets *x* and *y*, specified in pixels. Signs are required for both. Offsets are affected by -gravity setting. Offsets are not affected by `%` or other *size* operators.

Basic adjustments to width and height; the operators `%` , `^` , and `!`

Here, just below, are a few simple examples of *geometry*, showing how it might be used as an argument to the -resize option. We'll use the internal image `logo:` for our input image. This fine image is 640 pixels wide and 480 pixels high. We say its *dimensions* are 640x480. When we give dimensions of an image, the width (the horizontal dimension) always precedes the height (the vertical dimension). This will be true when we speak of coordinates or *offsets* into an image, which will always be *x*–value followed by *y*. Just think of your high school algebra classes and the *xy*–plane. (Well, almost: our *y*–axis will always go downward!)

```
magick logo: -resize '200%' bigWiz.png
magick logo: -resize '200x50%' longShortWiz.png
magick logo: -resize '100x200' notThinWiz.png
magick logo: -resize '100x200^' biggerNotThinWiz.png
magick logo: -resize '100x200!' dochThinWiz.png
```

The first of the four commands is simple—it stretches both the width and height of the input image by `200%` in each direction; it magnifies the whole thing by a factor of two. The second command specifies different percentages for each direction, stretching the width to `200` % and squashing the height to `50%` . The resulting image (in this example) has dimensions 1280x240. Notice that the percent symbol needn't be repeated; the following are equivalent: `200x50%` , `200%x50` , `200%x50%` .

By default, the width and height given in a *geometry* argument are *maximum* values unless a percentage is specified. That is, the image is expanded or contracted to fit the specified width and height value while maintaining the *aspect ratio* (the ratio of its height to its width) of the image. For instance, the third command above "tries" to set the dimensions to `100x200` . Imagine gradually shrinking the original image (which is 640x480), keeping is aspect ratio constant, until it just fits into a 100x200 rectangle. Since the image is longer than it is tall, it will fit when its width shrinks to 100 pixels. To preserve the aspect ratio, the height will therefore have to be (480/640)×100 pixels=75 pixels, so the final dimensions will be 100x75.

Notice that in the previous example, at least one of the specified dimensions will be attained (in this case, the width, 100 pixels). The resulting image fits snugly within the original. One can do just the opposite of this by invoking the `^` operator, as in the fourth example above. In that case, when `100x200^` is given as the argument, again at least one of the dimensions will be attained, but in this case the resulting image can snugly contain the original. Here the *geometry* argument gives *minimum* values. In our example, the height will become 200 and the width will be scaled to preserve the aspect ratio, becoming (640/480)×200 pixels=267 pixels. With the `^` operator, one of those dimensions will match the requested size, but the image will likely overflow the dimensions requested to preserve its aspect ratio. (The `^` feature is new as of IM 6.3.8-2.)

We see that ImageMagick is very good about preserving aspect ratios of images, to prevent distortion of your favorite photos and images. But you might really want the dimensions to be `100x200`, thereby stretching the image. In this case just tell ImageMagick you really mean it (!) by appending an exclamation operator to the geometry. This will force the image size to exactly what you specify. So, for example, if you specify `100x200!` the dimensions will become exactly 100x200 (giving a small, vertically elongated wizard).

Bounding the width, height, and area; the operators `>`, `<`, and `@`

Here are a few more examples:

```
magick logo: -resize '100' wiz1.png
magick logo: -resize 'x200' wiz2.png
magick logo: -resize '100x200>' wiz3.png
magick logo: -resize '100x200<' wiz4.png
```

If only one dimension is given it is taken to be the width. When only the width is specified, as in the first example above, the width is accepted as given and the height is chosen to maintain the aspect ratio of the input image. Similarly, if only the height is specified, as in the second example above, the height is accepted and the width is chosen to maintain the aspect ratio.

Use `>` to shrink an image *only* if its dimension(s) are larger than the corresponding *width* and/or *height* arguments. Use `<` to enlarge an image *only* if its dimension(s) are smaller than the corresponding *width* and/or *height* arguments. In either case, if a change is made, the result is as if the `>` or `<` operator was not present. So, in the third example above, we specified `100x200>` and the original image size is 640x480, so the image size is reduced as if we had specified `100x200`. However, in the fourth example above, there will be no change to its size.

Finally, use `@` to specify the maximum area in pixels of an image, again while attempting to preserve aspect ratio. (Pixels take only integer values, so some approximation is always at work.) In the following example, an area of 10000 pixels is requested. The resulting file has dimensions 115x86, which has 9890 pixels.

```
magick logo: -resize '10000@' wiz10000.png
```

In all the examples above and below, we have enclosed the *geometry* arguments within quotation marks. Doing so is optional in many cases, but not always. We *must* enclose the geometry specifications in quotation marks when using `<` or `>` to prevent these characters from being interpreted by the shell as *file redirection*. On Windows systems, the carat `^` needs to be within quotes, else it is ignored. To be safe, one should probably maintain a habit of enclosing all *geometry* arguments in quotes, as we have here.

Offsets in geometry

Here are some examples to illustrate the use of *offsets* in *geometry* arguments. One typical use of offsets is in conjunction with the –region option. This option allows many other options to modify the pixels within a specified rectangular subregion of an image. As such, it needs to be given the width and height of that region, and also an *offset* into the image, which is a pair of coordinates that indicate the location of the region within the larger image. Below, in the first example, we specify a region of size `100x200` to be located at the *xy*–coordinates *x*=10, *y*=20. Let's use the usual algebraic notation (*x*,*y*)=(10,20), for convenience.

```
magick logo: -region '100x200+10+20' -negate wizNeg1.png
magick logo: -region '100x200-10+20' -negate wizNeg2.png
magick logo: -gravity center -region '100x200-10+20' -negate wizNeg3.png
```

Note that offsets always require +/− signs. The offset is not actually a true location within the image; its coordinates must be added to some other location. Let's refer to that as the *current location*. In the first two examples above, though, that location is the upper-left hand corner of the image, which has coordinates (0,0). (That is the default situation when there are no other directives given to change it.) The first example above puts the `100x200` rectangle's own upper-left corner at (10,20).

A negative offset can make sense in many cases. In the second example above, the offset is (-10,20), specified by `-10+20`. In that case, only the portion of the (virtual) rectangle obtained that lies within the image can be negated; here it is equivalent to specifying the geometry as `90x200+0+20`.

In the third example above, the −gravity setting precedes the others and sets the current location within the image at the very center of the image. In this case that is at pixel (320,240), since the size of the image is 640x480. This means that the offsets apply to that location, which thereby gets moved, in this case, to (320-10,240+20)=(310,260). But the `100x200` region itself is affected by the −gravity setting, so instead of affecting its upper-left corner, the region's own center (at (+50,+100) within it) is determined. Therefore the center of the `100x200` rectangle is moved to (310,260). The negated rectangle's upper-left corner is now at (310-50,260-100)=(260,160).

Image Stack

In school, your teacher probably permitted you to work on problems on a scrap of paper and then copy the results to your test paper. An image stack is similar. It permits you to work on an image or image sequence in isolation and subsequently introduce the results back into the command-line. The image stack is delineated with parenthesis. Image operators only affect images in the current stack. For example, we can limit the image rotation to just the wizard image like this:

```
magick wand.gif \( wizard.gif -rotate 30 \) +append images.gif
```

Notice again that the parentheses are *escaped* by preceding them with backslashes. This is required under Unix, where parentheses are special *shell* characters. The backslash tells the shell not to interpret these characters, but to pass them directly to the command being executed. Do not escape the parentheses under Windows. Each parenthesis (or escaped parenthesis) must have spaces on either side, as in the example shown above.

In addition to the image operators already discussed, the following image operators are most useful when processing images in an image stack:

−clone • −delete • −insert • −swap

The arguments to these operators are indexes into the image sequence by number, starting with zero, for the first image, and so on. However if you give a negative index, the images are indexed from the end (last image added). That is, an index of -1 is the last image in the current image sequence, -2 gives the second-to-last, and so on.

Output Filename

ImageMagick extends the concept of an output filename to include:

1. an explicit image format
2. write to *standard out*
3. filename references

Each of these extensions are explained in the next few paragraphs.

Explicit Image Format

Images can be stored in a mryiad of image formats including the better known JPEG, PNG, TIFF and others. ImageMagick must know the desired format of the image before it is written. ImageMagick leverages the filename extension to determine the format. For example, `image.jpg` tells ImageMagick to write the image in the JPEG format.

In some cases the filename does not identify the image format. In these cases, the image is written in the format it was originally read unless an explicit image format is specified. For example, suppose we want to write our image to a filename of `image` in the raw red, green, and blue intensity format:

```
magick image.jpg rgb:image
```

Standard Out

Unix permits the output of one command to be piped to another. ImageMagick permits piping one command to another with a filename of `-`. In this example we pipe the output of convert to the display program:

```
magick logo: gif:- | display gif:-
```

Here the explicit format is optional. The GIF image format has a signature that uniquely identifies it so ImageMagick can readily recognize the format as GIF.

Filename References

Optionally, use an embedded formatting character to write a sequential image list. Suppose our output filename is `image-%d.jpg` and our image list includes 3 images. You can expect these images files to be written:

```
image-0.jpg
image-1.jpg
image-2.jpg
```

Or retrieve image properties to modify the image filename. For example, the command

```
magick rose: -set filename:area '%wx%h' 'rose-%[filename:area].png'
```

writes an image with this filename:

```
rose-70x46.png
```

Finally to convert multiple JPEG images to individual PDF pages, use:

```
magick *.jpg +adjoin page-%d.pdf
```

Stream Buffering

By default, the output stream is buffered. To ensure information appears on the destination file or terminal as soon as written, set the buffer size to 0:

```
magick -define stream:buffer-size=0 logo: gif:- | display gif:-
```

</div>

Configuration Files • Modules • Fonts • Environment Variables

ImageMagick depends on a number of external resources including configuration files, loadable modules, fonts, and environment variables.

Configuration Files

ImageMagick depends on a number of external configuration files detailed here:

coder.xml

Associate an image format with the specified coder module. ImageMagick has a number of coder modules to support the reading and/or writing of an image format (e.g. JPEG). Some coder modules support more than one associated image format and the mapping between an associated format and its respective coder module is defined in this configuration file. For example, the PNG coder module not only supports the PNG image format, but the JNG and MNG formats as well.

colors.xml

Associate a color name with its red, green, blue, and alpha intensities. A number of command line options require a color parameter. It is often convenient to refer to a color by name (e.g. white) rather than by hex value (e.g. #fff). This file maps a color name to its equivalent red, green, blue, and alpha intensities (e.g. for white, red = 255, green = 255, blue = 255, and alpha = 0).

configure.xml

Set ImageMagick build parameters and system-wide environment variables (e.g. MAGICK_TEMPORARY_PATH). As ImageMagick is built, a number of build parameters are saved to this configuration file. They include the version, release date, dependent delegate libraries, and quantum depth among others.

delegates.xml

Associate delegate programs with certain image formats. ImageMagick relies on a number of delegate programs to support certain image formats such as ufraw-batch to read raw camera formats or Ghostscript to read Postscript images. Use this configuration file to map an input or output format to an external delegate program.

english.xml

Associate message tags with English translations.

francais.xml

Associate message tags with French translations.

locale.xml

Associate message tags with a translation for your locale. ImageMagick has a number of informational, warning, and error messages that are represented as tags. Tags are short descriptions of a message such as *FileNotFound* or *MemoryAllocationFailed*. This configuration file lists locales that have a translation for each tag recognized by ImageMagick. Currently only English and French translations are available in the `english.xml` and `francais.xml` configuration files.

log.xml

Configure logging parameters. ImageMagick is capable of spewing copious amounts of informational or debugging statements. Use this file to configure how the information will appear in a log message and where you want the logging messages posted.

mime.xml

Associate an internet media type with a unique identifier. Many files and data streams have identifiers that uniquely identify a particular internet media type. For example, files in the "Corel Draw drawing" format (mime type="application/vnd.corel-draw") are associated with the filename pattern `*.cdr`, and also have an initial string of the characters "CDRXvrsn". ImageMagick uses combinations of this information, when available, to attempt to quickly determine the internet media type of a file or data stream.

policy.xml

Configure ImageMagick policies. By default any coder, delegate, filter, or file path is permitted. Use a policy to deny access to, for example, the MPEG video delegate, or permit reading images from a file system but deny writing to that same file system. Or use the resource policy to set resource limits. Policies are useful for multi-user servers that want to limit the overall impact ImageMagick has on the system. For example, to limit the maximum image size in memory to 100MP:

```
<policy domain="resource" name="area" value="100MP"/>
```

Any image larger than this area limit is cached to disk rather than memory. Use `width` to limit the maximum width of an image in pixels. Exceed this limit and an exception is thrown and processing stops.

```
<policy domain="resource" name="width" value="8KP"/>
```

To limit the elapsed time of any ImageMagick command to 5 minutes, use this policy:

```
<policy domain="resource" name="time" value="300"/>
```

Define arguments for the memory, map, and disk resources with SI prefixes (.e.g 100MB). In addition, resource policies are maximums for each instance of ImageMagick (e.g. policy memory limit 1GB, the `-limit 2GB` option exceeds policy maximum so memory limit is 1GB).

quantization-table.xml

Custom JPEG quantization tables. Activate with `-define:q-table=quantization-table.xml` .

thresholds.xml

Set threshold maps for ordered posterized dither.

type.xml

Configure fonts. Define the font name, family, foundry, style, format, metrics, and glyphs for any font you want to use within ImageMagick.

type-ghostscript.xml

Configure Ghostscript fonts. The Ghostscript package includes a number of fonts that can be accessed with ImageMagick.

type-windows.xml

Associate names with Windows font glyphs.

Under Unix and Linux, ImageMagick searches for each of the configuration files listed above by looking in the locations given below, in order, and loads them if found:

```
$MAGICK_CONFIGURE_PATH
$PREFIX/etc/ImageMagick-7
$PREFIX/share/ImageMagick-7
$XDG_CACHE_HOME/ImageMagick
$HOME/.config/ImageMagick
<client path>/etc/ImageMagick
```

The environmental variable $PREFIX is the default install path (e.g. `/usr/local`). The *client path* is the execution path of your ImageMagick client (e.g. `/usr/local`) .

For the Unix or Linux pre-compiled uninstalled binary distributions, the configuration load order is:

```
$MAGICK_CONFIGURE_PATH
$MAGICK_HOME/etc/ImageMagick-7
```

```
$MAGICK_HOME/share/ImageMagick-7
$PREFIX/share/ImageMagick-7
$XDG_CACHE_HOME/ImageMagick
$HOME/.config/ImageMagick/
<client path>/etc/ImageMagick
<current directory>
```

Under Windows, ImageMagick searches for these configuration files in the following order, and loads them if found:

```
$MAGICK_CONFIGURE_PATH
<windows registry>
$PREFIX/config
$USERPROFILE/.config/ImageMagick
<client path>
```

Above, $PREFIX is the default install path, typically `c:\\Program Files\\ImageMagick-7.0.8`.

For an uninstalled Windows installation, the configuration load order is:

```
$MAGICK_CONFIGURE_PATH
$MAGICK_HOME
$USERPROFILE/.config/ImageMagick
client path
<current directory>
```

If a configuration file cannot not be found, ImageMagick relies on built-in default values.

Modules

Coders

An image coder (i.e. encoder / decoder) is responsible for registering, optionally classifying, optionally reading, optionally writing, and unregistering one image format (e.g. PNG, GIF, JPEG, etc.). ImageMagick searches for coders in the following order and it uses the first match found:

```
$MAGICK_HOME/lib/ImageMagick-7.0.8/modules-Q16/coders
<client path>/../lib/ImageMagick-7.0.8/modules-Q16/coders
$MAGICK_HOME/lib/ImageMagick-7.0.8/modules-Q16/coders
$MAGICK_HOME/share/ImageMagick-7.0.8/modules-Q16/coders
$XDG_CACHE_HOME/ImageMagick
$HOME/.config/ImageMagick
<client path>/lib/ImageMagick-7.0.8/modules-Q16/coders
```

Filters

ImageMagick provides a convenient mechanism for adding your own custom image processing algorithms. ImageMagick searches for filters in the following order and it uses the first match found:

```
$MAGICK_HOME/lib/ImageMagick-7.0.8/modules-Q16/filters
<client path>/../lib/ImageMagick-7.0.8/modules-Q16/filters
$MAGICK_HOME/lib/ImageMagick-7.0.8/modules-Q16/filters
$MAGICK_HOME/share/ImageMagick-7.0.8/modules-Q16/filters
$XDG_CACHE_HOME/ImageMagick
$HOME/.config/ImageMagick
<client path>/lib/ImageMagick-7.0.8/modules-Q16/filters
```

Fonts

ImageMagick is able to load raw TrueType and Postscript font files. It searches for the font configuration file, type.xml, in the following order, and loads them if found:

```
$MAGICK_CONFIGURE_PATH
$MAGICK_HOME/etc/ImageMagick/-7.0.8
$MAGICK_HOME/share/ImageMagick-7.0.8
$XDG_CACHE_HOME/ImageMagick
$HOME/.config/ImageMagick
<client path>/etc/ImageMagick
$MAGICK_FONT_PATH
```

Environment Variables

Environment variables recognized by ImageMagick include:

HOME	Set path to search for configuration files in `$HOME/.config/ImageMagick` if the directory exists.
LD_LIBRARY_PATH	Set path to the ImageMagick shareable libraries and other dependent libraries.
MAGICK_AREA_LIMIT	Set the maximum *width * height* of an image that can reside in the pixel cache memory. Images that exceed the area limit are cached to disk (see MAGICK_DISK_LIMIT) and optionally memory-mapped.
MAGICK_CODER_FILTER_PATH	Set search path to use when searching for filter process modules (invoked via -process). This path permits the user to extend ImageMagick's image processing functionality by adding loadable modules to a preferred location rather than copying them into the ImageMagick installation directory. The formatting of the search path is similar to operating system search paths (i.e. colon delimited for Unix, and semi-colon delimited for Microsoft Windows). This user specified search path is searched before trying the default search path.
MAGICK_CODER_MODULE_PATH	Set path where ImageMagick can locate its coder modules. This path permits the user to arbitrarily extend the image formats supported by ImageMagick by adding loadable coder modules from an preferred location rather than copying them into the ImageMagick installation directory. The formatting of the search path is similar to operating system search paths (i.e. colon delimited for Unix, and semi-colon delimited for Microsoft Windows). This user specified search path is searched before trying the default search path.
MAGICK_CONFIGURE_PATH	Set path where ImageMagick can locate its configuration files. Use this search path to search for configuration (.xml) files. The formatting of the search path is similar to operating system search paths (i.e. colon delimited for Unix, and semi-colon delimited for Microsoft Windows). This user specified search path is searched before trying the default search path.
MAGICK_DEBUG	Set debug options. See -debug for a description of debugging options.
MAGICK_DISK_LIMIT	Set maximum amount of disk space in bytes permitted for use by the pixel cache. When this limit is exceeded, the pixel cache is not be created and an error message is returned.
MAGICK_ERRORMODE	Set the process error mode (Windows only). A typical use might be a value of 1 to prevent error mode dialogs from displaying a message box and hanging the application.
MAGICK_FILE_LIMIT	Set maximum number of open pixel cache files. When this limit is exceeded, any subsequent pixels cached to disk are closed and reopened on demand. This behavior permits a large number of images to be accessed simultaneously on disk, but with a speed penalty due to repeated open/close calls.

MAGICK_FONT_PATH	Set path ImageMagick searches for TrueType and Postscript Type1 font files. This path is only consulted if a particular font file is not found in the current directory.
MAGICK_HEIGHT_LIMIT	Set the maximum *height* of an image.
MAGICK_HOME	Set the path at the top of ImageMagick installation directory. This path is consulted by *uninstalled* builds of ImageMagick which do not have their location hard-coded or set by an installer.
MAGICK_LIST_LENGTH_LIMIT	Set the maximum length of an image sequence.
MAGICK_MAP_LIMIT	Set maximum amount of memory map in bytes to allocate for the pixel cache. When this limit is exceeded, the image pixels are cached to disk (see MAGICK_DISK_LIMIT).
MAGICK_MEMORY_LIMIT	Set maximum amount of memory in bytes to allocate for the pixel cache from the heap. When this limit is exceeded, the image pixels are cached to memory-mapped disk (see MAGICK_MAP_LIMIT).
MAGICK_OCL_DEVICE	Set to `off` to disable hardware acceleration of certain accelerated algorithms (e.g. blur, convolve, etc.).
MAGICK_PRECISION	Set the maximum number of significant digits to be printed.
MAGICK_SHRED_PASSES	If you want to keep the temporary files ImageMagick creates private, overwrite them with zeros or random data before they are removed. On the first pass, the file is zeroed. For subsequent passes, random data is written.
MAGICK_SYNCHRONIZE	Set to "true" to ensure all image data is fully flushed and synchronized to disk. There is a performance penalty, however, the benefits include ensuring a valid image file in the event of a system crash and early reporting if there is not enough disk space for the image pixel cache.
MAGICK_TEMPORARY_PATH	Set path to store temporary files.
MAGICK_THREAD_LIMIT	Set maximum parallel threads. Many ImageMagick algorithms run in parallel on multi-processor systems. Use this environment variable to set the maximum number of threads that are permitted to run in parallel.
MAGICK_THROTTLE_LIMIT	Periodically yield the CPU for at least the time specified in milliseconds.
MAGICK_TIME_LIMIT	Set maximum time in seconds. When this limit is exceeded, an exception is thrown and processing stops.
MAGICK_WIDTH_LIMIT	Set the maximum *width* of an image.

Define arguments for the `MAGICK_MEMORY_LIMIT` , `MAGICK_DISK_LIMIT` , and `MAGICK_MEMORY_LIMIT` environment variables with SI prefixes (.e.g `100MB`). `MAGICK_WIDTH_LIMIT` , `MAGICK_HEIGHT_LIMIT` and `MAGICK_AREA_LIMIT` accepts pixel suffixes such as MP for mega-pixels (e.g. 100MP).

</div>

The functionality of ImageMagick is typically utilized from the command-line or you can use the features from programs written in your favorite language. Choose from these interfaces: G2F (Ada), MagickCore (C), MagickWand (C), ChMagick (Ch), ImageMagickObject (COM+), Magick++ (C++), JMagick (Java), WASM-ImageMagick (Javascript/Typescript), JuliaIO (Julia), L-Magick (Lisp), Lua (LuaJIT), NMagick (Neko/haXe), Magick.NET (.NET), PascalMagick (Pascal), PerlMagick (Perl), MagickWand for PHP (PHP), IMagick (PHP), PythonMagick (Python), magick (R), RMagick (Ruby), or TclMagick (Tcl/TK). With a language interface, use ImageMagick to modify or create images dynamically and *automagically*.

Choose from these language interfaces:

C

Use MagickWand to convert, compose, and edit images from the C language. There is also the low-level MagickCore library for wizard-level developers.

Ch

ChMagick is a Ch binding to the MagickCore and MagickWand API. Ch is an embeddable C/C++ interpreter for cross-platform scripting.

COM+

Use ImageMagickObject to convert, compose, and edit images from a Windows COM+ compatible component.

C++

Magick++ provides an object-oriented C++ interface to ImageMagick. See A Gentle Introduction to Magick++ for an introductory tutorial to Magick++. We include the source if you want to correct, enhance, or expand the tutorial.

GO

GoImagick is a set of Go bindings to ImageMagick's MagickWand and MagickCore C APIs.

Java

JMagick provides an object-oriented Java interface to ImageMagick. Im4java is a pure-java interface to the ImageMagick command-line.

Javascript/TypeScript

WASM-ImageMagick Webassembly compiliation of ImageMagick that allows serverless clientside bindings for Typescript and Javascript. Works in Progressive Web Apps.

Julia

JuliaIO provides an object-oriented Julia interface to ImageMagick.

LabVIEW

LVOOP ImageMagick is an object-oriented LabVIEW interface to ImageMagick.

Lisp

CL-Magick provides a Common Lisp interface to the ImageMagick library.

Lua

Lua bindings to ImageMagick for LuaJIT using FFI.

Lua bindings to ImageMagick for Lua using pure-C.

Neko

NMagick is a port of the ImageMagick library to the haXe and Neko platforms. It provides image manipulation capabilities to both web and desktop applications using Neko.

.NET

Use Magick.NET to convert, compose, and edit images from Windows .NET.

ImageMagickApp is a .NET application written in C# that utilizes the ImageMagick command line to allow conversion of multiple image formats to different formats.

Pascal

PascalMagick a Pascal binding for the MagickWand API and also the low-level MagickCore library. It works with Free Pascal / Lazarus and Delphi.

Perl

Use PerlMagick to convert, compose, and edit images from the Perl language.

PHP

MagickWand for PHP a native PHP-extension to the ImageMagick MagickWand API.

IMagick is a native PHP extension to create and modify images using the ImageMagick API. Documentation for the extension is available here.

phMagick is a wrapper class for ImageMagick, wrapping the most common web image manipulation actions in easy to use functions, but allowing full access to ImageMagick's power by issuing system calls to it's command-line programs.

Python

Wand is a ctypes-based ImagedMagick binding library for Python.

PythonMagick is an object-oriented Python interface to ImageMagick.

PythonMagickWand is an object-oriented Python interface to MagickWand based on ctypes.

Scilab Image Processing toolbox utilizes ImageMagick to do imaging tasks such as filtering, blurring, edge detection, thresholding, histogram manipulation, segmentation, mathematical morphology, color image processing, etc..

REALbasic

The MBS Realbasic ImageMagick is a plugin that utilizes the power of ImageMagick from within the RealBasic environment.

R

The magick package wraps the Magick++ STL to provide vectorized image processing in R. Get started with using the package vignette.

Ruby

RMagick is an interface between the Ruby programming language and the MagickCore image processing libraries. Get started with RMagick by perusing the documentation.

MagickWand for Ruby is an interface between the Ruby programming language and the MagickWand image processing libraries. Get started with MagickWand for PHP by perusing the documentation.

MiniMagick is a Ruby wrapper for ImageMagick command line. MiniMagick gives you convenient access to all the command line options ImageMagick supports.

QuickMagick is a gem for easily accessing ImageMagick command line tools from Ruby programs.

Rust

RustWand is a MagickWand bindings for the Rust language.

Tcl/Tk

TclMagick a native Tcl-extension to the ImageMagick MagickWand API.

XML RPC

RemoteMagick is an XML-RPC web service that creates image thumbnails.