Wand Documentation

Release 0.6.10

Hong Minhee **E. McConville**

Aug 15, 2022

CONTENTS

1 Why just another binding? 3

2 Requirements 5

3 User's guide 7
- 3.1 What's new in Wand 0.6? . . . 7
- 3.2 Installation . . . 9
- 3.3 Security . . . 13
- 3.4 Reading images . . . 14
- 3.5 Writing images . . . 18
- 3.6 Resizing and cropping . . . 20
- 3.7 Image Effects . . . 24
- 3.8 Special Effects (FX) . . . 34
- 3.9 Transformation . . . 46
- 3.10 Colorspace . . . 51
- 3.11 Color Enhancement . . . 52
- 3.12 Distortion . . . 58
- 3.13 Drawing . . . 71
- 3.14 Reading EXIF . . . 83
- 3.15 Layers . . . 84
- 3.16 Montage . . . 89
- 3.17 Morphology . . . 94
- 3.18 Sequence . . . 104
- 3.19 Resource management . . . 107
- 3.20 Quantize . . . 107
- 3.21 Threshold . . . 111
- 3.22 CLI Reference . . . 118
- 3.23 Running tests . . . 123
- 3.24 Roadmap . . . 124
- 3.25 Wand Changelog . . . 125
- 3.26 Talks and Presentations . . . 147

4 References 149
- 4.1 `wand` — Simple MagickWand API binding for Python . . . 149

5 Troubleshooting 271
- 5.1 Stack Overflow . . . 271
- 5.2 Documentation . . . 271

6 Open source 273

7 Indices and tables 275

Python Module Index 277

Index 279

Wand is a ctypes-based simple ImageMagick binding for Python.

```
from wand.image import Image
from wand.display import display

with Image(filename='mona-lisa.png') as img:
    print(img.size)
    for r in 1, 2, 3:
        with img.clone() as i:
            i.resize(int(i.width * r * 0.25), int(i.height * r * 0.25))
            i.rotate(90 * r)
            i.save(filename='mona-lisa-{0}.png'.format(r))
            display(i)
```

You can install it from PyPI (and it requires MagickWand library):

```
$ apt-get install libmagickwand-dev
$ pip install Wand
```

CHAPTER

ONE

WHY JUST ANOTHER BINDING?

There are already many MagickWand API bindings for Python, however they are lacking something we need:

- Pythonic and modern interfaces
- Good documentation
- Binding through `ctypes` (not C API) — we are ready to go PyPy!
- Installation using **`pip`**

CHAPTER

TWO

REQUIREMENTS

- Python 2.7 or higher
  - CPython 2.7 or higher
  - CPython 3.3 or higher
  - PyPy 1.5 or higher
- MagickWand library
  - `libmagickwand-dev` for APT on Debian/Ubuntu
  - `imagemagick` for MacPorts/Homebrew on Mac
  - `ImageMagick-devel` for Yum on CentOS

CHAPTER

THREE

USER'S GUIDE

3.1 What's new in Wand 0.6?

This guide doesn't cover all changes in 0.6. See the full list of changes in *0.6 series*.

3.1.1 CMYK & Gray Color Spaces Added to Numpy's Array Interface

New in version 0.6.2.

When exporting pixel data into a Numpy array, Gray & CMYK color-spaces will be represented.

Note that the shape of the array only has one data channel for grayscale images.

```
>>> with Image(filename="rose:") as img:
...     img.transform_colorspace("gray")
...     print(np.array(img).shape)
(46, 70, 1)
```

As expected, CMYK images will export 4 bytes for each color channel.

```
>>> with Image(filename="rose:") as img:
...     img.transform_colorspace("cmyk")
...     print(np.array(img).shape)
(46, 70, 4)
```

Numpy array's do not transport channel assignment by default, so users will be responsible for passing this information back into a raster library.

```
>>> with Image.form_array(my_cmyk_array, channel_map="cmyk")  as img:
...     img.save(filename="output.tiff")
```

Users expecting to keep RGBA array shapes should perform color space transformations before passing to Numpy.

```
>>> with Image(filename="cmyk_photo.tiff") as img:
...     if img.colorspace == "cmyk":
...         img.transform_colorspace("srgb")
...     arr = numpy.array(img)
...     arr = cv2.cvtColor(arr, cv2.COLOR_RGB2BGR)
```

3.1.2 Completed MagickWand API

The majority of the MagickWand API has been integrated into Wand between 0.5 & 0.6 release. Documentation referring to incomplete, or minimal integrations of the API have been updated.

Ensure to run Wand with the latest ImageMagick-7 library to take advantage of all the new methods.

- *Image.auto_threshold()* method.
- *Image.canny()* method.
- *Image.clahe()* method. Also known as "Contrast Limited Adaptive Histogram Equalization".
- *Image.color_threshold()* method.
- *Image.complex()* method.
- *Image.connected_components()* method.
- *Image.convex_hull()* method.
- *Image.hough_lines()* method.
- *Image.kmeans()* method.
- *Image.kuwahara()* method.
- *Image.level_colors()* method.
- *Image.levelize()* method.
- *Image.levelize_colors()* method.
- *Image.local_contrast()* method.
- *Image.mean_shift()* method.
- *Image.minimum_bounding_box()* method.
- *Image.polynomial()* method.
- *Image.range_threshold()* method.
- *Image.read_mask()* method.
- *Image.rotational_blur()* method.
- *Image.wavelet_denoise()* method.
- *Image.white_balance()* method.
- *Image.write_mask()* method.

3.1.3 Numpy I/O Fixes

The original integration of Numpy's array interface exported shape data as `( WIDTH, HEIGHT, CHANNELS )`. However many other imaging libraries that work with Numpy expect this shape data as `( ROWS, COLUMNS, CHANNELS )`. Wand-0.6 adjusted the shape data to be in alignment & compatible with other libraries.

3.1.4 Documentation & Test Cases Ship with Source Distribution

The source distribution now includes Wand's reStructuredText documentation, and pytest regression tests source files. Hopefully this will help offline users. See *Running tests* document for info on local testing.

Use setuptools-extra to install additional development dependencies:

```
pip install -U Wand[doc,test]
```

3.1.5 Improved Memory Deallocation & `atexit` Support

Several memory leaks have been addressed by reworking the *`wand.resource`* allocation & deallocation functions.

It's still recommended to use Wand's *`Image`* class in a `with` statement for proper memory-resource context:

```
with Image(filename='input.jpg') as img:
    pass
```

Users not using the `with` statement forfeit memory deallocation over to Python's garbage-collector `gc` module.

The `MagickWandTerminus()` function is now only called during Python's `atexit` shutdown routine.

Note: For "What's New in Wand 0.5", see previous announcements.

3.2 Installation

Wand itself can be installed from PyPI using **`pip`**:

```
$ pip install Wand
```

Wand is a Python binding of ImageMagick, so you have to install it as well:

- *Debian/Ubuntu*
- *Fedora/CentOS*
- *Mac*
- *Windows*
- *Explicitly link to specific ImageMagick*

Or you can simply install Wand and its entire dependencies using the package manager of your system (it's way convenient but the version might be outdated):

- *Debian/Ubuntu*
- *Fedora*
- *FreeBSD*
- *Alpine*

3.2.1 Install ImageMagick on Debian/Ubuntu

If you're using Linux distributions based on Debian like Ubuntu, it can be easily installed using APT:

```
$ sudo apt-get install libmagickwand-dev
```

3.2.2 Install ImageMagick on Fedora/CentOS

If you're using Linux distributions based on Redhat like Fedora or CentOS, it can be installed using Yum:

```
$ yum update
$ yum install ImageMagick-devel
```

3.2.3 Install ImageMagick on Mac

You need one of Homebrew or MacPorts to install ImageMagick.

Homebrew

```
$ brew install imagemagick
```

MacPorts

```
$ sudo port install imagemagick
```

If your Python in not installed using MacPorts, you have to export `MAGICK_HOME` path as well. Because Python that is not installed using MacPorts doesn't look up `/opt/local`, the default path prefix of MacPorts packages.

```
$ export MAGICK_HOME=/opt/local
```

3.2.4 Install ImageMagick on Windows

You could build ImageMagick by yourself, but it requires a build tool chain like Visual Studio to compile it. The easiest way is simply downloading a prebuilt binary of ImageMagick for your architecture (`win32` or `win64`).

You can download it from the following link:

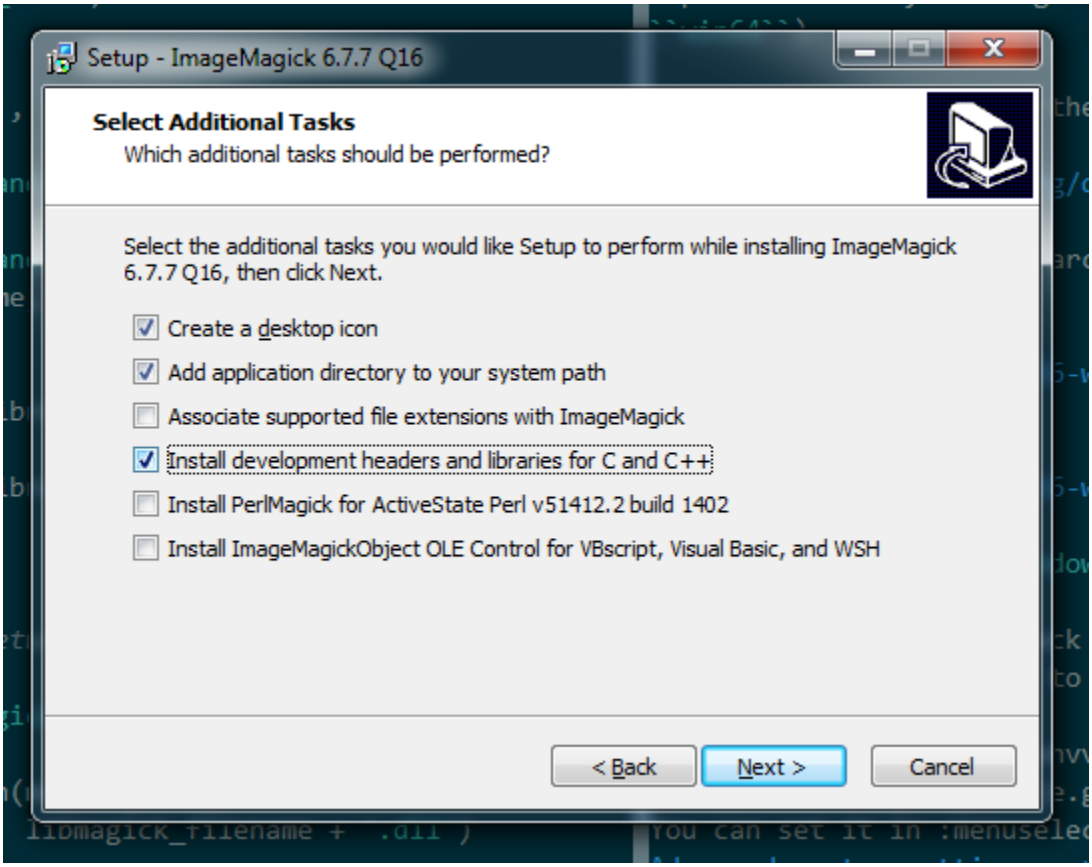
https://imagemagick.org/script/download.php#windows

Choose a binary for your architecture:

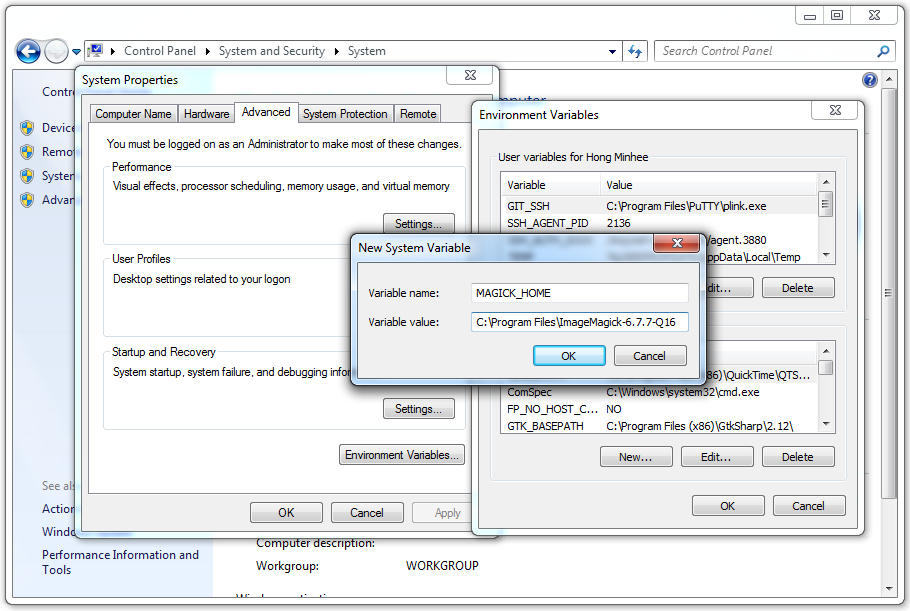
Windows 32-bit ImageMagick-7.0.x-x-Q16-x86-dll.exe

Windows 64-bit ImageMagick-7.0.x-x-Q16-HDRI-x64-dll.exe

Note: Double check your Python runtime, and ensure the architectures match. A 32-bit Python runtime can not load a 64-bit dynamic library.



Note that you have to check *Install development headers and libraries for C and C++* to make Wand able to link to it.



Lastly you have to set `MAGICK_HOME` environment variable to the path of ImageMagick (e.g. `C:\Program Files\ImageMagick-6.9.3-Q16`). You can set it in *Computer* → *Properties* → *Advanced system settings* → *Advanced* → *Environment Variables*....

3.2.5 Explicitly link to specific ImageMagick

Although Wand tries searching operating system's standard library paths for a ImageMagick installation, sometimes you need to explicitly specify the path of ImageMagick installation.

In that case, you can give the path to Wand by setting `MAGICK_HOME`. Wand respects `MAGICK_HOME`, the environment variable which has been reserved by ImageMagick.

3.2.6 Explicitly define ImageMagick library suffix

New in version 0.5.8.

Wand will attempt to load all popular combinations of ImageMagick's shared library suffixes. By default, the library suffix would follow a pattern similar to:

```
             ~~~~~~~~~~      Library Suffix
libMagickWand-7.Q16HDRI.so
             ~~              Major version number. Can be blank, 6, or 7.
                ~~~          Magick Quantum. Can be blank, Q8, or Q16.
                   ~~~~      Optional HDRI-Support. Can be blank, or HDRI
```

If you have compiled ImageMagick with custom suffixes, you can tell the Wand module how to search for it by setting `MAGICK_HOME`, like *above*, and `WAND_MAGICK_LIBRARY_SUFFIX` environment variables.

The `WAND_MAGICK_LIBRARY_SUFFIX` would be a semicolon delimited list

```
$ export WAND_MAGICK_LIBRARY_SUFFIX="-7.Q32;-7.Q32HDRI;.Q32HDRI;.Q32"
$ python3 wand_app.py
```

3.2.7 Install Wand on Debian/Ubuntu

Wand itself is already packaged in Debian/Ubuntu APT repository: python-wand. You can install it using **apt-get** command:

```
$ sudo apt-get install python-wand
```

3.2.8 Install Wand on Fedora

Wand itself is already packaged in Fedora package DB: python-wand. You can install it using **dnf** command:

```
$ dnf install python-wand   # Python 2
$ dnf install python3-wand  # Python 3
```

3.2.9 Install Wand on FreeBSD

Wand itself is already packaged in FreeBSD ports collection: py-wand. You can install it using **pkg_add** command:

```
$ pkg_add -r py-wand
```

3.2.10 Install Wand on Alpine

Wand can be installed on Alpine Linux with **pip**, but due to the security nature of Alpine, `MAGICK_HOME` must be defined before running any Wand applications.

```
# apk add imagemagick
# pip install Wand
# export MAGICK_HOME=/usr
```

You may need to create a couple symbolic links for the ImageMagick libraries.

```
# ln -s /usr/lib/libMagickCore-7.Q16HDRI.so.9 /usr/lib/libMagickCore-7.Q16HDRI.so
# ln -s /usr/lib/libMagickWand-7.Q16HDRI.so.9 /usr/lib/libMagickWand-7.Q16HDRI.so
```

3.3 Security

The authors & contributors of the `Wand` module, ImageMagick library, and all the third party image delegates make a genuine effort to release stable code. However there is a trade off between convenience & secure environment, and everyone makes honest mistakes. Ensure you're using the latest library versions, and the system is up to date with security patches. If you are using `Wand` to process images from the public, then you **must** be more vigilant.

- Never use `Wand` directly within a HTTP service, or on any server with public access. A simple queue based background worker can be used. For example: Celery, Redis, or Amazon's SQS, but there are many others.
- Update the `policy.xml` on the system, and reduce the resource limits to something reasonable to your system.

```
<policy domain="resource" name="memory" value="256MiB"/>
<policy domain="resource" name="map" value="512MiB"/>
<policy domain="resource" name="width" value="8KP"/>
<policy domain="resource" name="height" value="8KP"/>
<policy domain="resource" name="area" value="16KP"/>
<policy domain="resource" name="disk" value="1GiB"/>
<policy domain="resource" name="file" value="768"/>
<policy domain="resource" name="thread" value="1"/>
<policy domain="resource" name="throttle" value="0"/>
<policy domain="resource" name="time" value="120"/>
<policy domain="resource" name="list-length" value="128"/>
```

- Update the `policy.xml` on the system to restrict any formats that are unused, or have a history of abuse.

```
<policy domain="coder" rights="none" pattern="MVG" />
<policy domain="coder" rights="none" pattern="EPS" />
<policy domain="coder" rights="none" pattern="PS" />
<policy domain="coder" rights="none" pattern="PS2" />
<policy domain="coder" rights="none" pattern="PS3" />
<policy domain="coder" rights="none" pattern="PDF" />
```

(continues on next page)

(continued from previous page)

```
<policy domain="coder" rights="none" pattern="XPS" />
<policy domain="filter" rights="none" pattern="*" />
<policy domain="delegate" rights="none" pattern="HTTPS" />
<policy domain="delegate" rights="none" pattern="SHOW" />
<policy domain="delegate" rights="none" pattern="WIN" />
<policy domain="path" rights="none" pattern="@*"/>
```

- Check the "magick bytes" of all untrusted files before processing. Never assume that the file extension suffix, or mimetype is good enough. For example:

```
def assert_png(filename):
    """Ensure the file at a give path has the PNG magick-number
    header.  Throw an `AssertionError` if it does not match.
    """
    PNG_HEADER = [
        0x89, 0x50, 0x4E, 0x47,
        0x0D, 0x0A, 0x1A, 0x0A
    ]
    with open(filename, 'rb') as fd:
        file_header = list(fd.read(8))
    assert file_header == PNG_HEADER

try:
    assert_png(user_file)
    with Image(filename='png:'+user_file) as img:
        # ... do work ...
except AssertionError:
    # ... handle exception ...
```

- Ensure that any Python code is invoked with a low-privileged system user.
- Ensure filenames are sanitized.
- Ensure filenames are prefixed with coder protocol.

```
with Image(filename='png:input.png') as img:
    # ... do work ...
```

- Ensure error handling is in place. Expect *PolicyError* exceptions if a file-format was banned, and *ResourceLimitError* if the system is unable to allocate additional memory/disk resources. Both can be configured by the `policy.xml` listed above.

3.4 Reading images

There are several ways to open images:

- *To open an image file*
- *To read a input stream (file-like object) that provides an image binary*
- *To read a binary string that contains image*
- *To copy an existing image object*
- *To open an empty image*

All of these operations are provided by the constructor of *Image* class.

3.4.1 Open an image file

The most frequently used way is just to open an image by its filename. *Image*'s constructor can take the parameter named `filename`:

```
from __future__ import print_function
from wand.image import Image

with Image(filename='pikachu.png') as img:
    print('width =', img.width)
    print('height =', img.height)
```

Note: It must be passed by keyword argument exactly. Because the constructor has many parameters that are exclusive to each other.

There is a keyword argument named `file` as well, but don't confuse it with `filename`. While `filename` takes a string of a filename, `file` takes a input stream (file-like object).

3.4.2 Read a input stream

If an image to open cannot be located by a filename but can be read through input stream interface (e.g. opened by `os.popen()`, contained in `StringIO`, read by `urllib2.urlopen()`), it can be read by *Image* constructor's `file` parameter. It takes all file-like objects which implements `read()` method:

```
from __future__ import print_function
from urllib2 import urlopen
from wand.image import Image

response = urlopen('https://stylesha.re/minhee/29998/images/100x100')
try:
    with Image(file=response) as img:
        print('format =', img.format)
        print('size =', img.size)
finally:
    response.close()
```

In the above example code, `response` object returned by `urlopen()` function has `read()` method, so it also can be used as an input stream for a downloaded image.

3.4.3 Read a blob

If you have just a binary string (`str`) of the image, you can pass it into *Image* constructor's `blob` parameter to read:

```
from __future__ import print_function
from wand.image import Image

with open('pikachu.png') as f:
    image_binary = f.read()

with Image(blob=image_binary) as img:
    print('width =', img.width)
    print('height =', img.height)
```

It is a way of the lowest level to read an image. There will probably not be many cases to use it.

3.4.4 Clone an image

If you have an image already and have to copy it for safe manipulation, use `clone()` method:

```
from wand.image import Image

with Image(filename='pikachu.png') as original:
    with original.clone() as converted:
        converted.format = 'png'
        # operations on a converted image...
```

For some operations like format converting or cropping, there are safe methods that return a new image of manipulated result like *convert()* or slicing operator. So the above example code can be replaced by:

```
from wand.image import Image

with Image(filename='pikachu.png') as original:
    with original.convert('png') as converted:
        # operations on a converted image...
```

3.4.5 Hint file format

When it's read from a binary string or a file object, you can explicitly give the hint which indicates file format of an image to read — optional `format` keyword is for that:

```
from wand.image import Image

with Image(blob=image_binary, format='ico') as image:
    print(image.format)
```

New in version 0.2.1: The `format` parameter to *Image* constructor.

3.4.6 Open an empty image

To open an empty image, you have to set its width and height:

```
from wand.image import Image

with Image(width=200, height=100) as img:
    img.save(filename='200x100-transparent.png')
```

Its background color will be transparent by default. You can set `background` argument as well:

```
from wand.color import Color
from wand.image import Image

with Color('red') as bg:
    with Image(width=200, height=100, background=bg) as img:
        img.save(filename='200x100-red.png')
```

New in version 0.2.2: The `width`, `height`, and `background` parameters to *Image* constructor.

3.4.7 Open a Pseudo Image

A pseudo image can refer to any of ImageMagick's internal images that are accessable through coder protocols.

```
from wand.image import Image

with Image(width=100, height=100, pseudo='plasma:') as img:
    img.save(filename='100x100-plasma.png')
```

Commun Pseudo images

- `'canvas:COLOR'`, or `'xc:COLOR'`, where *COLOR* is any valid color value string.
- `'caption:TEXT'`, where *TEXT* is a string message.
- `'gradient:START-END'`, generates a blended gradient between two colors, where both *START* and *END* are color value strings.
- `'hald:'`, creates a Higher And Lower Dimension matrix table.
- `'inline:VALUE'`, where VALUE is a data-url / base64 string value.
- `'label:TEXT'`, where *TEXT* is a string message.
- `'pattern:LABEL'`, generates a repeating pattern, where *LABEL* is the pattern name. See Built-in Patterns
- `'plasma:'`, generates a plasma fractal image.
- `'radial-gradient:'`, similar to *gradient:*, but generates a gradual blend from center of the image.
- `'tile:FILENAME'`, generates a repeating tile effect from a given images, where *FILENAME* is the path of a source image.

A list of all pseudo images can be found at https://imagemagick.org/script/formats.php#pseudo

New in version 0.5.0: The `pseudo` parameter was added to the *Image* constructor.

3.4.8 Read Modifiers

Opening an image with the *filename* property allows for ImageMagick's Read Modifiers to be processed.

Single, or groups of, frames can be read without decoding all data. This can be useful to quick load the first page in a PDF:

```
with Image(filename='document.pdf[0]') as first_page:
    pass
```

Or a range of frames:

```
with Image(filename='animation.gif[0-11]') as first_dozen:
    pass
```

Or specific frames:

```
with Image(filename='animation.gif[0,2]') as first_and_third:
    pass
```

You can also use `[WxH]` format to resize the input image during read:

```
with Image(filename='logo.png[400x300]') as four_three_aspect:
    pass
```

Cropping an image can be achieved by following the `[WxH+x+y]` modifier:

```
with Image(filename='logo.png[100x100+50+75]') as sub_image:
    pass
```

3.5 Writing images

You can write an *Image* object into a file or a byte string buffer (blob) as format what you want.

3.5.1 Convert images to JPEG

If you wonder what is image's format, use `format` property.

```
>>> image.format
'JPEG'
```

The `format` property is writable, so you can convert images by setting this property.

```
from wand.image import Image

with Image(filename='pikachu.png') as img:
    img.format = 'jpeg'
    # operations to a jpeg image...
```

If you want to convert an image without any changes of the original, use *convert()* method instead:

```
from wand.image import Image

with Image(filename='pikachu.png') as original:
    with original.convert('jpeg') as converted:
        # operations to a jpeg image...
        pass
```

Note: Support for some of the formats are delegated to libraries or external programs. To get a complete listing of which image formats are supported on your system, use **identify** command provided by ImageMagick:

```
$ identify -list format
```

3.5.2 Save to file

In order to save an image to a file, use *save()* method with the keyword argument `filename`:

```
from wand.image import Image

with Image(filename='pikachu.png') as img:
    img.format = 'jpeg'
    img.save(filename='pikachu.jpg')
```

Note: The image format does not effect the file being saved, to save with a given colorspace use:

```
from wand.image import Image

with Image(filename='pikachu.jpg') as img:
    img.format = 'jpeg'
    img.save(filename='PNG24:pikachu.png')
```

3.5.3 Save to stream

You can write an image into a output stream (file-like object which implements `write()` method) as well. The parameter `file` takes a such object (it also is the first positional parameter of *save()* method).

For example, the following code converts `pikachu.png` image into JPEG, gzips it, and then saves it to `pikachu.jpg.gz`:

```
import gzip
from wand.image import Image

gz = gzip.open('pikachu.jpg.gz')
with Image(filename='pikachu.png') as img:
    img.format = 'jpeg'
    img.save(file=gz)
gz.close()
```

3.5.4 Get binary string

Want just a binary string of the image? Use *make_blob()* method so:

```
from wand.image import Image

with image(filename='pikachu.png') as img:
    img.format = 'jpeg'
    jpeg_bin = img.make_blob()
```

There's the optional `format` parameter as well. So the above example code can be simpler:

```
from wand.image import Image

with Image(filename='pikachu.png') as img:
    jpeg_bin = img.make_blob('jpeg')
```

3.6 Resizing and cropping

Creating thumbnails (by resizing images) and cropping are most frequent works about images. This guide explains ways to deal with sizes of images.

Above all, to get the current size of the image check *width* and *height* properties:

```
>>> from urllib.request import urlopen
>>> from wand.image import Image
>>> f = urlopen('http://pbs.twimg.com/profile_images/712673855341367296/WY6aLbBV_normal.
↪jpg')
>>> with Image(file=f) as img:
...     width = img.width
...     height = img.height
...
>>> f.close()
>>> width
48
>>> height
48
```

If you want the pair of (*width*, *height*), check *size* property also.

Note: These three properties are all readonly.

3.6.1 Resize images

It scales an image into a desired size even if the desired size is larger than the original size. ImageMagick provides so many algorithms for resizing. The constant *FILTER_TYPES* contains names of filtering algorithms.

See also:

ImageMagick Resize Filters Demonstrates the results of resampling three images using the various resize filters and blur settings available in ImageMagick, and the file size of the resulting thumbnail images.

Image.resize() method takes `width` and `height` of a desired size, optional `filter` (`'undefined'` by default which means IM will try to guess best one to use) and optional `blur` (default is 1). It returns nothing but resizes itself in-place.

```
>>> img.size
(500, 600)
>>> img.resize(50, 60)
>>> img.size
(50, 60)
```

3.6.2 Sample images

Although *Image.resize()* provides many `filter` options, it's relatively slow. If speed is important for the job, you'd better use *Image.sample()* instead. It works in similar way to *Image.resize()* except it doesn't provide `filter` and `blur` options:

```
>>> img.size
(500, 600)
>>> img.sample(50, 60)
>>> img.size
(50, 60)
```

3.6.3 Crop images

To extract a sub-rectangle from an image, use the *crop()* method. It crops the image in-place. Its parameters are `left`, `top`, `right`, `bottom` in order.

```
>>> img.size
(200, 300)
>>> img.crop(10, 20, 50, 100)
>>> img.size
(40, 80)
```

It can also take keyword arguments `width` and `height`. These parameters replace `right` and `bottom`.

```
>>> img.size
(200, 300)
>>> img.crop(10, 20, width=40, height=80)
>>> img.size
(40, 80)
```

There is an another way to crop images: slicing operator. You can crop an image by `[left:right, top:bottom]` with maintaining the original:

```
>>> img.size
(300, 300)
>>> with img[10:50, 20:100] as cropped:
...     print(cropped.size)
...
(40, 80)
>>> img.size
(300, 300)
```

Specifying `gravity` along with `width` and `height` keyword arguments allows a simplified cropping alternative.

```
>>> img.size
(300, 300)
>>> img.crop(width=40, height=80, gravity='center')
>>> img.size
(40, 80)
```

3.6.4 Transform images

Use this function to crop and resize an image at the same time, using ImageMagick geometry strings. Cropping is performed first, followed by resizing.

For example, if you want to crop your image to 300x300 pixels and then scale it by 2x for a final size of 600x600 pixels, you can call:

```
img.transform('300x300', '200%')
```

Other example calls:

```
# crop top left corner
img.transform('50%')

# scale height to 100px and preserve aspect ratio
img.transform(resize='x100')

# if larger than 640x480, fit within box, preserving aspect ratio
img.transform(resize='640x480>')

# crop a 320x320 square starting at 160x160 from the top left
img.transform(crop='320+160+160')
```

See also:

ImageMagick Geometry Specifications Cropping and resizing geometry for the `transform` method are specified according to ImageMagick's geometry string format. The ImageMagick documentation provides more information about geometry strings.

3.6.5 Seam carving (also known as *content-aware resizing*)

New in version 0.3.0.

Seam carving is an algorithm for image resizing that functions by establishing a number of *seams* (paths of least importance) in an image and automatically removes seams to reduce image size or inserts seams to extend it.

In short: you can magickally resize images without distortion! See the following examples:

Original	Resized
	
Cropped	**Seam carving**
	

You can easily rescale images with seam carving using Wand: use *Image.liquid_rescale()* method:

```
>>> image = Image(filename='seam.jpg')
>>> image.size
(320, 234)
>>> with image.clone() as resize:
```

(continues on next page)

(continued from previous page)

```
...     resize.resize(234, 234)
...     resize.save(filename='seam-resize.jpg')
...     resize.size
...
(234, 234)
>>> with image[:234, :] as crop:
...     crop.save(filename='seam-crop.jpg')
...     crop.size
...
(234, 234)
>>> with image.clone() as liquid:
...     liquid.liquid_rescale(234, 234)
...     liquid.save(filename='seam-liquid.jpg')
...     liquid.size
...
(234, 234)
```

Note: It may raise *MissingDelegateError* if your ImageMagick is configured `--without-lqr` option. In this case you should recompile ImageMagick.

See also:

Seam carving — Wikipedia The article which explains what seam carving is on Wikipedia.

Note: The image `seam.jpg` used in the above example is taken by D. Sharon Pruitt and licensed under CC-BY-2.0. It can be found the original photography from Flickr.

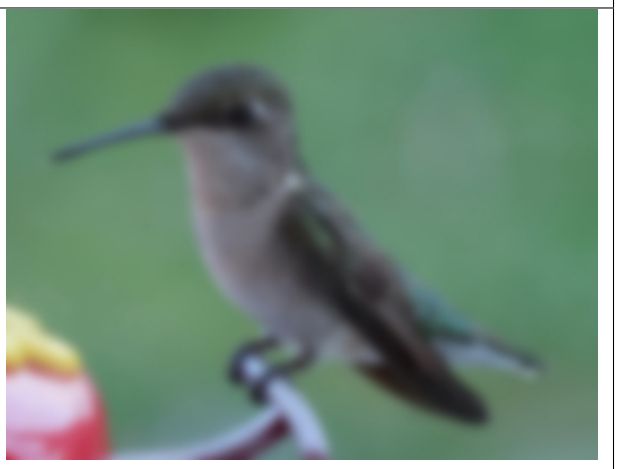
3.7 Image Effects

3.7.1 Blur

New in version 0.4.5.

Basic blur operation. The `radius` argument defines the size of the area to sample, and the `sigma` defines the standard deviation. For all blur based methods, the best results are given when the `radius` is larger than `sigma`. However, if `radius` is omitted, or zero valued, the value will be selected based off the given `sigma` property.

```
with Image(filename="hummingbird.jpg") as img:
    img.blur(radius=0, sigma=3)
    img.save(filename="effect-blur.jpg")
```

Original	Blur
	

Adaptive Blur

New in version 0.5.3.

This method blurs less intensely around areas of an image with detectable edges, and blurs more intensely for areas without edges. The `radius` should always be larger than the `sigma` (standard deviation).

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.adaptive_blur(radius=8, sigma=4)
    img.save(filename="effect-adaptive-blur.jpg")
```

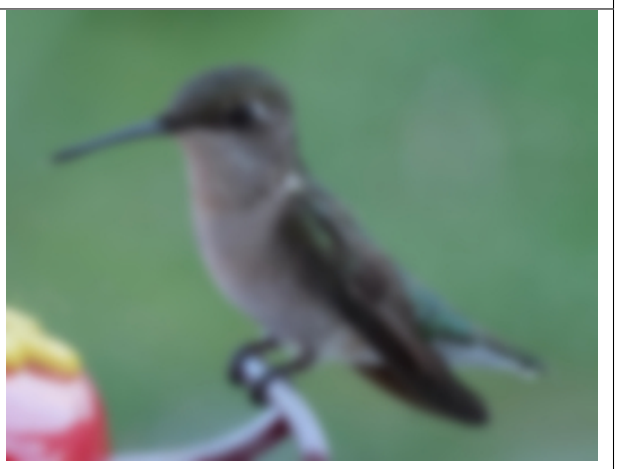
Original	Adaptive Blur
	

Gaussian Blur

New in version 0.3.3.

Smooths images by performing a Gaussian function. The `sigma` argument is used to define the standard deviation.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.gaussian_blur(sigma=3)
    img.save(filename="effect-gaussian-blur.jpg")
```

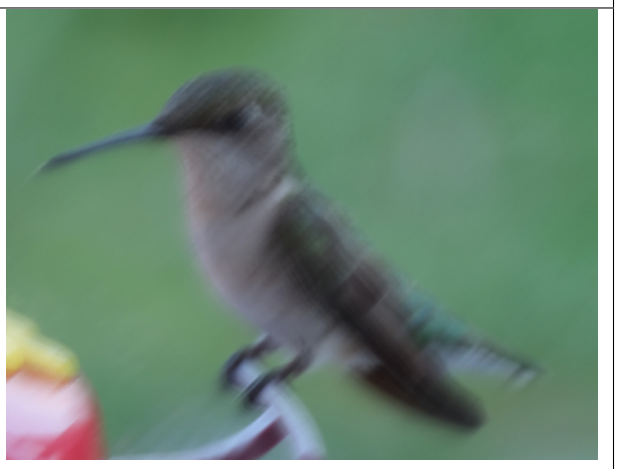
Original	Gaussian Blur
	

Motion Blur

New in version 0.5.4.

Performs a Gaussian blur operation along a linear direction to simulate a motion effect. The `radius` argument should always be larger than the `sigma` argument, but if the `radius` is not given (or `0` value) the radius value is selected for you.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.motion_blur(radius=16, sigma=8, angle=-45)
    img.save(filename="effect-motion-blur.jpg")
```

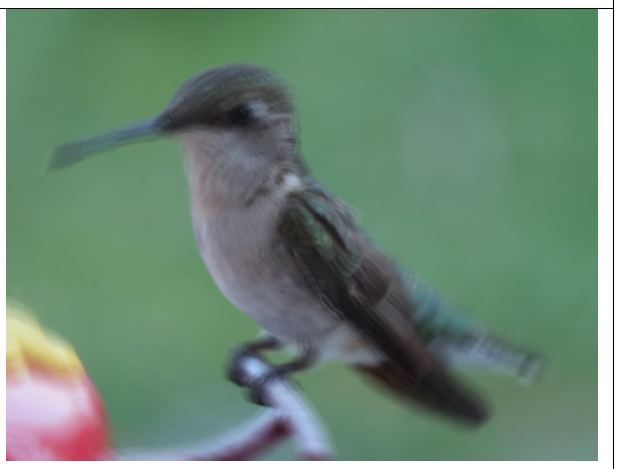
Original	Motion Blur
	

Rotational Blur

New in version 0.5.4.

This method simulates a motion blur by rotating at the center of the image. The larger the angle, the more extreme the blur will be. Unlike the other blur methods, there is no `radius` or `sigma` arguments. The `angle` parameter can be between `0°` and `360°` degrees with `0°` having no effect.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.rotational_blur(angle=5)
    img.save(filename="effect-rotational-blur.jpg")
```

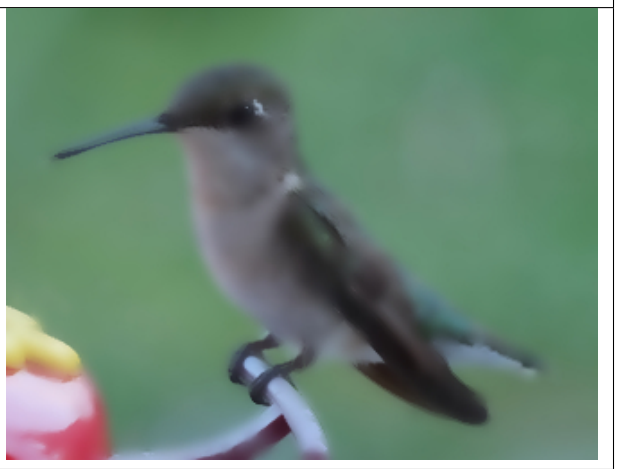
Original	Rotational Blur
	

Selective Blur

New in version 0.5.3.

Similar to *Image.blur()* method, this method will only effect parts of the image that have a contrast below a given quantum threshold.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.selective_blur(radius=8,
                       sigma=3,
                       threshold=0.25 * img.quantum_range)
    img.save(filename="effect-selective-blur.jpg")
```

Original	Selective Blur
	

3.7.2 Despeckle

New in version 0.5.0.

Despeckling is one of the many techniques you can use to reduce noise on a given image. Also see *Enhance*.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.despeckle()
    img.save(filename="effect-despeckle.jpg")
```

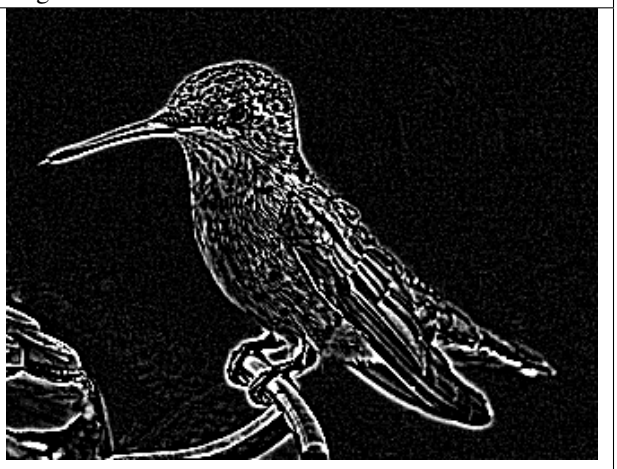
Original	Despeckle
	

3.7.3 Edge

New in version 0.5.0.

Detects edges on black and white images with a simple convolution filter. If used with a color image, the transformation will be applied to each color-channel.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.transform_colorspace('gray')
    img.edge(radius=1)
    img.save(filename="effect-edge.jpg")
```

Original	Edge
	

3.7.4 Emboss

New in version 0.5.0.

Generates a 3D effect that can be described as print reliefs. Like *Edge*, best results can be generated with grayscale image. Also see *Shade*.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.transform_colorspace('gray')
    img.emboss(radius=3.0, sigma=1.75)
    img.save(filename="effect-emboss.jpg")
```

Original	Emboss
	

3.7.5 Kuwahara

New in version 0.5.5.

Warning: Class method only available with ImageMagick 7.0.8-41 or greater.

The *kuwahara()* method applies a smoothing filter to reduce noise in an image, but also preserves edges.

```
from image.wand import Image

with Image(filename="hummingbird.jpg") as img:
    img.kuwahara(radius=2, sigma=1.5)
    img.save(filename="effect-kuwahara.jpg")
```

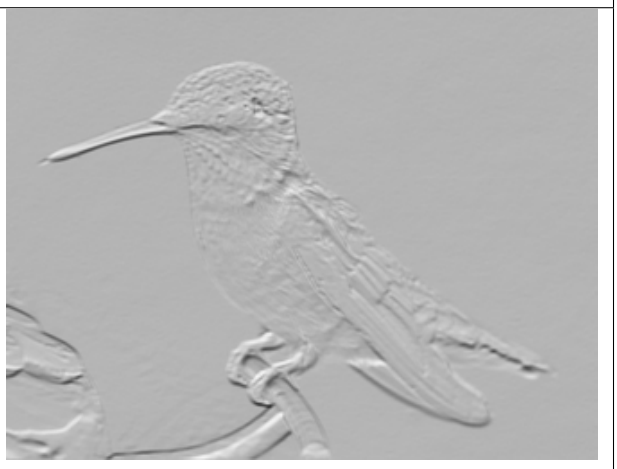
Original	Kuwahara
	

3.7.6 Shade

New in version 0.5.0.

Creates a 3D effect by simulating light from source where `aziumth` controls the X/Y angle, and `elevation` controls the Z angle. You can also determine of the resulting image should be transformed to grayscale by passing `gray` boolean.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.shade(gray=True,
              azimuth=286.0,
              elevation=45.0)
    img.save(filename="effect-shade.jpg")
```

Original	Shade
	

3.7.7 Sharpen

New in version 0.5.0.

Convolves an image with a Gaussian operator to enhance blurry edges into a more distinct "sharp" edge. The `radius` should always be larger than `sigma` value. The radius value will be calculated automatically if only `sigma` is given.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.sharpen(radius=8, sigma=4)
    img.save(filename="effect-sharpen.jpg")
```

Original	Sharpen
	

Adaptive Sharpen

New in version 0.5.3.

Just like *Image.sharpen()*, adaptive sharpen uses a convolve & Gaussian operations to sharpen blurred images. However, the effects of *Image.adaptive_sharpen()* are more intense around pixels with detectable edges, and less farther away from edges. In the example below, notice the visible changes around the edge of the feathers, and limited changes in the out-of-focus background.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.adaptive_sharpen(radius=8, sigma=4)
    img.save(filename="effect-adaptive-sharpen.jpg")
```

Original	Adaptive Sharpen
	

Unsharp Mask

New in version 0.3.4.

Identical to *Image.sharpen* method, but gives users the control to blend between filter & original (`amount` parameter), and the `threshold`. When the `amount` value is greater than `1.0` more if the sharpen filter is applied, and less if the value is under `1.0`. Values for `threshold` over `0.0` reduce the sharpens.

```
with Image(filename="hummingbird.jpg") as img:
    img.unsharp_mask(radius=10,
                     sigma=4,
                     amount=1,
                     threshold=0)
    img.save(filename="effect-unsharp-mask.jpg")
```

Original	Unsharp Mask
	

3.7.8 Spread

New in version 0.5.3.

Spread replaces each pixel with the a random pixel value found near by. The size of the area to search for a new pixel can be controlled by defining a radius.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as img:
    img.spread(radius=8.0)
    img.save(filename="effect-spread.jpg")
```

Original	Spread
	

3.8 Special Effects (FX)

3.8.1 Add Noise

New in version 0.5.3.

You can add random noise to an image. This operation can be useful when applied before a blur operation to defuse an image. The types of noise can be any of the following.

- `'gaussian'`
- `'impulse'`
- `'laplacian'`
- `'multiplicative_gaussian'`
- `'poisson'`
- `'random'`
- `'uniform'`

The amount of noise can be adjusted by passing an *attenuate* kwarg where the value can be between *0.0* and *1.0*.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.noise("laplacian", attenuate=1.0)
    img.save(filename="fx-noise.jpg")
```

Original	Noise
	

3.8.2 Blue Shift

New in version 0.5.3.

Gently mutes colors by shifting blue values by a factor. This produces a nighttime scene with a moonlight effect.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.blue_shift(factor=1.25)
    img.save(filename="fx-blue-shift.jpg")
```

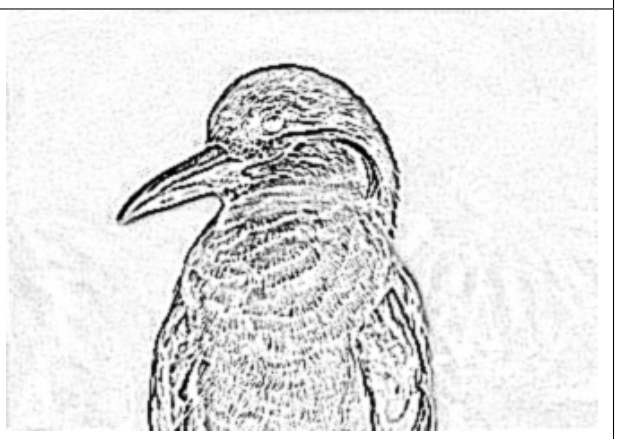
Original	Blue Shift
	

3.8.3 Charcoal

New in version 0.5.3.

One of the artistic simulations, *charcoal()* can emulate a drawing on paper.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.charcoal(radius=1.5, sigma=0.5)
    img.save(filename="fx-charcoal.jpg")
```

Original	Charcoal
	

3.8.4 Color Matrix

New in version 0.5.3.

This method allows you to recalculate color values by applying a matrix transform. A matrix can be up to a 6x6 grid where each column maps to a color channel to reference, and each row represents a color channel to effect. Usually `red`, `green`, `blue`, `n/a`, `alpha`, and a constant (a.k.a offset) for RGB images, or `cyan`, `yellow`, `magenta`, `black`, `alpha`, and a constant for CMYK images.

For example: To swap Red & Blue channels.

$$
\begin{aligned}
red' &= 0.0 * red + 0.0 * green + 1.0 * blue \\
green' &= 0.0 * red + 1.0 * green + 0.0 * blue \\
blue' &= 1.0 * red + 0.0 * green + 0.0 * blue
\end{aligned}
$$

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    matrix = [[0, 0, 1],
              [0, 1, 0],
              [1, 0, 0]]
    img.color_matrix(matrix)
    img.save(filename="fx-color-matrix.jpg")
```

Original	Color Matrix
	

3.8.5 Colorize

New in version 0.5.3.

Blends an image with a constant color. With *Image.colorize()*, the `color` parameter is the constant color to blend, and the `alpha` is a mask-color to control the blend rate per color channel.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.colorize(color="yellow", alpha="rgb(10%, 0%, 20%)")
    img.save(filename="fx-colorize.jpg")
```

Original	Colorize
	

3.8.6 FX

New in version 0.4.1.

FX special effects are a powerful "micro" language to work with. Simple functions & operators offer a unique way to access & manipulate image data. The *fx()* method applies a FX expression, and generates a new *Image* instance.

We can create a custom DIY filter that will turn the image black & white, except colors with a hue above 324°, or below 36°.

```
from wand.image import Image

fx_filter="(hue > 0.9 || hue < 0.1) ? u : lightness"

with Image(filename="inca_tern.jpg") as img:
    with img.fx(fx_filter) as filtered_img:
       filtered_img.save(filename="fx-fx.jpg")
```

Original	FX
	

3.8.7 Implode

New in version 0.5.2.

This special effect "pulls" pixels into the middle of the image. The `amount` argument controls the range of pixels to pull towards the center. With ImageMagick 7, you can define the pixel interpolate methods. See *PIXEL_INTERPOLATE_METHODS*.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.implode(amount=0.35)
    img.save(filename="fx-implode.jpg")
```

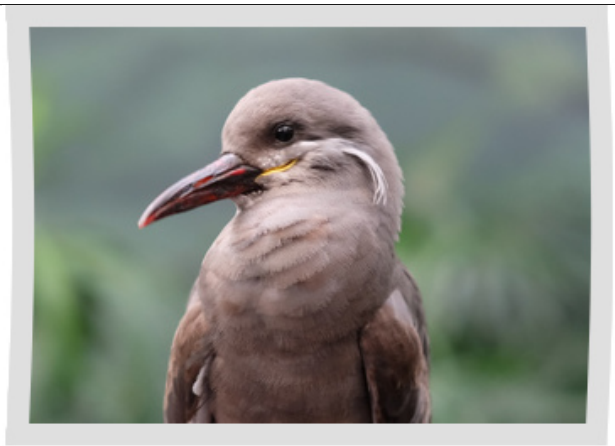
Original	Implode
	

3.8.8 Polaroid

New in version 0.5.4.

Wraps am image in a white board, and a slight shadow to create the special effect of a Polaroid print.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.polaroid()
    img.save(filename="fx-polaroid.jpg")
```

Original	Polaroid
	

3.8.9 Sepia Tone

New in version 0.5.7.

We can simulate old-style silver based chemical photography printing by applying sepia toning to images.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.sepia_tone(threshold=0.8)
    img.save(filename="fx-sepia-tone.jpg")
```

Original	Sepia Tone
	

3.8.10 Sketch

New in version 0.5.3.

Simulates an artist sketch drawing. Also see *Charcoal*.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.transform_colorspace("gray")
    img.sketch(0.5, 0.0, 98.0)
    img.save(filename="fx-sketch.jpg")
```

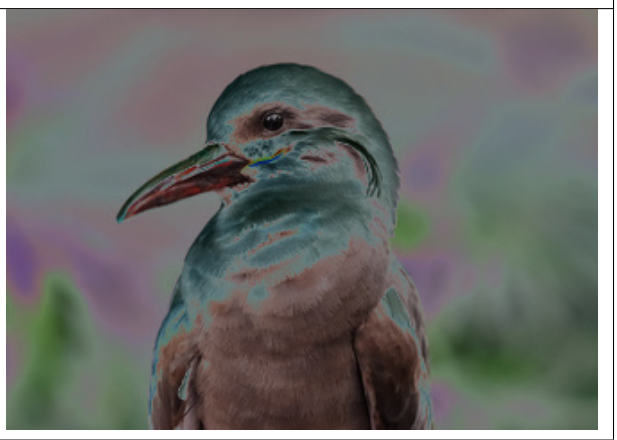
Original	Sketch
	

3.8.11 Solarize

New in version 0.5.3.

Creates a “burned” effect on the image by replacing pixel values above a defined threshold with a negated value.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.solarize(threshold=0.5 * img.quantum_range)
    img.save(filename="fx-solarize.jpg")
```

Original	Solarize
	

3.8.12 Stereogram

New in version 0.5.4.

Also known as "anaglyph", this class method takes two *Image* instances (one for each eye), and creates a 3d image by separating the Red & Cyan.

```
from wand.image import Image

with Image(filename="left_camera.jpg") as left_eye:
    with Image(filename="right_camera.jpg") as right_eye:
        with Image.stereogram(left=left_eye,
                              right=right_eye) as img:
            img.save(filename="fx-stereogram.jpg")
```

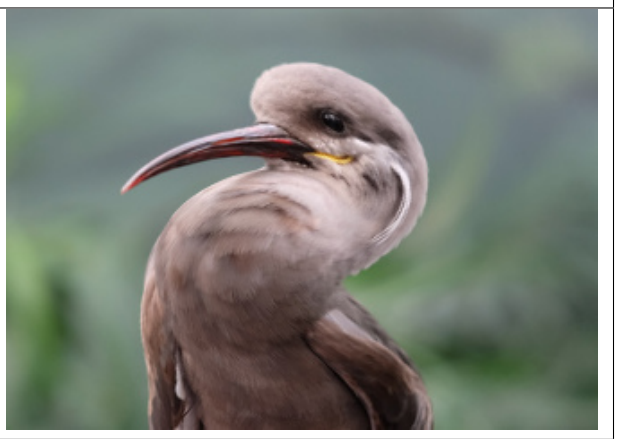
Stereogram

3.8.13 Swirl

New in version 0.5.7.

Creates a visual whirlpool effect by rotating pixels around the center of the image. The value of `degree` controls the amount, and distance, of pixels to rotate around the center. Negative degrees move pixels clockwise, and positive values move pixels counter-clockwise.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.swirl(degree=-90)
    img.save(filename='fx-swirl.jpg')
```

Original	Swirl
	

3.8.14 Tint

New in version 0.5.3.

Tint colorizes midtones of an image by blending the given `color`. The `alpha` parameter controls how the blend is effected between color channels. However, this can be tricky to use, so when in doubt, use a `alpha="gray(50)"` argument.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.tint(color="yellow", alpha="rgb(40%, 60%, 80%)")
    img.save(filename="fx-tint.jpg")
```

Original	Tint
	

3.8.15 Vignette

New in version 0.5.2.

Creates a soft & blurry ellipse on the image. Use the `x` & `y` arguments to control edge of the ellipse inset from the image border, and `radius` & `sigma` argument to control the blurriness. The `radius` can be omitted if you wish ImageMagick to select a value from the defined `sigma` value.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.vignette(sigma=3, x=10, y=10)
    img.save(filename="fx-vignette.jpg")
```

Original	Vignette
	

3.8.16 Wave

New in version 0.5.2.

Creates a ripple effect within the image. With ImageMagick 7, you can define the pixel interpolate methods. See *PIXEL_INTERPOLATE_METHODS*.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.wave(amplitude=img.height / 32,
             wave_length=img.width / 4)
    img.save(filename="fx-wave.jpg")
```

Original	Wave
	

3.8.17 Wavelet Denoise

New in version 0.5.5.

This method removes noise by applying a wavelet transform. The `threshold` argument should be a value between `0.0` & *quantum_range*, and the `softness` argument should be a value between `0.0` & `1.0`.

```
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.wavelet_denoise(threshold=0.05 * img.quantum_range,
                        softness=0.0)
    img.save(filename="fx-wavelet-denoise.jpg")
```

Original	Wavelet Denoise
	

3.9 Transformation

Note: The image `transform.jpg` used in this docs is taken by Megan Trace, and licensed under CC BY-NC 2.0. It can be found the original photography from Flickr.

3.9.1 Enhance

New in version 0.5.0.

Reduce the noise of an image by applying an auto-filter. Also see *Despeckle*.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.enhance()
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-enhance.jpg")
```

3.9.2 Flip and flop

New in version 0.3.0.

You can make a mirror image by reflecting the pixels around the central x- or y-axis. For example, where the given image `transform.jpg`:

The following code flips the image using *Image.flip()* method:

```
from wand.image import Image

with Image(filename='transform.jpg') as image:
    with image.clone() as flipped:
        flipped.flip()
        flipped.save(filename='transform-flipped.jpg')
```

The image `transform-flipped.jpg` generated by the above code looks like:

As like *flip()*, *flop()* does the same thing except it doesn't make a vertical mirror image but horizontal:

```
from wand.image import Image

with Image(filename='transform.jpg') as image:
    with image.clone() as flopped:
```

(continues on next page)

(continued from previous page)

```
        flopped.flop()
        flopped.save(filename='transform-flopped.jpg')
```

The image `transform-flopped.jpg` generated by the above code looks like:

3.9.3 Rotation

New in version 0.1.8.

Image object provides a simple method to rotate images: *rotate()*. It takes a `degree` which can be 0 to 359. (Actually you can pass 360, 361, or more but it will be the same to 0, 1, or more respectively.)

For example, where the given image `transform.jpg`:

The below code makes the image rotated 90° to right:

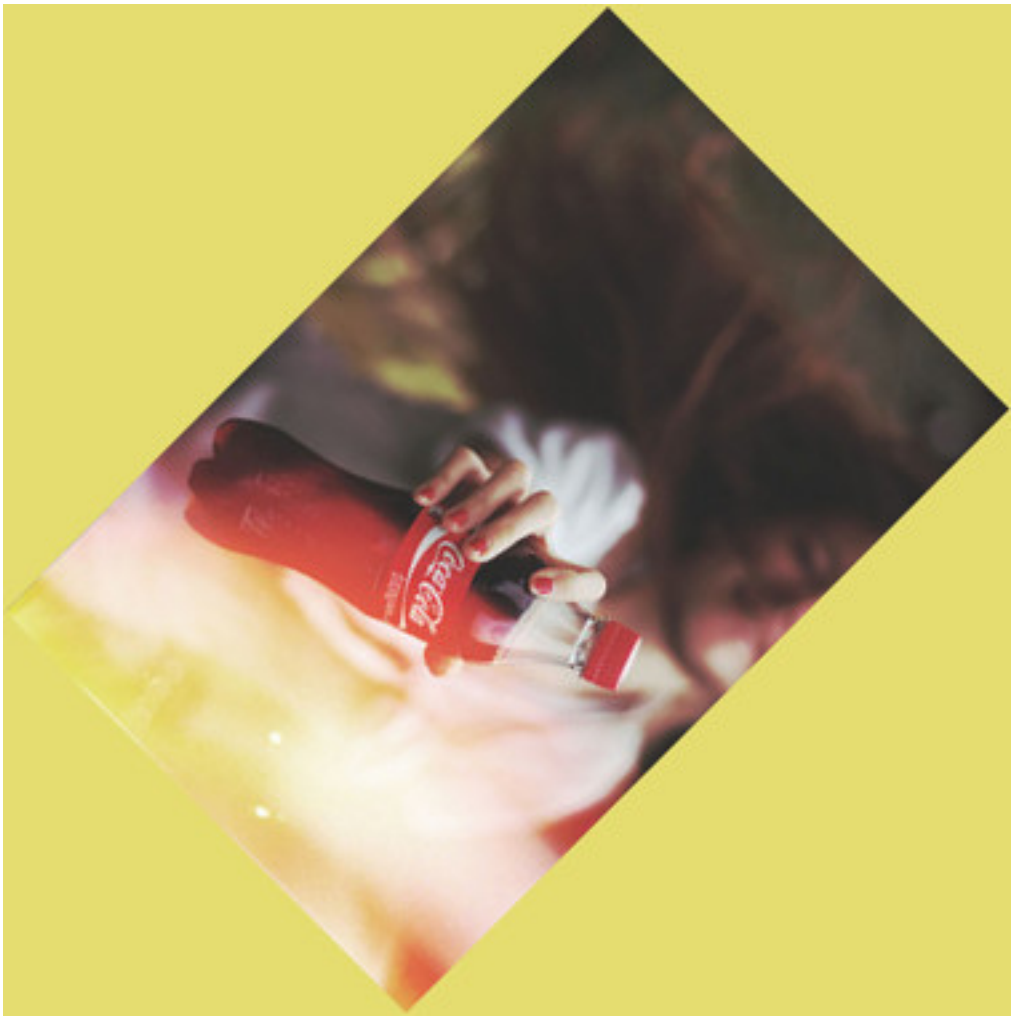
```
from wand.image import Image

with Image(filename='transform.jpg') as image:
    with image.clone() as rotated:
        rotated.rotate(90)
        rotated.save(filename='transform-rotated-90.jpg')
```

The generated image `transform-rotated-90.jpg` looks like:

If `degree` is not multiples of 90, the optional parameter `background` will help (its default is transparent):

```
from wand.color import Color
from wand.image import Image

with Image(filename='transform.jpg') as image:
    with image.clone() as rotated:
        rotated.rotate(135, background=Color('rgb(229,221,112)'))
        rotated.save(filename='transform-rotated-135.jpg')
```

The generated image `transform-rotated-135.jpg` looks like:



3.9.4 Statistic

New in version 0.5.3.

Similar to *Spread*, but replaces each pixel with the result of a mathematical operation performed against neighboring pixel values.

The type of statistic operation can be any of the following.

- `'gradient'`
- `'maximum'`
- `'mean'`
- `'median'`
- `'minimum'`
- `'mode'`
- `'nonpeak'`
- `'root_mean_square'`
- `'standard_deviation'`

The size neighboring pixels to evaluate can be defined by passing `width`, and `height` kwargs.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.statistic("median", width=8, height=5)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-statistic.jpg")
```

3.10 Colorspace

3.10.1 Image types

Every *Image* object has *type* property which identifies its colorspace. The value can be one of *IMAGE_TYPES* enumeration, and set of its available values depends on its `format` as well. For example, `'grayscale'` isn't available on JPEG.

```
>>> from wand.image import Image
>>> with Image(filename='wandtests/assets/bilevel.gif') as img:
...     img.type
...
'bilevel'
>>> with Image(filename='wandtests/assets/sasha.jpg') as img2:
...     img2.type
...
'truecolor'
```

You can change this value:

```
with Image(filename='wandtests/assets/bilevel.gif') as img:
    img.type = 'truecolor'
    img.save(filename='truecolor.gif')
```

See also:

-type — ImageMagick: command-line-Options Corresponding command-line option of `convert` program.

3.10.2 Enable alpha channel

You can find whether an image has alpha channel and change it to have or not to have the alpha channel using *alpha_channel* property, which is preserving a `bool` value.

```
>>> with Image(filename='wandtests/assets/sasha.jpg') as img:
...     img.alpha_channel
...
False
>>> with Image(filename='wandtests/assets/croptest.png') as img:
...     img.alpha_channel
...
True
```

It's a writable property:

```
with Image(filename='wandtests/assets/sasha.jpg') as img:
    img.alpha_channel = True
```

3.11 Color Enhancement

3.11.1 Evaluate Expression

New in version 0.4.1.

Pixel channels can be manipulated by applying an arithmetic, relational, or logical expression. See *EVALUATE_OPS* for a list of valid operations.

For example, when given image `enhancement.jpg`:

We can reduce the amount of data in the blue channel by applying the right-shift binary operator, and increase data in the right channel with left-shift operator:

```
from wand.image import Image

with Image(filename='enhancement.jpg') as img:
    # B >> 1
    img.evaluate(operator='rightshift', value=1, channel='blue')
    # R << 1
    img.evaluate(operator='leftshift', value=1, channel='red')
```

3.11.2 Function Expression

New in version 0.4.1.

Similar to *evaluate()*, *function()* applies a multi-argument function to pixel channels. See *FUNCTION_TYPES* for a list of available function formulas.

For example, when given image `enhancement.jpg`:

We can apply a **Sinusoid** function with the following:

```
from wand.image import Image

with Image(filename='enhancement.jpg') as img:
    frequency = 3
    phase_shift = -90
    amplitude = 0.2
    bias = 0.7
    img.function('sinusoid', [frequency, phase_shift, amplitude, bias])
```

3.11.3 Gamma

New in version 0.4.1.

Gamma correction allows you to adjust the luminance of an image. Resulting pixels are defined as `pixel^(1/gamma)`. The value of `gamma` is typically between 0.8 & 2.3 range, and value of 1.0 will not affect the resulting image.

The `level()` method can also adjust `gamma` value.

For example, when given image `enhancement.jpg`:

We can step through 4 pre-configured gamma correction values with the following:

```
from wand.image import Image

with Image(filename='enhancement.jpg') as img_src:
    for Y in [0.8, 0.9, 1.33, 1.66]:
        with Image(img_src) as img_cpy:
            img_cpy.gamma(Y)
```


gamma: 0.8
gamma: 0.9
gamma: 1.33
gamma: 1.66

3.11.4 Level

New in version 0.4.1.

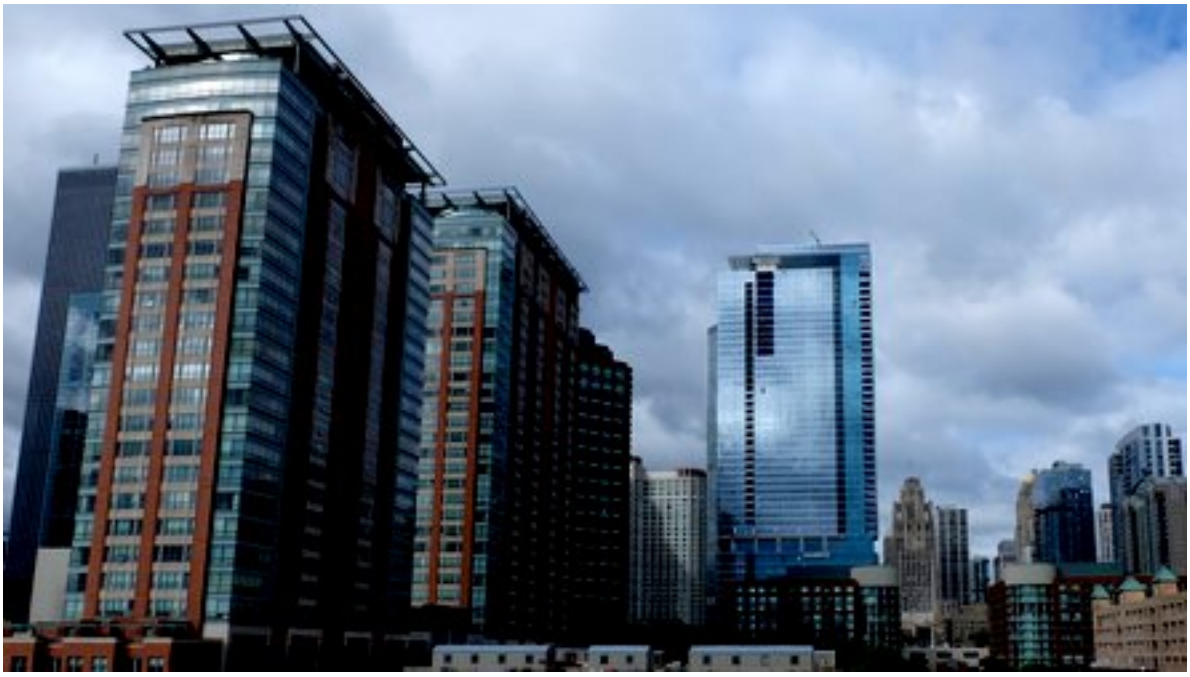
Black & white boundaries of an image can be controlled with `level()` method. Similar to the `gamma()` method, mid-point levels can be adjusted with the `gamma` keyword argument.

The `black` and `white` point arguments are expecting values between 0.0 & 1.0 which represent percentages.

For example, when given image `enhancement.jpg`:

We can adjust the level range between 20% & 90% with slight mid-range increase:

```
from wand.image import Image

with Image(filename='enhancement.jpg') as img:
    img.level(0.2, 0.9, gamma=1.1)
    img.save(filename='enhancement-level.jpg')
```



3.12 Distortion

ImageMagick provides several ways to distort an image by applying various transformations against user-supplied arguments. In Wand, the method *Image.distort* is used, and follows a basic function signature of:

```
with Image(...) as img:
    img.distort(method, arguments)
```

Where `method` is a string provided by *DISTORTION_METHODS*, and `arguments` is a list of doubles. Each `method` parses the `arguments` list differently. For example:

```
# Arc can have a minimum of 1 argument
img.distort('arc', (degree, ))
# Affine 3-point will require 6 arguments
points = (x1, y1, x2, y2,
          x3, y3, x4, y4,
          x5, y5, x6, y6)
img.distort('affine', points)
```

A more complete & detailed overview on distortion can be found in Distorting Images usage article by Anthony Thyssen.

3.12.1 Controlling Resulting Images

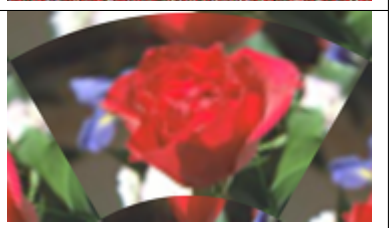
Virtual Pixels

When performing distortion on raster images, the resulting image often includes pixels that are outside original bounding raster. These regions are referred to as vertical pixels, and can be controlled by setting *Image.virtual_pixel* to any value defined in *VIRTUAL_PIXEL_METHOD*.

Virtual pixels set to `'transparent'`, `'black'`, or `'white'` are the most common, but many users prefer use the existing background color.

```
with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = img[70, 46]
    img.virtual_pixel = 'background'
    img.distort('arc', (60, ))
```

Other *virtual_pixel* values can create special effects.

Virtual Pixel	Example
dither	
edge	
mirror	
random	
tile	

Matte Color

Some distortion transitions can not be calculated in the virtual-pixel space. Either being invalid, or **NaN** (not-a-number). You can define how such a pixel should be represented by setting the *Image.matte_color* property.

```
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.matte_color = Color('ORANGE')
    img.virtual_pixel = 'tile'
    args = (0, 0, 30, 60, 140, 0, 110, 60,
            0, 92, 2, 90, 140, 92, 138, 90)
    img.distort('perspective', args)
```

Rendering Size

Setting the `'distort:viewport'` artifact allows you to define the size, and offset of the resulting image:

```
img.artifacts['distort:viewport'] = '300x200+50+50'
```

Setting the `'distort:scale'` artifact will resizing the final image:

```
img.artifacts['distort:scale'] = '75%'
```

3.12.2 Affine

Affine distortion performs a shear operation. The arguments are similar to perspective, but only need a pair of 3 points, or 12 real numbers.

```
src1$_x$, src1$_y$, dst1$_x$, dst1$_y$,
src2$_x$, src2$_y$, dst2$_x$, dst2$_y$,
src3$_x$, src3$_y$, dst3$_x$, dst3$_y$,
...
```

For example:

```
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    args = (
        10, 10, 15, 15,  # Point 1: (10, 10) => (15,  15)
        139, 0, 100, 20, # Point 2: (139, 0) => (100, 20)
        0, 92, 50, 80    # Point 3: (0,  92) => (50,  80)
    )
    img.distort('affine', args)
```

3.12.3 Affine Projection

Affine projection is identical to *Scale Rotate Translate*, but requires exactly 6 real numbers for the distortion arguments.

```
Scale$_x$, Rotate$_x$, Rotate$_y$, Scale$_y$, Translate$_x$, Translate$_y$
```

For example:

```
from collections import namedtuple
from wand.color import Color
from wand.image import Image

Point = namedtuple('Point', ['x', 'y'])

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    rotate = Point(0.1, 0)
    scale = Point(0.7, 0.6)
    translate = Point(5, 5)
    args = (
        scale.x, rotate.x, rotate.y,
        scale.y, translate.x, translate.y
    )
    img.distort('affine_projection', args)
```

3.12.4 Arc

Arc distortion curves the image around the center point of an image. The arguments are:

```
ArcAngle, RotateAngle, TopRadius, BottomRadius
```

Where *ArcAngle* is the only required arguments, and the rest are optional.

For example:

```
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    args = (
        270,  # ArcAngle
        45,   # RotateAngle
```

(continues on next page)

(continued from previous page)

```
    )
    img.distort('arc', args)
```

3.12.5 Barrel

Barrel distortion attempts to correct spherical distortion caused by camera lenses. It operates with four constant coefficient values *A*, *B*, *C*, & *D* mapped to the images EXIF meta-data. Usually camera, lens, and zoom attributes. The equation for `barrel` distortion is:

$$R_{src} = r * \left(A * r^3 + B * r^2 + C * r + D\right)$$

Where `r` is the destination radius. The arguments for the distortion are:

```
A B C D X Y
```

Where `X` & `Y` are optional center coordinates.

For example:

```
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    args = (
        0.2,  # A
        0.0,  # B
        0.0,  # C
        1.0,  # D
    )
    img.distort('barrel', args)
```

3.12.6 Barrel Inverse

The barrel inverse distortion has the same arguments as the barrel distortion, but performs a different equation.

$$R_{src} = \frac{r}{(A * r^3 + B * r^2 + C * r + D)}$$

It does not reverse, or undo, the effects of the barrel distortion.

For example:

```
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    args = (
        0.0,   # A
        0.0,   # B
        -0.5,  # C
        1.5,   # D
    )
    img.distort('barrel_inverse', args)
```

3.12.7 Bilinear

The bilinear distortion is similar to perspective distortion, but evenly distributes pixels rather than compress & reduce shrinking areas. This means that shrunk areas of the image will appear flat with bilinear as opposed to farther away. The arguments are:

```
src1$_x$, src1$_y$, dst1$_x$, dst1$_y$,
src2$_x$, src2$_y$, dst2$_x$, dst2$_y$,
src3$_x$, src3$_y$, dst3$_x$, dst3$_y$,
src4$_x$, src4$_y$, dst4$_x$, dst4$_y$,
...
```

Unlike other inverse methods, bilinear distortion effects are directional. Meaning you can undo a distortion previously applied. For example:

```
from itertools import chain
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
```

(continues on next page)

(continued from previous page)

```
    img.virtual_pixel = 'background'
    source_points = (
        (0, 0),
        (140, 0),
        (0, 92),
        (140, 92)
    )
    destination_points = (
        (14, 4.6),
        (126.9, 9.2),
        (0, 92),
        (140, 92)
    )
    order = chain.from_iterable(zip(source_points, destination_points))
    arguments = list(chain.from_iterable(order))
    img.distort('bilinear_forward', arguments)
```

And then reverted with by swapping the destination with source coordinates, and using `'bilinear_reverse'`:

```
order = chain.from_iterable(zip(destination_points, source_points))
arguments = list(chain.from_iterable(order))
img.distort('bilinear_reverse', arguments)
```

3.12.8 Cylinder & Plane

Cylinder 2 plane is a radial projection to correct common field of vision (fov) distortions. The arguments are:

```
FovAngle, CenterX, CenterY, FovOutput, DestCenterX, DestCenterY
```

Where only the first argument is required, and the rest are optional. The *FovAngle* value can be roughly calculated by:

$$FovAngle = \frac{LensFocalLength}{FilmWidth} * \frac{180}{\pi}$$

The `'plane_2_cylinder'` is the inverted behavior. The arguments are:

```
FovAngle, CenterX, CenterY
```

For (a rather poor) example:

```
import math
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    lens = 60
    film = 35
    args = (
        lens/film * 180/math.pi,
    )
    img.distort('plane_2_cylinder', args)
```

And the inverse:

```
img.distort('cylinder_2_plane', args)
```

3.12.9 Perspective

Perspective distortion requires 4 pairs of points which is a total of 16 doubles. The order of the `arguments` are groups of source & destination coordinate pairs.

```
src1$_x$, src1$_y$, dst1$_x$, dst1$_y$,
src2$_x$, src2$_y$, dst2$_x$, dst2$_y$,
src3$_x$, src3$_y$, dst3$_x$, dst3$_y$,
src4$_x$, src4$_y$, dst4$_x$, dst4$_y$,
...
```

For example:

```
from itertools import chain
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
```

(continues on next page)

(continued from previous page)

```
    source_points = (
        (0, 0),
        (140, 0),
        (0, 92),
        (140, 92)
    )
    destination_points = (
        (14, 4.6),
        (126.9, 9.2),
        (0, 92),
        (140, 92)
    )
    order = chain.from_iterable(zip(source_points, destination_points))
    arguments = list(chain.from_iterable(order))
    img.distort('perspective', arguments)
```

3.12.10 Polar & Depolar

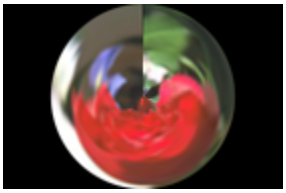
Polar distortion is similar to arc distort method, but does not attempt to preserve aspect ratios.

`Radius$_max$, Radius$_min$, Center$_x$, Center$_y$, Angle$_start$, Angle$_end$`

All the arguments are optional, and an argument of `0` will use the distance of the center to the closet edge as the default radius.

For example:

```
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.virtual_pixel = 'black'
    img.distort('polar', (0,))
```



For depolar distortion, the arguments are the same. However to revert the image previously distorted with nearest edge (argument `0`), use `-1` for ImageMagick to calculate the radius distance from the middle to the farthest edge.

For example:

```
from wand.image import Image

with Image(filename='distort-polar.png') as img:
    img.virtual_pixel = 'horizontal_tile'
    img.distort('depolar', (-1,))
```

3.12.11 Polynomial

Polynomial distortion requires at least four sets of coordinates & control points, and will perform a standard polynomial equation. However, the first distortion argument is reserved to define the `Order`, or `Complexity` of the two dimensional equation.

```
Order, src1$_x$, src1$_y$, dst1$_x$, dst1$_y$,
       src2$_x$, src2$_y$, dst2$_x$, dst2$_y$,
       src3$_x$, src3$_y$, dst3$_x$, dst3$_y$,
       src4$_x$, src4$_y$, dst4$_x$, dst4$_y$,
       ...
```

For example:

```
from collections import namedtuple
from wand.color import Color
from wand.image import Image

Point = namedtuple('Point', ['x', 'y', 'i', 'j'])

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    order = 1.5
    alpha = Point(0, 0, 26, 0)
    beta = Point(139, 0, 114, 23)
    gamma = Point(139, 91, 139, 80)
    delta = Point(0, 92, 0, 78)
    args = (
        order,
        alpha.x, alpha.y, alpha.i, alpha.j,
        beta.x, beta.y, beta.i, beta.j,
        gamma.x, gamma.y, gamma.i, gamma.j,
        delta.x, delta.y, delta.i, delta.j,
    )
    img.distort('polynomial', args)
```

3.12.12 Scale Rotate Translate

A more common form of distortion, the method `'scale_rotate_translate'` can be controlled by the total number of arguments.

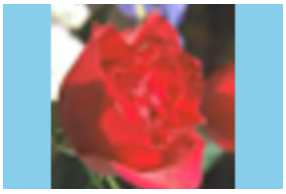
The total arguments dictate the following order.

Total Arguments	Argument Order
1	Angle
2	Scale, Angle
3	X, Y, Angle
4	X, Y, Scale, Angle
5	X, Y, ScaleX, ScaleY, Angle
6	X, Y, Scale, Angle, NewX, NewY
7	X, Y, ScaleX, ScaleY, Angle, NewX, NewY

For example…

A single argument would be treated as an angle:

```
from wand.color import Color
from wand.image import Image

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    angle = 90.0
    img.distort('scale_rotate_translate', (angle,))
```



Two arguments would be treated as a scale & angle:

```
with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    angle = 90.0
    scale = 0.5
    img.distort('scale_rotate_translate', (scale, angle,))
```

And three arguments would describe the origin of rotation:

```
with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    x = 80
    y = 60
    angle = 90.0
    img.distort('scale_rotate_translate', (x, y, angle,))
```

... and so forth.

3.12.13 Shepards

Shepard's distortion moves (or smudges) a given source point to a destination coordinate. The arguments are:

```
src1$_x$, src1$_y$, dst1$_x$, dst1$_y$,
src2$_x$, src2$_y$, dst2$_x$, dst2$_y$,
...
```

The size, or inverse weighted distance, of the source point can be controlled by defining `shepards:power` artifact.

For example:

```
from collections import namedtuple
from wand.color import Color
from wand.image import Image

Point = namedtuple('Point', ['x', 'y', 'i', 'j'])

with Image(filename='rose:') as img:
    img.resize(140, 92)
    img.background_color = Color('skyblue')
    img.virtual_pixel = 'background'
    img.artifacts['distort:viewport'] = "160x112-10+10"
    img.artifacts['shepards:power'] = "4.0"
    alpha = Point(0, 0, 30, 15)
    beta = Point(70, 46, 60, 70)
    args = (
        *alpha,
```

(continues on next page)

(continued from previous page)

```
        *beta
    )
    img.distort('shepards', args)
```

3.13 Drawing

New in version 0.3.0.

The *wand.drawing* module provides some basic drawing functions. *wand.drawing.Drawing* object buffers instructions for drawing shapes into images, and then it can draw these shapes into zero or more images.

It's also callable and takes an *Image* object:

```
from wand.drawing import Drawing
from wand.image import Image

with Drawing() as draw:
    # does something with ``draw`` object,
    # and then...
    with Image(filename='wandtests/assets/beach.jpg') as image:
        draw(image)
```

3.13.1 Arc

New in version 0.4.0.

Arcs can be drawn by using *arc()* method. You'll need to define three pairs of (x, y) coordinates. First & second pair of coordinates will be the minimum bounding rectangle, and the last pair define the starting & ending degree.

An example:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color

with Drawing() as draw:
    draw.stroke_color = Color('black')
    draw.stroke_width = 2
    draw.fill_color = Color('white')
    draw.arc(( 25, 25),  # Stating point
             ( 75, 75),  # Ending point
             (135,-45))  # From bottom left around to top right
    with Image(width=100,
```

(continues on next page)

(continued from previous page)

```
                height=100,
                background=Color('lightblue')) as img:
        draw.draw(img)
        img.save(filename='draw-arc.gif')
```

3.13.2 Bezier

New in version 0.4.0.

You can draw bezier curves using *bezier()* method. This method requires at least four points to determine a bezier curve. Given as a list of (x, y) coordinates. The first & last pair of coordinates are treated as start & end, and the second & third pair of coordinates act as controls.

For example:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color

with Drawing() as draw:
    draw.stroke_color = Color('black')
    draw.stroke_width = 2
    draw.fill_color = Color('white')
    points = [(10,50),  # Start point
              (50,10),  # First control
              (50,90),  # Second control
              (90,50)]  # End point
    draw.bezier(points)
    with Image(width=100,
               height=100,
               background=Color('lightblue')) as image:
        draw(image)
```

Control width & color of curve with the drawing properties:

- *stroke_color*
- *stroke_width*

3.13.3 Circle

New in version 0.4.0.

You can draw circles using *circle()* method. Circles are drawn by defining two pairs of (x, y) coordinates. First coordinate for the center "`origin`" point, and a second pair for the outer `perimeter`. For example, the following code draws a circle in the middle of the `image`:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color
```

(continues on next page)

(continued from previous page)

```
with Drawing() as draw:
    draw.stroke_color = Color('black')
    draw.stroke_width = 2
    draw.fill_color = Color('white')
    draw.circle((50, 50), # Center point
                (25, 25)) # Perimeter point
    with Image(width=100, height=100, background=Color('lightblue')) as image:
        draw(image)
```

3.13.4 Color & Matte

New in version 0.4.0.

You can draw with colors directly on the coordinate system of an image. Define which color to set by setting *fill_color*. The behavior of *color()* is controlled by setting one of *PAINT_METHOD_TYPES* paint methods.

- `'point'` alters a single pixel.
- `'replace'` swaps on color for another. Threshold is influenced by `fuzz`.
- `'floodfill'` fills area of a color influenced by `fuzz`.
- `'filltoborder'` fills area of a color until border defined by *border_color*.
- `'reset'` replaces the whole image to a single color.

Example fill all to green boarder:

```
from wand.drawing import Drawing
from wand.color import Color

with Drawing() as draw:
    draw.border_color = Color('green')
    draw.fill_color = Color('blue')
    draw.color(15, 25, 'filltoborder')
```

The *matte()* method is identical to the *color()* method above, but alters the alpha channel of the color area selected. Colors can be manipulated, but not replaced.

```
with Drawing() as draw:
    draw.fill_color = None  # or Color('none')
    draw.matte(15, 25, 'floodfill')
```

3.13.5 Composite

New in version 0.4.0.

Similar to *composite_channel()*, this *composite()* method will render a given image on top of the drawing subject image following the *COMPOSITE_OPERATORS* options. An compositing image must be given with a destination `top`, `left`, `width`, and `height` values.

```
from wand.image import Image, COMPOSITE_OPERATORS
from wand.drawing import Drawing
from wand.display import display

wizard = Image(filename='wizard:')
rose = Image(filename='rose:')

for o in COMPOSITE_OPERATORS:
  w = wizard.clone()
  r = rose.clone()
  with Drawing() as draw:
    draw.composite(operator=o, left=175, top=250,
                   width=r.width, height=r.height, image=r)
    draw(w)
    display(w)
```

3.13.6 Ellipse

New in version 0.4.0.

Ellipse can be drawn by using the *ellipse()* method. Like drawing circles, the ellipse requires a `origin` point, however, a pair of (x, y) `radius` are used in relationship to the `origin` coordinate. By default a complete "closed" ellipse is drawn. To draw a partial ellipse, provide a pair of starting & ending degrees as the third parameter.

An example of a full ellipse:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color

with Drawing() as draw:
    draw.stroke_color = Color('black')
    draw.stroke_width = 2
    draw.fill_color = Color('white')
    draw.ellipse((50, 50), # Origin (center) point
                 (40, 20)) # 80px wide, and 40px tall
    with Image(width=100, height=100, background=Color('lightblue')) as image:
        draw(image)
```

Same example as above, but with a half-partial ellipse defined by the third parameter:

```
draw.ellipse((50, 50), # Origin (center) point
             (40, 20), # 80px wide, and 40px tall
             (90,-90)) # Draw half of ellipse from bottom to top
```

3.13.7 Lines

You can draw lines using *line()* method. It simply takes two (x, y) coordinates for start and end of a line. For example, the following code draws a diagonal line into the `image`:

```
draw.line((0, 0), image.size)
draw(image)
```

Or you can turn this diagonal line upside down:

```
draw.line((0, image.height), (image.width, 0))
draw(image)
```

The line color is determined by *fill_color* property, and you can change this of course. The following code draws a red diagonal line into the `image`:

```
from wand.color import Color

with Color('red') as color:
    draw.fill_color = color
    draw.line((0, 0), image.size)
    draw(image)
```

3.13.8 Paths

New in version 0.4.0.

Paths can be drawn by using any collection of path functions between *path_start()* and *path_finish()* methods. The available path functions are:

- *path_close()* draws a path from last point to first.
- *path_curve()* draws a cubic bezier curve.
- *path_curve_to_quadratic_bezier()* draws a quadratic bezier curve.
- *path_elliptic_arc()* draws an elliptical arc.
- *path_horizontal_line()* draws a horizontal line.
- *path_line()* draws a line path.
- *path_move()* adjust current point without drawing.
- *path_vertical_line()* draws a vertical line.

Each path method expects a destination point, and will draw from the current point to the new point. The destination point will become the new current point for the next applied path method. Destination points are given in the form of (`x`, `y`) coordinates to the `to` parameter, and can by relative or absolute to the current point by setting the `relative` flag. The *path_curve()* and *path_curve_to_quadratic_bezier()* expect additional `control` points, and can complement previous drawn curves by setting a `smooth` flag. When the `smooth` flag is set to `True` the first control point is assumed to be the reflection of the last defined control point.

For example:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color

with Drawing() as draw:
    draw.stroke_width = 2
    draw.stroke_color = Color('black')
    draw.fill_color = Color('white')
    draw.path_start()
    # Start middle-left
    draw.path_move(to=(10, 50))
    # Curve accross top-left to center
    draw.path_curve(to=(40, 0),
                    controls=[(10, -40), (30,-40)],
                    relative=True)
    # Continue curve accross bottom-right
    draw.path_curve(to=(40, 0),
                    controls=(30, 40),
                    smooth=True,
                    relative=True)
    # Line to top-right
    draw.path_vertical_line(10)
    # Diagonal line to bottom-left
    draw.path_line(to=(10, 90))
    # Close first & last points
    draw.path_close()
    draw.path_finish()
    with Image(width=100, height=100, background=Color('lightblue')) as image:
        draw(image)
```

3.13.9 Point

New in version 0.4.0.

You can draw points by using *point()* method. It simply takes two `x`, `y` arguments for the point coordinate.

The following example will draw points following a math function across a given `image`:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color
import math

with Drawing() as draw:
    for x in xrange(0, 100):
        y = math.tan(x) * 4
        draw.point(x, y + 50)
    with Image(width=100, height=100, background=Color('lightblue')) as image:
        draw(image)
```

Color of the point can be defined by setting the following property

- *fill_color*

3.13.10 Polygon

New in version 0.4.0.

Complex shapes can be created with the *polygon()* method. You can draw a polygon by given this method a list of points. Stroke line will automatically close between first & last point.

For example, the following code will draw a triangle into the `image`:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color

with Drawing() as draw:
    draw.stroke_width = 2
    draw.stroke_color = Color('black')
    draw.fill_color = Color('white')
    points = [(25, 25), (75, 50), (25, 75)]
    draw.polygon(points)
    with Image(width=100, height=100, background=Color('lightblue')) as image:
        draw(image)
```

Control the fill & stroke with the following properties:

- *stroke_color*
- *stroke_dash_array*
- *stroke_dash_offset*
- *stroke_line_cap*
- *stroke_line_join*
- *stroke_miter_limit*
- *stroke_opacity*
- *stroke_width*
- *fill_color*
- *fill_opacity*
- *fill_rule*

3.13.11 Polyline

New in version 0.4.0.

Identical to *polygon()*, except *polyline()* will not close the stroke line between the first & last point.

For example, the following code will draw a two line path on the `image`:

```
from wand.image import Image
from wand.drawing import Drawing
from wand.color import Color

with Drawing() as draw:
    draw.stroke_width = 2
    draw.stroke_color = Color('black')
    draw.fill_color = Color('white')
    points = [(25, 25), (75, 50), (25, 75)]
    draw.polyline(points)
    with Image(width=100, height=100, background=Color('lightblue')) as image:
        draw(image)
```

Control the fill & stroke with the following properties:

- *stroke_color*
- *stroke_dash_array*
- *stroke_dash_offset*
- *stroke_line_cap*
- *stroke_line_join*
- *stroke_miter_limit*
- *stroke_opacity*
- *stroke_width*
- *fill_color*
- *fill_opacity*
- *fill_rule*

3.13.12 Push & Pop

New in version 0.4.0.

When working with complex vector graphics, you can use ImageMagick's internal graphic-context stack to manage different styles & operations. The methods *push()*, *push_clip_path()*, *push_defs()*, and *push_pattern()* are used to mark the beginning of a sub-routine. The clip path & pattern methods take a name based identifier argument, and can be referenced at a latter point with *clip_path*, or *set_fill_pattern_url()* / *set_stroke_pattern_url()* respectively. With stack management, *pop()* is used to mark the end of a sub-routine, and return the graphical context to its pervious state before *push()* was invoked. Methods *pop_clip_path()*, *pop_defs()*, and *pop_pattern()* exist to match there pop counterparts.

```
from wand.color import Color
from wand.image import Image
from wand.drawing import Drawing
from wand.compat import nested
from math import cos, pi, sin

with nested(Color('lightblue'),
            Color('transparent'),
            Drawing()) as (bg, fg, draw):
    draw.stroke_width = 3
    draw.fill_color = fg
    for degree in range(0, 360, 15):
        draw.push()  # Grow stack
        draw.stroke_color = Color('hsl({0}%, 100%, 50%)'.format(degree * 100 / 360))
        t = degree / 180.0 * pi
        x = 35 * cos(t) + 50
        y = 35 * sin(t) + 50
        draw.line((50, 50), (x, y))
        draw.pop()  # Restore stack
    with Image(width=100, height=100, background=Color('lightblue')) as img:
        draw(img)
```

3.13.13 Rectangles

New in version 0.3.6.

Changed in version 0.4.0.

If you want to draw rectangles use *rectangle()* method. It takes `left`/`top` coordinate, and `right`/`bottom` coordinate, or `width` and `height`. For example, the following code draws a square on the `image`:

```
draw.rectangle(left=10, top=10, right=40, bottom=40)
draw(image)
```

Or using `width` and `height` instead of `right` and `bottom`:

```
draw.rectangle(left=10, top=10, width=30, height=30)
draw(image)
```

Support for rounded corners was added in version 0.4.0. The `radius` argument sets corner rounding.

```
draw.rectangle(left=10, top=10, width=30, height=30, radius=5)
draw(image)
```

Both horizontal & vertical can be set independently with `xradius` & `yradius` respectively.

```
draw.rectangle(left=10, top=10, width=30, height=30, xradius=5, yradius=3)
draw(image)
```

Note that the stoke and the fill are determined by the following properties:

- *stroke_color*

- *stroke_dash_array*
- *stroke_dash_offset*
- *stroke_line_cap*
- *stroke_line_join*
- *stroke_miter_limit*
- *stroke_opacity*
- *stroke_width*
- *fill_color*
- *fill_opacity*
- *fill_rule*

3.13.14 Texts

Drawing object can write texts as well using its *text()* method. It takes `x` and `y` coordinates to be drawn and a string to write:

```
draw.font = 'wandtests/assets/League_Gothic.otf'
draw.font_size = 40
draw.text(image.width / 2, image.height / 2, 'Hello, world!')
draw(image)
```

As the above code shows you can adjust several settings before writing texts:

- *font*
- *font_family*
- *font_resolution*
- *font_size*
- *font_stretch*
- *font_style*
- *font_weight*
- *gravity*
- *text_alignment*
- *text_antialias*
- *text_decoration*
- *text_direction*
- *text_interline_spacing*
- *text_interword_spacing*
- *text_kerning*
- *text_under_color*

3.13.15 Word Wrapping

The *Drawing* class, by nature, doesn't implement any form of word-wrapping, and users of the wand library would be responsible for implementing this behavior unique to their business requirements.

ImageMagick's `caption:` coder does offer a word-wrapping solution with *Image.caption()* method, but Python's `textwrap` is a little more sophisticated.

```
from textwrap import wrap
from wand.color import Color
from wand.drawing import Drawing
from wand.image import Image


def draw_roi(contxt, roi_width, roi_height):
    """Let's draw a blue box so we can identify what
    our region of intrest is."""
    ctx.push()
    ctx.stroke_color = Color('BLUE')
    ctx.fill_color = Color('TRANSPARENT')
    ctx.rectangle(left=75, top=255, width=roi_width, height=roi_height)
    ctx.pop()


def word_wrap(image, ctx, text, roi_width, roi_height):
    """Break long text to multiple lines, and reduce point size
    until all text fits within a bounding box."""
    mutable_message = text
    iteration_attempts = 100

    def eval_metrics(txt):
        """Quick helper function to calculate width/height of text."""
        metrics = ctx.get_font_metrics(image, txt, True)
        return (metrics.text_width, metrics.text_height)

    while ctx.font_size > 0 and iteration_attempts:
        iteration_attempts -= 1
        width, height = eval_metrics(mutable_message)
        if height > roi_height:
            ctx.font_size -= 0.75  # Reduce pointsize
            mutable_message = text  # Restore original text
        elif width > roi_width:
            columns = len(mutable_message)
            while columns > 0:
                columns -= 1
                mutable_message = '\n'.join(wrap(mutable_message, columns))
                wrapped_width, _ = eval_metrics(mutable_message)
                if wrapped_width <= roi_width:
                    break
            if columns < 1:
                ctx.font_size -= 0.75  # Reduce pointsize
                mutable_message = text  # Restore original text
        else:
            break
```

(continues on next page)

(continued from previous page)

```
    if iteration_attempts < 1:
        raise RuntimeError("Unable to calculate word_wrap for " + text)
    return mutable_message


message = """This is some really long sentence with the
 word "Mississippi" in it."""

ROI_SIDE = 175

with Image(filename='logo:') as img:
    with Drawing() as ctx:
        draw_roi(ctx, ROI_SIDE, ROI_SIDE)
        # Set the font style
        ctx.fill_color = Color('RED')
        ctx.font_family = 'Times New Roman'
        ctx.font_size = 32
        mutable_message = word_wrap(img,
                                    ctx,
                                    message,
                                    ROI_SIDE,
                                    ROI_SIDE)
        ctx.text(75, 275, mutable_message)
        ctx.draw(img)
        img.save(filename='draw-word-wrap.png')
```

3.14 Reading EXIF

New in version 0.3.0.

Image.metadata contains metadata of the image including EXIF. These are prefixed by `'exif:'` e.g. `'exif:ExifVersion'`, `'exif:Flash'`.

Here's a straightforward example to access EXIF of an image:

```
exif = {}
with Image(filename='wandtests/assets/beach.jpg') as image:
    exif.update((k[5:], v) for k, v in image.metadata.items()
                           if k.startswith('exif:'))
```

Note: You can't write into *Image.metadata*.

3.14.1 Image Profiles

Although wand provides a way to quickly access profile attributes through *Image.metadata*, ImageMagick is not a tag editor. Users are expected to export the profile payload, modify as needed, and import the payload back into the source image. Payload are byte-arrays, and should be treated as binary blobs.

Image profiles can be imported, extracted, and deleted with *Image.profiles* dictionary:

```
with Image(filename='wandtests/assets/beach.jpg') as image:
    # Extract EXIF payload
    if 'EXIF' in image.profiles:
        exif_binary = image.profiles['EXIF']
    # Import/replace ICC payload
    with open('color_profile.icc', 'rb') as icc:
        image.profiles['ICC'] = icc.read()
    # Remove XMP payload
    del image.profiles['XMP']
```

Note: Each write operation on any profile type requires the raster image-data to be re-encoded. On lossy formats, such encoding operations can be considered a generation loss.

3.15 Layers

3.15.1 Coalesce Layers

New in version 0.5.0.

When *reading* animations that have already been optimized, be sure to call *coalesce()* before performing any additional operations. This is especially important as the `MagickWand` internal iterator state may be pointing to the last frame read into the image stack, and with optimized images, this is usually a sub-image only holding a frame delta.

```
>>>  with Image(filename='layers-optmized.gif') as img:
...      img.coalesce()
...      # ... do work ...
```

3.15.2 Optimizing Layers

New in version 0.5.0.

A few optimization techniques exist when working with animated graphics. For example, a GIF image would have a rather large file size if every frame requires the full image to be redrawn. Let's take a look at the effects of *optimize_layers()*, and *optimize_transparency()*.

To start, we can quickly create an animated gif.

```
from wand.color import Color
from wand.image import Image

with Image(width=100, height=100, pseudo='pattern:crosshatch') as canvas:
    canvas.negate()
```

(continues on next page)

(continued from previous page)

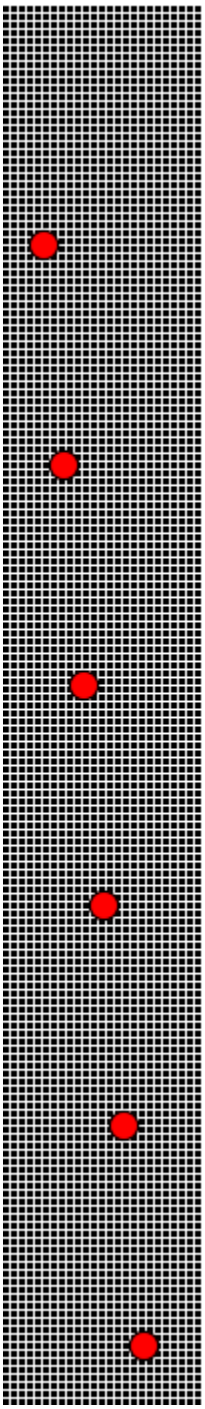
```
    for offset in range(20, 80, 10):
        with canvas.clone() as frame:
            with Drawing() as ctx:
                ctx.fill_color = Color('red')
                ctx.stroke_color = Color('black')
                ctx.circle((offset, offset), (offset+5, offset+5))
                ctx.draw(frame)
            canvas.sequence.append(frame)
    canvas.save(filename='layers.gif')
```

Another quick helper method to allow us to view/debug each frame.

```
def debug_layers(image, output):
    print('Debugging to file', output)
    with Image(image) as img:
        img.background_color = Color('lime')
        for index, frame in enumerate(img.sequence):
            print('Frame {0} size : {1} page: {2}'.format(index,
                                                          frame.size,
                                                          frame.page))
        img.concat(stacked=True)
        img.save(filename=output)
```

We can debug the previously created `layers.gif` by running the following:

```
>>>  with Image(filename='layers.gif') as img:
...      debug_layers(img, 'layers-expanded.png')
Debugging to file layers-expanded.png
Frame 0 size : (100, 100) page: (100, 100, 0, 0)
Frame 1 size : (100, 100) page: (100, 100, 0, 0)
Frame 2 size : (100, 100) page: (100, 100, 0, 0)
Frame 3 size : (100, 100) page: (100, 100, 0, 0)
Frame 4 size : (100, 100) page: (100, 100, 0, 0)
Frame 5 size : (100, 100) page: (100, 100, 0, 0)
Frame 6 size : (100, 100) page: (100, 100, 0, 0)
```



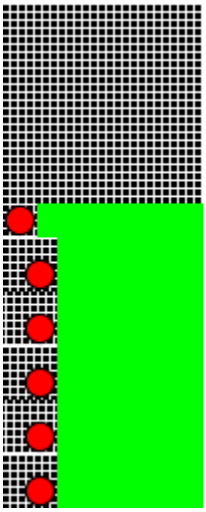
The moving circle is the only thing that changes between each frame, so we can optimize by having each frame only contain the delta.

```
>>>  with Image(filename='layers.gif') as img:
...      img.optimize_layers()
...      debug_layers(img, 'layers-optmized-layers.png')
Debugging to file layers-optmized-layers.png
Frame 0 size : (100, 100) page: (100, 100, 0, 0)
Frame 1 size : (17, 17) page: (100, 100, 12, 12)
```

(continues on next page)

(continued from previous page)

```
Frame 2 size : (26, 27) page: (100, 100, 12, 12)
Frame 3 size : (26, 27) page: (100, 100, 23, 22)
Frame 4 size : (26, 27) page: (100, 100, 32, 32)
Frame 5 size : (26, 27) page: (100, 100, 43, 42)
Frame 6 size : (26, 27) page: (100, 100, 52, 52)
```



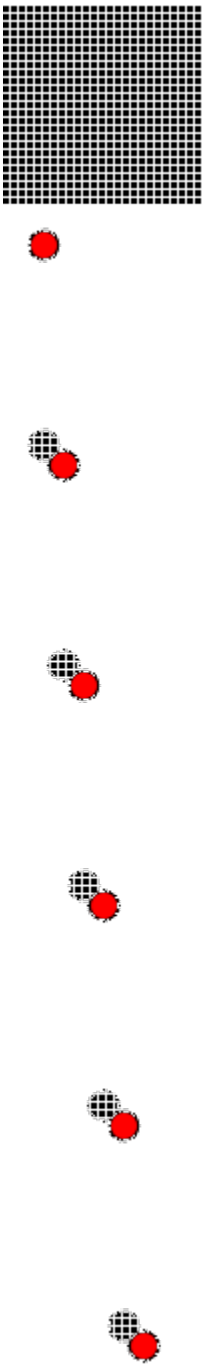
Notice each frame after the first has a reduce size & page x/y offset. Contacting each frame shows only the minimum bounding region covering the pixel changes across each previous frame. *Note: the lime-green background is only there for a visual cue one the website, and has not special meaning outside of "no-data here."*

3.15.3 Optimizing Transparency

New in version 0.5.0.

Following the above examples, we can also optimize by forcing pixels transparent if they are unchanged since the previous frame.

```
>>> with Image(filename='layers.gif') as img:
...     img.optimize_transparency()
...     debug_layers(img, 'layers-optmized-transparent.png')
Debugging to file layers-optmized-transparent.png
Frame 0 size : (100, 100) page: (100, 100, 0, 0)
Frame 1 size : (100, 100) page: (100, 100, 0, 0)
Frame 2 size : (100, 100) page: (100, 100, 0, 0)
Frame 3 size : (100, 100) page: (100, 100, 0, 0)
Frame 4 size : (100, 100) page: (100, 100, 0, 0)
Frame 5 size : (100, 100) page: (100, 100, 0, 0)
Frame 6 size : (100, 100) page: (100, 100, 0, 0)
```



Notice both the size of each frame, and the page offset are unchanged. This technique only really saves if the subject already contains transparency color channels, and so most modern gif animations would not benefit from this method.

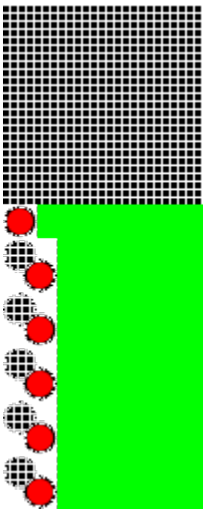
Naturally, applying both layer & transparency optimization will demonstrate both effects.

```
>>> with Image(filename='layers.gif') as img:
...     img.optimize_layers()
...     img.optimize_transparency()
...     debug_layers(img, 'layers-optmized-layers-transparent.png')
```

(continues on next page)

(continued from previous page)

```
Debugging to file layers-optmized-layers-transparent.png
Frame 0 size : (100, 100) page: (100, 100, 0, 0)
Frame 1 size : (17, 17) page: (100, 100, 12, 12)
Frame 2 size : (26, 27) page: (100, 100, 12, 12)
Frame 3 size : (26, 27) page: (100, 100, 23, 22)
Frame 4 size : (26, 27) page: (100, 100, 32, 32)
Frame 5 size : (26, 27) page: (100, 100, 43, 42)
Frame 6 size : (26, 27) page: (100, 100, 52, 52)
```



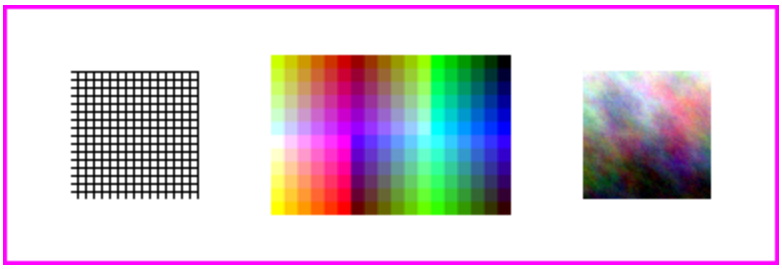
Note: Lime-green background added for visibility cue.

3.16 Montage

New in version 0.6.8.

The *Image.montage()* method iterates over each image loaded on the stack, and generates a new image containing thumbnails for appending together. Similar to a contact sheets used by film photography.

```
from wand.image import Image

with Image() as img:
    for src in ['pattern:crosshatch', 'netscape:', 'plasma:']:
        with Image(width=64, height=64, pseudo=src) as item:
            img.image_add(item)
    img.montage()
    img.border('magenta', 2, 2)
    img.save(filename='montage-default.png')
```



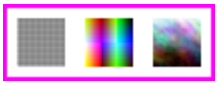
3.16.1 Thumbnail Size

We can control the size of the thumbnail, and the padding between with the `thumbnail=` key-word argument. The value of this argument is a geometry string following the format:

```
{WIDTH}x{HEIGHT}+{X}+{Y}
```

For example:

```
from wand.image import Image

with Image() as img:
    for src in ['pattern:crosshatch', 'netscape:', 'plasma:']:
        with Image(width=64, height=64, pseudo=src) as item:
            img.image_add(item)
    img.montage(thumbnail="24x24!+5+5")
    img.border('magenta', 2, 2)
    img.save(filename='montage-thumbnail.png')
```



Note: Thumbnail geometry modifiers are supported.

flag	Description
!	Ignore aspect.
>	Only shrink larger images.
<	Only enlarge small images.
^	Fill area.
%	Use percent value.
@	Use pixel count.

3.16.2 Tile Layout

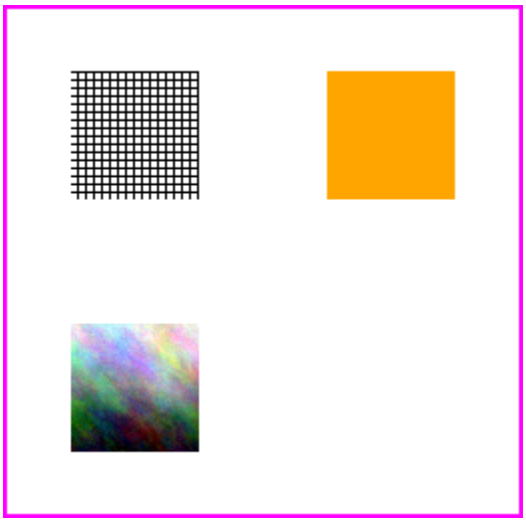
New in version 0.6.8.

The number of images per row & column can be controlled by setting the `tile=` key-word argument. This argument's value follows the geometry format:

```
{COLUMNS}x{ROWS}
```

For example:

```
from wand.image import Image

with Image() as img:
    for src in ['pattern:crosshatch', 'canvas:orange', 'plasma:']:
        with Image(width=64, height=64, pseudo=src) as item:
            img.image_add(item)
    img.montage(tile='2x2')
    img.border('magenta', 2, 2)
    img.save(filename='montage-tile.png')
```



3.16.3 Concatenation Mode

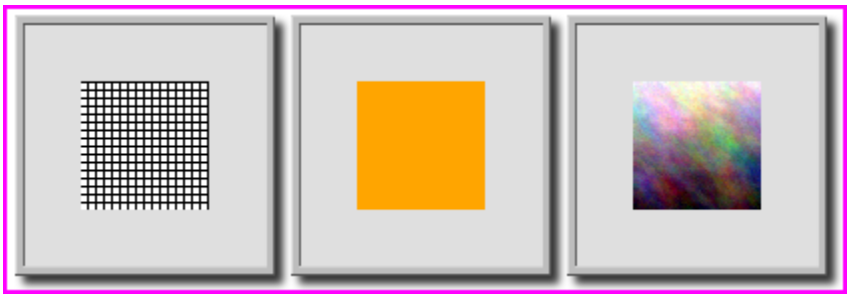
Use the `mode="concatenate"` keyword argument to eliminate extra whitespace between image thumbnails. For example:

```
from wand.image import Image

with Image() as img:
    for src in ['pattern:crosshatch', 'canvas:orange', 'plasma:']:
        with Image(width=64, height=64, pseudo=src) as item:
            img.image_add(item)
    img.montage(mode='concatenate')
    img.border('magenta', 2, 2)
    img.save(filename='montage-concatenate.png')
```

3.16.4 Frame Mode

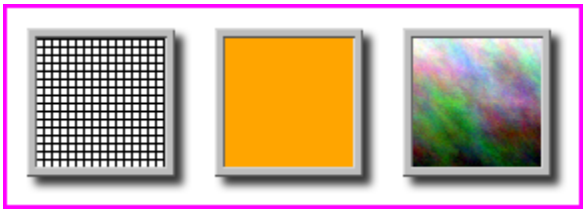
To draw a decorative frame around each thumbnail, use a combination of `mode="frame"` and `frame=geometry` key-words.

```
from wand.image import Image

with Image() as img:
    for src in ['pattern:crosshatch', 'canvas:orange', 'plasma:']:
        with Image(width=64, height=64, pseudo=src) as item:
            img.image_add(item)
    img.montage(mode='frame', frame='5')
    img.border('magenta', 2, 2)
    img.save(filename='montage-frame.png')
```



Define the `thumbnail=` key-word to control the border between the thumbnail, and frame.

```
img.montage(mode='frame', frame='5', thumbnail="64x64+10+10")
```



3.16.5 Labels

The montage method will apply a text label under the thumbnail if the source image contains an ImageMagick label attribute. You can style the typeface of the label's text by passing *Font* instance to the `font=` key-word.

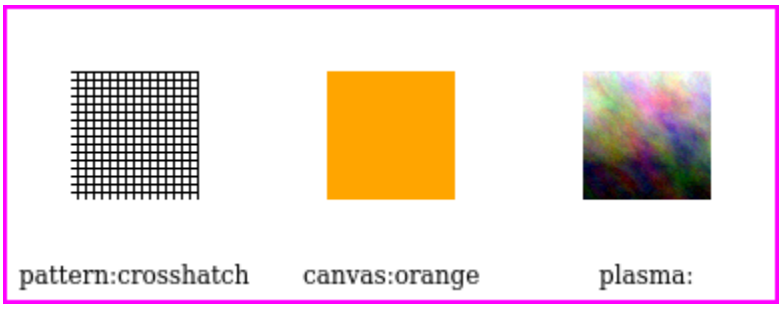
For example:

```
from wand.font import Font
from wand.image import Image

with Image() as img:
```

(continues on next page)

(continued from previous page)

```
    for src in ['pattern:crosshatch', 'canvas:orange', 'plasma:']:
        with Image() as item:
            # NOTE: Set the label before reading the image.
            item.options['label'] = src
            item.pseudo(64, 64, src)
            img.image_add(item)
    style = Font("DejaVuSerif.ttf", 12, 'black')
    img.montage(font=style)
    img.border('magenta', 2, 2)
    img.save(filename='montage-concatenate.png')
```

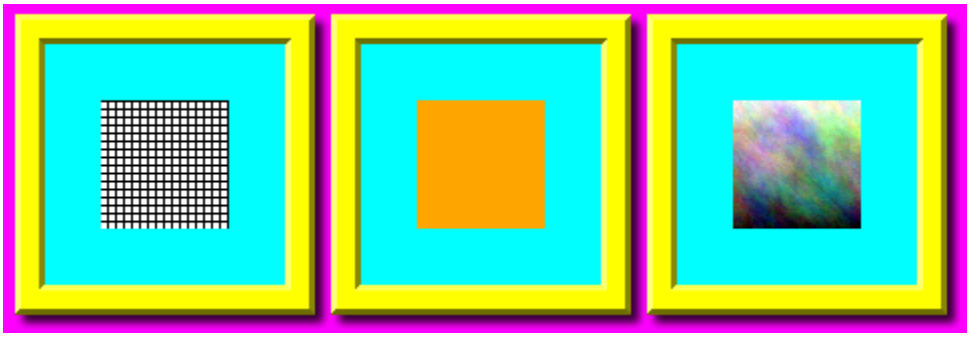


3.16.6 Colors

You can apply colors to each component within the montage image. The overall background can be defined by setting *background_color* attribute on the base image, and *matte_color* & *border_color* on each thumbnail's source image.

For example:

```
from wand.image import Image

with Image() as img:
    for src in ['pattern:crosshatch', 'canvas:orange', 'plasma:']:
        with Image(width=64, height=64, pseudo=src) as item:
            item.border_color = 'cyan'   # Inner Frame
            item.matte_color = 'yellow'  # Outer Frame
            img.image_add(item)
    img.background_color = 'magenta'     # Canvas background
    img.montage(font=style)
    img.save(filename='montage-color.png')
```



3.17 Morphology

Morphology modifies an image by evaluating the pixel values surrounding each pixel. The basic Wand method signature is:

```
img.morphology(method, kernel, iterations)
```

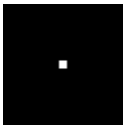
Where `method` is the operation to apply, and is defined by *MORPHOLOGY_METHODS*. The `kernel` can include predefined built-in shapes, or user-defined shapes.

3.17.1 Shapes

Shapes, also known as "kernels", are small matrices that control morphology method operations. The kernels define the size, and targeted pixels to modify.

To demonstrate a kernel's shape; let's generate a simple black canvas around a white pixel.

```
from wand.image import Image

with Image(width=1, height=1, pseudo='xc:white') as img:
    img.border('black', 6, 6, compose='copy')
    img.save(filename='morph-dot.png')
```



Built-In Kernels

ImageMagick contains about three dozen pre-built kernels that cover most common morphology uses, as well as a few specific ones leveraged for internal operations. To use built-in kernels, the following string format is required.

```
label[:arg1,arg2,arg3,..]
```

Where *label* is a string defined in *KERNEL_INFO_TYPES*. Each label can have additional optional arguments, which are defined by a comma separated list of doubles. A colon `':'` symbol should separate the label & argument list. For example:

```
disk:2.5,3,5
```

Below is a small list of examples for the most common kernel shapes.

Cross

```
with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel='cross:3')
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-cross3.png')
```

Diamond

```
with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel='diamond:3')
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-diamond3.png')
```

Disk

```
with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel='disk:5')
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-disk5.png')
```

Octagon

```
with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel='octagon:5')
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-octagon5.png')
```

Plus

```
with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel='plus:3')
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-plus3.png')
```

Ring

```
with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel='ring:5,4')
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-ring5.png')
```

Square

```
with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel='square:3')
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-square3.png')
```

Custom Kernels

Users can define their own kernel shape by building a string that follows the format:

```
geometry:pix1,pix2,pix3,...
```

Where geometry is defined as *WIDTHxHEIGHT* of the kernel, followed by a colon, and then a comma separated list of float values. For example:

```
custom_kernel = """
5x5:
    -,-,1,-,-
    -,1,1,1,-
    -,-,-,-,-
```

(continues on next page)

(continued from previous page)

```
    -,1,-,1,-
    1,1,1,1,1
"""

with Image(filename='morph-dot.png') as img:
    img.morphology(method='dilate', kernel=custom_kernel)
    img.sample(width=60, height=60)
    img.save(filename='morph-kernel-custom.png')
```

By default, the kernel's "origin" is calculated to be at the center of the kernel. Users can set the kernel origin by defining $\pm X \pm Y$ as part of the geometry. For example:

```
top_left_origin = """
3x3+0+0:
    1,1,-
    1,0,0
    -,0,-
"""

bottom_right_origin = """
3x3+2+2:
    1,1,-
    1,0,0
    -,0,-
"""
```

3.17.2 Methods

Morphology methods are broken into three general groups. Basic methods (such as *Erode*, *Dilate*, *Open*, & *Close*) are used to increase or reduce foreground shapes. Difference methods (such as *Edge In*, *Edge Out*, *Top Hat* & *Bottom Hat*) draw pixels around foreground edges. Pattern matching methods (such as *Hit and Miss*, *Thinning* & *Thicken*) add pixels when a kernel is matched.

Morphology is intended for images with a black background, and a white foreground. To demonstrate morphology methods, let's create a basic binary image. We can quickly generate a *PBM* image from bytes-string literal.

```
pbm = b"""P1
10 10
1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 0 0 0 0 1
1 0 0 0 1 0 0 1 1 1
1 1 0 1 1 0 0 0 0 1
1 1 1 1 1 0 0 1 1 1
1 1 1 1 0 0 0 0 0 1
1 0 0 0 0 0 0 1 1 1
1 0 1 0 1 0 1 1 1 1
1 0 1 0 1 0 1 1 0 1
```

(continues on next page)

(continued from previous page)

```
1 1 1 1 1 1 1 1 1 1
"""

with Image(blob=pbm, format="PBM") as img:
    img.sample(100, 100)
    img.save(filename="morph-src.png")
```

The morphology examples below will all use `'morph-src.png'` source image.

Erode

Erode reduces matching white pixels, and expands black spaces.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='erode', kernel='octagon')
    img.save(filename='morph-erode.png')
```

Dilate

Dilate increases matching white pixels, and reduces black spaces.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='dilate', kernel='octagon')
    img.save(filename='morph-dilate.png')
```

Open

Open rounds the white edges, but preserves "holes", or black corners.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='open', kernel='octagon')
    img.save(filename='morph-open.png')
```

Notices the black "inner" corners remain sharp, but the white "outer" corners are rounded.

Close

Close rounds the black edges, and removes any "holes".

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='close', kernel='octagon')
    img.save(filename='morph-close.png')
```

Notices the white "outer" corners remain sharp, but the black "inner" corners are rounded.

Smooth

Smooth applies both *Open* & *Close* methods. This will remove small objects & holes, and smooth both white & black corners.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='smooth', kernel='octagon')
    img.save(filename='morph-smooth.png')
```

Edge In

Edge In method performs a *Erode*, but only keeps the targeted pixel next to a shape. This means the edge is drawn just inside the white of a object.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='edgein', kernel='octagon')
    img.save(filename='morph-edgein.png')
```

Edge Out

Edge Out performs similar to *Edge In*, but uses the results of *Dilate* to draw a edge border just outside of an object.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='edgeout', kernel='octagon')
    img.save(filename='morph-edgeout.png')
```

Edge

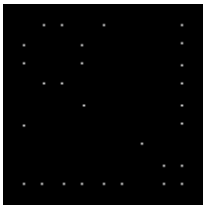
The Edge method performs both *Erode* & *Dilate* methods, but only keeps differences between them as the resulting image. The result is border drawn on the edge of the objects within the image.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='edge', kernel='octagon')
    img.save(filename='morph-edge.png')
```

Top Hat

The Top Hat method performs the *Open* morphology method, but only returns the pixels matched by the kernel.

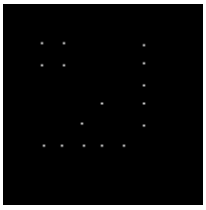
```
with Image(filename='morph-src.png') as img:
    img.morphology(method='tophat', kernel='octagon')
    img.save(filename='morph-tophat.png')
```



Bottom Hat

The Bottom Hat method performs the *Close* morphology method, but only returns the pixels matched by the kernel.

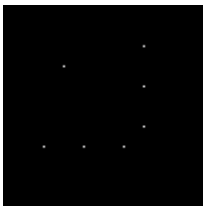
```
with Image(filename='morph-src.png') as img:
    img.morphology(method='bottom_hat', kernel='octagon')
    img.save(filename='morph-bottom_hat.png')
```



Hit and Miss

The hit-and-miss (a.k.a. HMT) method will remove all pixels from the image, unless a kernel pattern is matched; in which case, the pixel under the matched kernel will be set to white.

```
with Image(filename='morph-src.png') as img:
    corners = """
    3x3:
        1,1,-
        1,0,0
        -,0,-
    """
    img.morphology(method='hit_and_miss', kernel=corners)
    img.save(filename='morph-hit_and_miss.png')
```



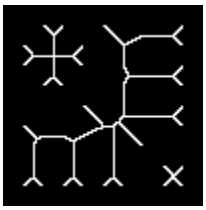
Thinning

The thinning method removes a pixel when the kernel matches neighboring pixels. When using custom kernels, you can control which pixel should be targeted by setting the X/Y offset of the kernel's geometry.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='thinning',
                   kernel='3x1-0-0:1,1,0',
                   iterations=3)
    img.save(filename='morph-thinning.png')
```

There's also a special `'skeleton'` built-in kernel, paired with *-1* iterations to continue to reduce all pixels down to a minimum line.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='thinning',
                   kernel='skeleton',
                   iterations=-1)
    img.save(filename='morph-thinning-skeleton.png')
```



Thicken

The thicken method adds a pixel whenever a kernel matches neighboring pixels. You can control the targeted pixel by defining the offset geometry on custom kernels.

```
with Image(filename='morph-src.png') as img:
    K = """
    3x3+0+0:
        0,-,-
        -,0,-
        -,-,1
    """
    img.morphology(method='thicken',
                   kernel=K,
                   iterations=4)
    img.save(filename='morph-thicken.png')
```

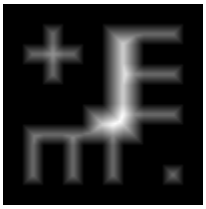
Distance

Distance method is a unique, and very special morphology. Given a binary black & white image, each white pixel will be replace with a color value corresponding to the distance to the nearest edge.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='distance',
                   kernel='euclidean',
                   iterations=4)
    img.save(filename='morph-distance-raw.png')
```

The resulting image is not that special. The pixel values are so low that they appear black. However, if we use *auto_level()* method, we should be able to shift the values across the full grayscale.

```
with Image(filename='morph-src.png') as img:
    img.morphology(method='distance',
                   kernel='euclidean',
                   iterations=4)
    img.auto_level()
    img.save(filename='morph-distance-auto.png')
```



Other kernels used for distance morphology are `'chebyshev'`, `'manhattan'`, `'octagonal'`, and `'euclidean'`. The basic kernel string format is:

```
distance_kernel[:radius[,scale]]
```

For example:

```
manhattan:5,400
```

3.18 Sequence

Note: The image `sequence-animation.gif` used in this docs has been released into the public domain by its author, C6541 at Wikipedia project. This applies worldwide. (Source)

New in version 0.3.0.

Some images may actually consist of two or more images. For example, animated *image/gif* images consist of multiple frames. Some *image/ico* images have different sizes of icons.

For example, the above image `sequence-animation.gif` consists of the following frames (actually it has 60 frames, but we sample only few frames to show here):

3.18.1 sequence is a Sequence

If we *open* this image, *Image* object has *sequence*. It's a list-like object that maintain its all frames.

For example, `len()` for this returns the number of frames:

```
>>> from wand.image import Image
>>> with Image(filename='sequence-animation.gif') as image:
...     len(image.sequence)
...
60
```

You can get an item by index from *sequence*:

```
>>> with Image(filename='sequence-animation.gif') as image:
...     image.sequence[0]
...
<wand.sequence.SingleImage: ed84c1b (256x256)>
```

Or slice it:

```
>>> with Image(filename='sequence-animation.gif') as image:
...     image.sequence[5:10]
...
[<wand.sequence.SingleImage: 0f49491 (256x256)>,
 <wand.sequence.SingleImage: 8eba0a5 (256x256)>,
 <wand.sequence.SingleImage: 98c10fa (256x256)>,
 <wand.sequence.SingleImage: b893194 (256x256)>,
 <wand.sequence.SingleImage: 181ce21 (256x256)>]
```

3.18.2 Image versus SingleImage

Note that each item of *sequence* is a *SingleImage* instance, not *Image*.

Image is a container that directly represents *image files* like `sequence-animation.gif`, and *SingleImage* is a single image that represents *frames* in animations or *sizes* in *image/ico* files.

They both inherit *BaseImage*, the common abstract class. They share the most of available operations and properties like *resize()* and *size*, but some are not. For example, *save()* and *mimetype* are only provided by *Image*. *delay* and *index* are only available for *SingleImage*.

In most cases, images don't have multiple images, so it's okay if you think that *Image* and *SingleImage* are the same, but be careful when you deal with animated *image/gif* files or *image/ico* files that contain multiple icons.

3.18.3 Manipulating SingleImage

When working with *sequence*, it's important to remember that each instance of *SingleImage* holds a *copy* of image data from the stack. Altering the copied data will not automatically sync back to the original image-stack.

```
>>> with Image(filename='animation.gif') as image:
...     # Changes on SingleImage are invisible to `image` container.
...     image.sequence[2].negate()
...     image.save(filename='output.gif')  # Changes ignored.
```

If you intended to alter a *SingleImage*, and have changes synchronized back to the parent image-stack, use an additional with-statement context manager.

```
>>> with Image(filename='animation.gif') as image:
...     # Changes on SingleImage are sync-ed after context manager closes.
...     with image.sequence[2] as frame:
...         frame.negate()
...     image.save(filename='output.gif')  # Changes applied.
```

3.18.4 Working directly with Image-Stack Iterators

A faster way to work with images in a sequence is to use the internal stack iterator. This does not create copies, or generate *Sequence* / *SingleImage* instances.

Warning: Users should **NOT** mix *Image.sequence* code with direct iterator methods.

When reading a image file, the internal iterator is pointing to the last frame read. To iterate over all frames, use *Image.iterator_reset()* and *Image.iterator_next()* methods.

```
>>> with Image(filename='link_to_the_past.gif') as img:
...     img.iterator_reset()
...     print("First frame", img.size)
...     while img.iterator_next():
...         print("Additional frame", img.size)
First frame (300, 289)
Additional frame (172, 128)
Additional frame (172, 145)
Additional frame (123, 112)
```

(continues on next page)

(continued from previous page)

```
Additional frame (144, 182)
Additional frame (107, 117)
Additional frame (171, 128)
Additional frame (123, 107)
```

You can also iterate backwards with *Image.iterator_last()* and *Image.iterator_previous()* methods.

```
>>> with Image(filename='link_to_the_past.gif') as img:
...     img.iterator_last()
...     print("End frame", img.size)
...     while img.iterator_previous():
...         print("Previous frame", img.size)
End frame (123, 107)
Previous frame (171, 128)
Previous frame (107, 117)
Previous frame (144, 182)
Previous frame (123, 112)
Previous frame (172, 145)
Previous frame (172, 128)
Previous frame (300, 289)
```

Method *Image.iterator_first()* is like *Image.iterator_reset()*, but allows the next image read to be prepended at the start of the stack.

```
>>> with Image(filename='support/link_net.gif') as img:
...     img.iterator_first()
...     img.pseudo(1, 1, 'xc:gold')
...     img.iterator_set(0)
...     print(img.size)
...     img.iterator_set(1)
...     print(img.size)
(1, 1)
(300, 289)
```

Note: The "image-stack" is general term for a linked list of sub-images in a file. The nomenclature varies between industries & users. You may find documents referencing sub-images as:

- *Frame* for animated formats (GIF)
- *Page* for document based formats (PDF)
- *Layer* for publishing formats (PSD, TIFF)

3.19 Resource management

See also:

wand.resource **— Global resource management** There is the global resource to manage in MagickWand API. This module implements automatic global resource management through reference counting.

Objects Wand provides are resources to be managed. It has to be closed (destroyed) after using like file or database connection. You can deal with it using `with` very easily and explicitly:

```
with Image(filename='') as img:
    # deal with img...
```

Or you can call its *destroy()* (or *close()* if it is an *Image* instance) method manually:

```
try:
    img = Image(filename='')
    # deal with img...
finally:
    img.destroy()
```

Note: It also implements the destructor that invokes *destroy()*, and if your program runs on CPython (which does reference counting instead of ordinary garbage collection) most of resources are automatically deallocated.

However it's just depending on CPython's implementation detail of memory management, so it's not a good idea. If your program runs on PyPy (which implements garbage collector) for example, invocation time of destructors is not determined, so the program would be broken.

3.20 Quantize

3.20.1 Kmeans

New in version 0.6.4.

Reduces the number of colors to a target number. Processing stops if max number of iterations, or tolerance are reached.

```
from wand.image import Image

with Image(filename='hummingbird.jpg') as img:
    img.kmeans(number_colors=32, max_iterations=100, tolerance=0.01)
    img.save(filename='quantize_kmeans.jpg')
```

Original	Quantize Kmeans
	

We can also seed the initial colors used on the first iteration by defining a semicolon delimited list of colors.

```
from wand.image import Image

with Image(filename='hummingbird.jpg') as img:
    img.artifacts['kmeans:seed-colors'] = 'teal;#586f5f;#4d545c;#617284'
    img.kmeans(number_colors=32, max_iterations=100, tolerance=0.01)
    img.save(filename='quantize_kmeans_seed.jpg')
```

Original	Quantize Kmeans (seed)
	

Note: Requires ImageMagick-7.0.10-37

3.20.2 Posterize

New in version 0.5.0.

Reduces number of colors per channel. Dither can be defined by passing `'no'`, `'riemersma'`, or `'floyd_steinberg'` arguments.

```
from wand.image import Image

with Image(filename='hummingbird.jpg') as img:
    img.posterize(levels=16, dither='floyd_steinberg')
    img.save(filename='quantize_posterize.jpg')
```

Original	Quantize Posterize
	

3.20.3 Quantize

New in version 0.4.2.

Analyzes the colors in an image, and replace pixel values from a fixed number of color.

```
from wand.image import Image

with Image(filename='hummingbird.jpg') as img:
    img.quantize(number_colors=8, colorspace_type='srgb',
                 treedepth=1, dither=True, measure_error=False)
    img.save(filename='quantize_quantize.jpg')
```

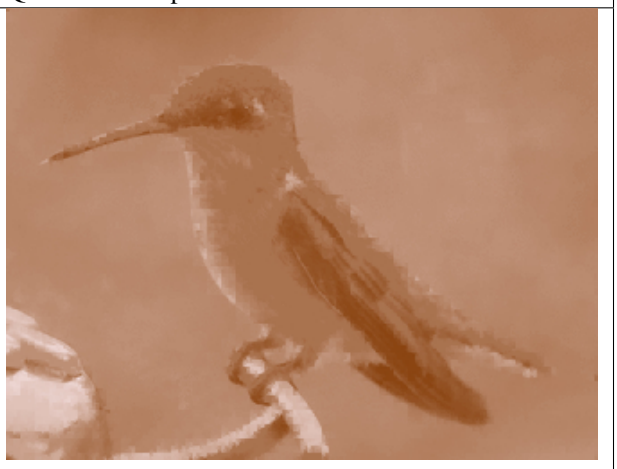
Original	Quantize
	

3.20.4 Remap

New in version 0.5.3.

Remap replaces all pixels with the closest matching pixel found in the *affinity* reference image.

```
from wand.image import Image

with Image(filename='hummingbird.jpg') as img:
    with Image(width=256, height=1,
               pseudo='gradient:SaddleBrown-LavenderBlush') as amap:
        img.remap(affinity=amap, method='riemersma')
    img.save(filename='quantize_remap.jpg')
```

Original	Quantize Remap
	

3.21 Threshold

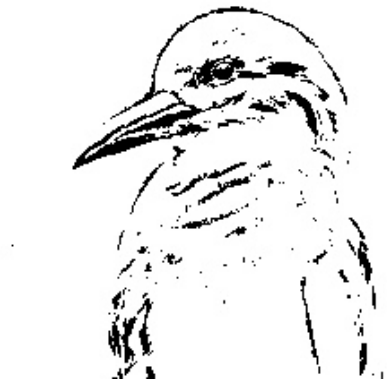
3.21.1 Adaptive Threshold

New in version 0.5.3.

Also known as Local Adaptive Threshold, each pixel value is adjusted by the surrounding pixels. If the current pixel has greater value than the average of the surrounding pixels, then the pixel becomes white, else black.

The size of the surrounding pixels is defined by passing `width` & `height` arguments. Use `offset` argument to apply a +/- value to the current pixel.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.transform_colorspace('gray')
    img.adaptive_threshold(width=16, height=16,
                           offset=-0.08 * img.quantum_range)
    img.save(filename='threshold_adaptive.jpg')
```

Original	Adaptive Threshold
	

Note: Requires ImageMagick-7.

3.21.2 Auto Threshold

New in version 0.5.5.

This method applies threshold automatically. You can define which method to use from *AUTO_THRESHOLD_METHODS*. Defaults to `'kapur'`.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.transform_colorspace('gray')
    with img.clone() as kapur:
        kapur.auto_threshold(method='kapur')
```

(continues on next page)

(continued from previous page)

```
        kapur.save(filename='threshold_auto_kapur.jpg')
    with img.clone() as otsu:
        otsu.auto_threshold(method='otsu')
        otsu.save(filename='threshold_auto_otsu.jpg')
    with img.clone() as triangle:
        triangle.auto_threshold(method='triangle')
        triangle.save(filename='threshold_auto_triangle.jpg')
```

Original	Auto Threshold (`'kapur'`)
	
Auto Threshold (`'otsu'`)	Auto Threshold (`'triangle'`)
	

Note: Requires ImageMagick-7.0.8-41

3.21.3 Black Threshold

New in version 0.5.3.

Force all pixels below a given pixel value to black. This works on a color channel-by-channel basis, and can be used to reduce unwanted colors.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.black_threshold(threshold='#930')
    img.save(filename='threshold_black.jpg')
```

Original	Black Threshold
	

3.21.4 Color Threshold

New in version 0.6.4.

Creates a binary image where all pixels between `start` & `stop` are forced to white, else black.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.color_threshold(start='#333', stop='#cdc')
    img.save(filename='threshold_color.jpg')
```

Original	Color Threshold
	

Note: Requires ImageMagick-7.0.10

3.21.5 Ordered Dither

Applies a pre-defined threshold map to create dithering to an image.

The pre-defined thresholds are the following:

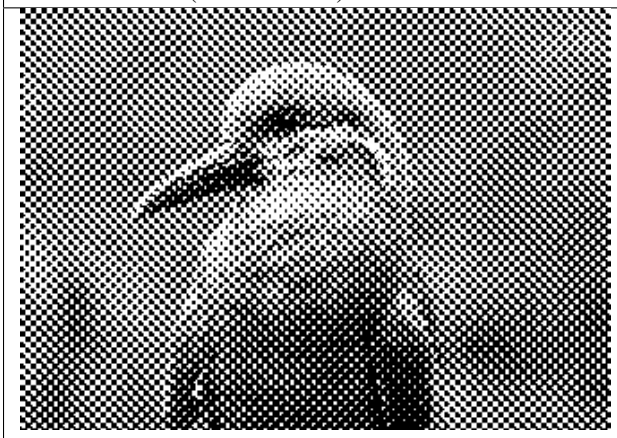
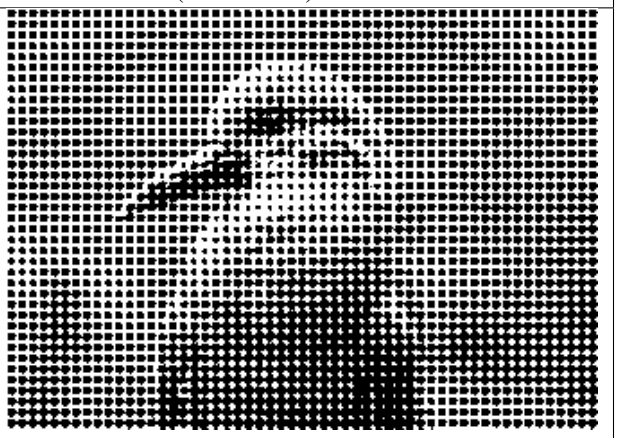
Map	Alias	Description
threshold	1x1	Threshold 1x1 (non-dither)
checks	2x1	Checkerboard 2x1 (dither)
o2x2	2x2	Ordered 2x2 (dispersed)
o3x3	3x3	Ordered 3x3 (dispersed)
o4x4	4x4	Ordered 4x4 (dispersed)
o8x8	8x8	Ordered 8x8 (dispersed)
h4x4a	4x1	Halftone 4x4 (angled)
h6x6a	6x1	Halftone 6x6 (angled)
h8x8a	8x1	Halftone 8x8 (angled)
h4x4o		Halftone 4x4 (orthogonal)
h6x6o		Halftone 6x6 (orthogonal)
h8x8o		Halftone 8x8 (orthogonal)
h16x16o		Halftone 16x16 (orthogonal)
c5x5b	c5x5	Circles 5x5 (black)
c5x5w		Circles 5x5 (white)
c6x6b	c6x6	Circles 6x6 (black)
c6x6w		Circles 6x6 (white)
c7x7b	c7x7	Circles 7x7 (black)
c7x7w		Circles 7x7 (white)

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.transform_colorspace('gray')
```

(continues on next page)

(continued from previous page)

```
    with img.clone() as dispersed:
        dispersed.ordered_dither('o3x3')
        dispersed.save(filename='threshold_ordered_dither_dispersed.jpg')
    with img.clone() as halftone:
        halftone.ordered_dither('h6x6a')
        halftone.save(filename='threshold_ordered_dither_halftone.jpg')
    with img.clone() as circles:
        circles.ordered_dither('c6x6b')
        circles.save(filename='threshold_ordered_dither_circles.jpg')
```

Original	Ordered Dither (Ordered 3x3)
	
Ordered Dither (Halftone 4x4)	Ordered Dither (Circles 6x6)
	

3.21.6 Random Threshold

New in version 0.5.7.

Applies a random threshold between `low` & `high` values.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.transform_colorspace('gray')
    img.random_threshold(low=0.3 * img.quantum_range,
```

(continues on next page)

(continued from previous page)

```
                        high=0.6 * img.quantum_range)
    img.save(filename='threshold_random.jpg')
```

Original	Random Threshold
	

3.21.7 Range Threshold

New in version 0.5.5.

This can either apply a soft, or hard, threshold between two quantum points.

To use a soft threshold, define the low & high range between each white & black point.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.transform_colorspace('gray')
    white_point = 0.9 * img.quantum_range
    black_point = 0.5 * img.quantum_range
    delta = 0.05 * img.quantum_range
    img.range_threshold(low_black=black_point - delta,
                        low_white=white_point - delta,
                        high_white=white_point + delta,
                        high_black=black_point + delta)
    img.save(filename='threshold_range_soft.jpg')
```

Original	Range Threshold (soft)
	

To use a hard threshold, pass the same values as both low & high range.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.transform_colorspace('gray')
    white_point = 0.9 * img.quantum_range
    black_point = 0.5 * img.quantum_range
    img.range_threshold(low_black=black_point,
                        low_white=white_point,
                        high_white=white_point,
                        high_black=black_point)
    img.save(filename='threshold_range_hard.jpg')
```

Original	Range Threshold (hard)
	

Note: Requires ImageMagick-7.0.8-41

3.21.8 White Threshold

New in version 0.5.2.

Force all pixels above a given pixel value to white. This works on a color channel-by-channel basis, and can be used to reduce unwanted colors.

```
from wand.image import Image

with Image(filename='inca_tern.jpg') as img:
    img.threshold_threshold(threshold='#ace')
    img.save(filename='threshold_white.jpg')
```

Original	White Threshold
	

3.22 CLI Reference

For users migrating old CLI scripts to python.

3.22.1 CLI Operators to Wand Methods

This table maps ImageMagick's CLI operators to Wand's *Image* methods.

CLI Operators	Wand Methods
-adaptive-blur	*wand.image.BaseImage.adaptive_blur()*
-adaptive-resize	*wand.image.BaseImage.adaptive_resize()*
-adaptive-sharpen	*wand.image.BaseImage.adaptive_sharpen()*
-annotate	*wand.image.BaseImage.annotate()*
-append	*wand.image.BaseImage.concat()*
-auto-gamma	*wand.image.BaseImage.auto_gamma()*
-auto-level	*wand.image.BaseImage.auto_level()*
-auto-orient	*wand.image.BaseImage.auto_orient()*
-auto-threshold	*wand.image.BaseImage.auto_threshold()*
-black-threshold	*wand.image.BaseImage.black_threshold()*
-blue-shift	*wand.image.BaseImage.blue_shift()*
-blur	*wand.image.BaseImage.blur()*

continues on next page

Table 1 – continued from previous page

CLI Operators	Wand Methods
-border	*wand.image.BaseImage.border()*
-brightness-contrast	*wand.image.BaseImage.brightness_contrast()*
-canny	*wand.image.BaseImage.canny()*
-caption	*wand.image.BaseImage.caption()*
-cdl	*wand.image.BaseImage.color_decision_list()*
-charcoal	*wand.image.BaseImage.charcoal()*
-chop	*wand.image.BaseImage.chop()*
-clahe	*wand.image.BaseImage.clahe()*
-clamp	*wand.image.BaseImage.clamp()*
-clut	*wand.image.BaseImage.clut()*
-coalesce	*wand.image.BaseImage.coalesce()*
-colorize	*wand.image.BaseImage.colorize()*
-colormap	*wand.image.BaseImage.color_map()*
-color-matrix	*wand.image.BaseImage.color_matrix()*
-color-threshold	*wand.image.BaseImage.color_threshold()*
-colorspace	*wand.image.BaseImage.transform_colorspace()*
-compare	*wand.image.BaseImage.compare()*
-complex	*wand.image.BaseImage.complex()*
-composite	*wand.image.BaseImage.composite()*
-connected-components	*wand.image.BaseImage.connected_components()*
-contrast	*wand.image.BaseImage.contrast()*
-contrast-stretch	*wand.image.BaseImage.contrast_stretch()*
-crop	*wand.image.BaseImage.crop()*
-cycle	*wand.image.BaseImage.cycle_color_map()*
-decipher	*wand.image.BaseImage.decipher()*
-deconstruct	*wand.image.BaseImage.deconstruct()*
-deskew	*wand.image.BaseImage.deskew()*
-despeckle	*wand.image.BaseImage.despeckle()*
-distort	*wand.image.BaseImage.distort()*
-edge	*wand.image.BaseImage.edge()*
-emboss	*wand.image.BaseImage.emboss()*
-encipher	*wand.image.BaseImage.encipher()*
-enhance	*wand.image.BaseImage.enhance()*
-equalize	*wand.image.BaseImage.equalize()*
-evaluate	*wand.image.BaseImage.evaluate()*
-extent	*wand.image.BaseImage.extent()*
-features	*wand.image.BaseImage.features()*
-fft	*wand.image.BaseImage.forward_fourier_transform()*
-flip	*wand.image.BaseImage.flip()*
-flop	*wand.image.BaseImage.flop()*
-frame	*wand.image.BaseImage.frame()*
-function	*wand.image.BaseImage.function()*
-fx	*wand.image.BaseImage.fx()*
-gamma	*wand.image.BaseImage.gamma()*
-gaussian-blur	*wand.image.BaseImage.gaussian_blur()*
-hald-clut	*wand.image.BaseImage.hald_clut()*
-hough-lines	*wand.image.BaseImage.hough_lines()*
-ift	*wand.image.BaseImage.inverse_fourier_transform()*
-implode	*wand.image.BaseImage.implode()*

continues on next page

Table 1 – continued from previous page

CLI Operators	Wand Methods
-kmeans	*wand.image.BaseImage.kmeans()*
-kuwahara	*wand.image.BaseImage.kuwahara()*
-lat	*wand.image.BaseImage.adaptive_threshold()*
-level	*wand.image.BaseImage.level()*
+level	*wand.image.BaseImage.levelize()*
-level-colors	*wand.image.BaseImage.level_colors()*
+level-colors	*wand.image.BaseImage.levelize_colors()*
-linear-stretch	*wand.image.BaseImage.linear_stretch()*
-liquid-rescale	*wand.image.BaseImage.liquid_rescale()*
-magnify	*wand.image.BaseImage.magnify()*
-mean-shift	*wand.image.BaseImage.mean_shift()*
-layers	*wand.image.BaseImage.merge_layers()*
-layers	*wand.image.Image.compare_layers()*
-layers	*wand.image.BaseImage.optimize_layers()*
-layers	*wand.image.BaseImage.optimize_transparency()*
-mode	*wand.image.BaseImage.mode()*
-modulate	*wand.image.BaseImage.modulate()*
-morphology	*wand.image.BaseImage.morphology()*
-motion-blur	*wand.image.BaseImage.motion_blur()*
-negate	*wand.image.BaseImage.negate()*
-noise	*wand.image.BaseImage.noise()*
-normalize	*wand.image.BaseImage.normalize()*
-paint	*wand.image.BaseImage.oil_paint()*
-opaque	*wand.image.BaseImage.opaque_paint()*
-ordered-dither	*wand.image.BaseImage.ordered_dither()*
-polaroid	*wand.image.BaseImage.polaroid()*
-polynomial	*wand.image.BaseImage.polynomial()*
-posterize	*wand.image.BaseImage.posterize()*
-quantize	*wand.image.BaseImage.quantize()*
-random-threshold	*wand.image.BaseImage.random_threshold()*
-range-threshold	*wand.image.BaseImage.range_threshold()*
-read-mask	*wand.image.BaseImage.read_mask()*
-remap	*wand.image.BaseImage.remap()*
-resample	*wand.image.BaseImage.resample()*
+repage	*wand.image.BaseImage.reset_coords()*
-resize	*wand.image.BaseImage.resize()*
-roll	*wand.image.BaseImage.roll()*
-rotate	*wand.image.BaseImage.rotate()*
-radial_blur	*wand.image.BaseImage.rotational_blur()*
-sample	*wand.image.BaseImage.sample()*
-scale	*wand.image.BaseImage.scale()*
-selective-blur	*wand.image.BaseImage.selective_blur()*
-sepia-tone	*wand.image.BaseImage.sepia_tone()*
-shade	*wand.image.BaseImage.shade()*
-shadow	*wand.image.BaseImage.shadow()*
-sharpen	*wand.image.BaseImage.sharpen()*
-shave	*wand.image.BaseImage.shave()*
-shear	*wand.image.BaseImage.shear()*
-sigmoidal-contrast	*wand.image.BaseImage.sigmoidal_contrast()*

continues on next page

Table 1 – continued from previous page

CLI Operators	Wand Methods
-similarity-threshold	See *wand.image.BaseImage.similarity()*
-subimage-search	See *wand.image.BaseImage.similarity()*
-sketch	*wand.image.BaseImage.sketch()*
-smush	*wand.image.BaseImage.smush()*
-solarize	*wand.image.BaseImage.solarize()*
-sparse-color	*wand.image.BaseImage.sparse_color()*
-splice	*wand.image.BaseImage.splice()*
-spread	*wand.image.BaseImage.spread()*
-stegano	*wand.image.BaseImage.stegano()*
-stereo	*wand.image.Image.stereogram()*
-statistic	*wand.image.BaseImage.statistic()*
-strip	*wand.image.BaseImage.strip()*
-swap	*wand.image.Image.image_swap()*
-swirl	*wand.image.BaseImage.swirl()*
-texture	*wand.image.BaseImage.texture()*
-threshold	*wand.image.BaseImage.threshold()*
-thumbnail	*wand.image.BaseImage.thumbnail()*
-tint	*wand.image.BaseImage.tint()*
-transform	*wand.image.BaseImage.transform()*
-transparent-color	*wand.image.BaseImage.transparent_color()*
-transpose	*wand.image.BaseImage.transpose()*
-transverse	*wand.image.BaseImage.transverse()*
-treedepth	See *wand.image.BaseImage.quantize()*
-trim	*wand.image.BaseImage.trim()*
-unique-colors	*wand.image.BaseImage.unique_colors()*
-unsharp-mask	*wand.image.BaseImage.unsharp_mask()*
-vignette	*wand.image.BaseImage.vignette()*
-watermark	*wand.image.BaseImage.watermark()*
-wave	*wand.image.BaseImage.wave()*
-wavelet-denoise	*wand.image.BaseImage.wavelet_denoise()*
-white-balance	*wand.image.BaseImage.white_balance()*
-white-threshold	*wand.image.BaseImage.white_threshold()*
-write-mask	*wand.image.BaseImage.write_mask()*

3.22.2 CLI Options to Wand Properties

This table list ImageMagick's options, and maps them to Wand's *Image* properties.

CLI Options	Wand Properties
-alpha	*wand.image.BaseImage.alpha_channel*
-antialias	*wand.image.BaseImage.font_antialias*
-antialias	*wand.image.BaseImage.antialias*
-attenuate	See *wand.image.BaseImage.noise()*
-background	*wand.image.BaseImage.background_color*
-blue-primary	*wand.image.BaseImage.blue_primary*
-bordercolor	*wand.image.BaseImage.border_color*
-colorspace	*wand.image.BaseImage.colorspace*
-compose	*wand.image.BaseImage.compose*

continues on next page

Table 2 – continued from previous page

CLI Options	Wand Properties
-compression	*wand.image.BaseImage.compression*
-delay	*wand.image.BaseImage.ticks_per_second*
-delay	*wand.sequence.SingleImage.delay*
-density	*wand.image.BaseImage.resolution*
-depth	*wand.image.BaseImage.depth*
-dispose	*wand.image.BaseImage.dispose*
-fill	wand.image.BaseImage.font_color
-font	*wand.image.BaseImage.font*
-format	*wand.image.BaseImage.format*
-fuzz	*wand.image.BaseImage.fuzz*
-gravity	*wand.image.BaseImage.gravity*
-green-primary	*wand.image.BaseImage.green_primary*
-intent	*wand.image.BaseImage.rendering_intent*
-interlace	*wand.image.BaseImage.interlace_scheme*
-interpolate	*wand.image.BaseImage.interpolate_method*
-loop	*wand.image.BaseImage.loop*
-mattecolor	*wand.image.BaseImage.matte_color*
-orientation	*wand.image.BaseImage.orientation*
-page	*wand.image.BaseImage.page*
-page	*wand.image.BaseImage.page_height*
-page	*wand.image.BaseImage.page_width*
-page	*wand.image.BaseImage.page_x*
-page	*wand.image.BaseImage.page_y*
-pointsize	*wand.image.BaseImage.font_size*
-quality	*wand.image.BaseImage.compression_quality*
-red-primary	*wand.image.BaseImage.red_primary*
-sampling-factor	*wand.image.BaseImage.sampling_factors*
-scene	*wand.image.BaseImage.scene*
-seed	*wand.image.BaseImage.seed*
-size	*wand.image.BaseImage.height*
-size	*wand.image.BaseImage.width*
-size	*wand.image.BaseImage.size*
-stroke	wand.image.BaseImage.stroke_color
-strokewidth	wand.image.BaseImage.stroke_width
-treedepth	See *wand.image.BaseImage.quantize()*
-type	*wand.image.BaseImage.type*
-units	*wand.image.BaseImage.units*
-virtual-pixel	*wand.image.BaseImage.virtual_pixel*
-white-point	*wand.image.BaseImage.white_point*

3.23 Running tests

Wand has unit tests and regression tests. It can be run using `setup.py` script:

```
$ python setup.py test
```

It uses pytest as its testing library. The above command will automatically install pytest as well if it's not installed yet.

Or you can manually install pytest and then use **`pytest`** command. It provides more options:

```
$ pip install pytest
$ pytest
```

3.23.1 Skipping tests

There are some time-consuming tests. You can skip these tests using `--skip-slow` option:

```
$ pytest --skip-slow
```

Be default, tests include regression testing for the PDF format. Test cases will fail if the system does not include Ghostscript binaries. You can skip PDF dependent tests with `--skip-pdf` option:

```
$ pytest --skip-pdf
```

The same behavior is true for Fourier Transform library. Use `--skip-fft` to skip over any discrete Fourier transformation test cases.

```
$ pytest --skip-fft
```

You can run only tests you want using `-k` option.

```
$ pytest -k image
```

The source code repository for Wand doesn't ship any *pytest.ini* configuration files. However nightly regression test are usually run in parallel with coverage reports. An example *pytest.ini* file might look like:

```
[pytest]
addopts=-n8 -rsfEw --cov wand --cov-report html
```

3.23.2 Using tox

Wand should be compatible with various Python implementations including CPython 2.7, 3.3, PyPy. tox is a testing software that helps Python packages to test on various Python implementations at a time.

It can be installed using **`pip`**:

```
$ pip install tox
```

If you type just **`tox`** at Wand directory it will be tested on multiple Python interpreters:

```
$ tox
GLOB sdist-make: /Users/emcconville/Desktop/wand/setup.py
py26 create: /Users/emcconville/Desktop/wand/.tox/py26
py26 installdeps: pytest
py26 sdist-inst: /Users/emcconville/Desktop/wand/.tox/dist/Wand-0.2.2.zip
py26 runtests: commands[0]
...
```

You can use a double `--` to pass options to pytest:

```
$ tox -- -k sequence
```

3.23.3 Continuous Integration

Travis CI automatically builds and tests every commit and pull request. The above banner image shows the current status of Wand build. You can see the detail of the current status from the following URL:

https://travis-ci.org/emcconville/wand

3.23.4 Code Coverage

Coveralls support tracking Wand's test coverage. The above banner image shows the current status of Wand coverage. You can see the details of the current status from the following URL:

https://coveralls.io/r/emcconville/wand

3.24 Roadmap

3.24.1 Very future versions

CFFI Wand will move to CFFI from ctypes.

PIL compatibility layer PIL has very long history and the most of Python projects still depend on it. We will work on PIL compatibility layer using Wand. It will provide two ways to emulate PIL:

- Module-level compatibility which can be used by changing `import`:

```
try:
    from wand.pilcompat import Image
except ImportError:
    from PIL import Image
```

- Global monkeypatcher which changes `sys.modules`:

```
from wand.pilcompat.monkey import patch; patch()
import PIL.Image  # it imports wand.pilcompat.Image module
```

CLI (`covert` command) to Wand compiler (#100) Primary interface of ImageMagick is **`convert`** command. It provides a small *parameter language*, and many answers on the Web contain code using this. The problem is that you can't simply copy-and-paste these code to utilize Wand.

This feature is to make these CLI codes possible to be used with Wand.

3.25 Wand Changelog

3.25.1 0.6 series

Version 0.6.10

Released on August 14th, 2022.

- Fixed segmentation fault during resource allocation on M1 processor. [#587]
- Fixed additional segmentation faults introduced with ImageMagick 7.1.0-45. [#587 & #586]

Version 0.6.9

Released on August 3rd, 2022.

- Updated *`Image.fx()`* method to raise *`WandRuntimeError`* if ImageMagick is unable to generate an image. [#582]
- Fixed *`Image.from_array()`* classmethod to handle Numpy's strided arrays. [#582]
- Fixed segmentation fault introduced with ImageMagick 7.1.0-45. [#586]

Version 0.6.8

Released on July 16th, 2022.

- Added *`Image.label()`* method.
- Added *`Image.region()`* method.
- Updated *`Image.chop()`* method to support `gravity` keyword.
- Updated *`Image.extent()`* method to support `gravity` keyword. [#554]
- Added *.so.9* shared library suffix to `wand.api.library_paths()` generator when searching `MAGICK_HOME` path.
- Added *`QUANTUM_SCALE`* constant.
- Added *`Image.montage()`* method. [#575]
- Added *`Image.roll()`* method.
- Fixed returned values for *`Image.connected_components()`* method for ImageMagick 7.1.1. [#574]
- Fixed `MagickSetImageDepth()` C-API method signature. [#577 by Pavel Borzenkov]
- Fixed *`Image.encipher()`* method to call the correct API. [#578 by Pavel Borzenkov]
- [DOC] Improved *`FontMetrics`* documentation. [#566]
- [TEST] Migrated CI from travis-ci.org to travis-ci.com.

- [TEST] Removed unneeded SVG dependency from regression test.
- [TEST] Suppressed *OptionWarning* when testing user errors.
- [TEST] Added Python 3.9 regression test for travis-ci.com.
- [TEST] Removed Python 3.7 & 3.8 regression test for travis-ci.com.
- [TEST] Added Python 3.10 regression tests for github actions.

Version 0.6.7

Released on August 16th, 2021.

- Added *Image.image_add()* method.
- Added *Image.image_get()* method.
- Added *Image.image_remove()* method.
- Added *Image.image_set()* method.
- Added *Image.image_swap()* method.
- Fixed sub-image extraction on read. [#532]
- Fixed *background_color* attribute when image was not read.
- [DOC] Completed *Distortion* guide. [#534]
- [DOC] Added *Morphology* guide.

Version 0.6.6

Released on February 28th, 2021.

- Added *Image.get_image_distortion()* method.
- Fixed *QuantumType* allocation for 32-bit architectures using HDRI. [#518]
- Fixed *MagickSizeType* allocation for *ResourceLimits.set_resource_limit()* and segfault with `armv7l` architecture. [#520]
- Fixed *Color* deallocation error on 32-bit architectures.
- Deprecated *wand.color.scale_quantum_to_int8()*
- [TEST] Deprecated PDF format from test assets.
- [TEST] Deprecated *Drawing* test *fx_wand* fixture to improve parallel CI testing.
- [TEST] Marked all ImageMagick-7 features skipped when running test suite with ImageMagick-6. [#522]

Version 0.6.5

Released on November 29th, 2020.

- Fixed memory allocation & deallocation bugs with PyPy3, and various memory leaks identified during regression testing. [#510]
- [TEST] Added Python 3.9 into Github regression tests. [#513 by Thijs Triemstra]

Version 0.6.4

Released on November 21st, 2020.

- Fixed *MagickFloatType* mapping for **s390x** architecture. [#504 & #505]
- Fixed image order when calling `wand.sequence.Sequence.__setitem__()` method. [#506]
- Fixed *Image.gaussian_blur()* method with `channel` parameter. [#507]
- Added *Image.color_threshold()* method.
- Added *Image.convex_hull()* method. Requires ImageMagick-7.0.10 or above.
- Added *Image.kmeans()* method. Only available with ImageMagick-7.0.10-37 or later.
- Added *Image.minimum_bounding_box()* method. Requires ImageMagick-7.0.10 or above;
- Added *Image.white_balance()* method. Only available with ImageMagick-7.0.10-37 or later.
- Added `percent_background` & `background_color` parameters to *Image.trim()* method.
- Added the following arguments to *Image.connected_components()*:
  - `angle_threshold`
  - `background_id`
  - `circularity_threshold`
  - `diameter_threshold`
  - `eccentricity_threshold`
  - `keep_colors`
  - `keep_top`
  - `major_axis_threshold`
  - `minor_axis_threshold`
  - `perimeter_threshold`
  - `remove_colors`
- Added `'inverse_log'` operator to *Image.evaluate()* method.
- Added `'rigidaffine'` operator to *Image.distort()* method. Requires ImageMagick-7.0.10 or above.
- Added *PAPERSIZE_MAP* dict as a convenience lookup table.
- Added support for setting *Image.page* attribute with papersizes defined in *PAPERSIZE_MAP*.
- [DOC] Created *Threshold* guide.
- [DOC] Created *Quantize* guide.

Version 0.6.3

Released on September 14th, 2020.

- Fixed buffer overflow bug in *Image.connected_components()* method. [#496]
- Added *Image.data_url()* method. [#489]
- Added *Image.sampling_factors* property. [#491]
- Added *Image.encipher()* & *Image.decipher()* methods.
- Argument `fuzz` for *Image.transparent_color()* now accepts `numbers.Real` numbers.
- Uniformed additional pre-read parameters between `Image.__init__()` & `Image.read()`.

Version 0.6.2

Released on July 6th, 2020.

- Added aspect cropping support for *Image.transform()* method.
- Added iterator methods to directly navigate the internal image-stack.
  - *Image.iterator_first()*
  - *Image.iterator_get()*
  - *Image.iterator_last()*
  - *Image.iterator_length()*
  - *Image.iterator_next()*
  - *Image.iterator_previous()*
  - *Image.iterator_reset()*
  - *Image.iterator_set()*
- Added `gray` & `cmyk` support for Numpy's array interface.
- Fixed *display()* on Windows & MacOS when previewing MIFF & XC formats.
- Fixed memory leak in *Image.transform()* for ImageMagick-6.
- Fixed animation preservation with *Image.transform()* method. [#251]
- Fixed *Image.interlace_scheme* property. [#488]
- [DOC] Make the documentation reproducible. [#484 by Chris Lamb]

Version 0.6.1

Released on May 15th, 2020.

- Fixed RuntimeError on deallocation when `allocation_map` changes. [#482 by Louis Sautier]

Version 0.6.0

Released on May 14th, 2020.

- Updated `numpy` array interface methods to accept / generate `shape` data values as `rows`, `columns`, and `channels`. This change should match other python-image numpy integrations. [#447]
- Added `adjoin`= argument to *`Image.save()`* method.
- Added `reset_coords`= argument to *`Image.trim()`* method. [#472]
- Added support for `atexit`'s shutdown routine. [#248 & #361]
- Added Python 2 classifiers to `MANIFEST.in`. [#462 by Thijs Triemstra]
- Added both test-cases and documentation to source distribution packages. [#7, #479, #480]
- Removed `README.rst` from `setup.py`. [#460]
- Rewrote memory allocation manager. [#300 & #312]
- Fixed segfault on macOS when invoking resource limits without calling `MagickWandGenesis()`.
- Fixed `grayscalealpha` spelling. [#463]
- Fixed *`Image.deskew()`* threshold argument. [#467]
- Fixed *`Image.alpha_channel`* property to apply changes to all images in the stack. [#468]
- Fixed *`Image.trim()`* paging offsets affected by 1x1 border. [#475]
- [TEST] Updated Travis CI environment to Ubuntu 18.04.04 LTS (Bionic)
- [TEST] Deprecated display fixtures.

3.25.2 0.5 series

Version 0.5.9

Released on February 10th, 2020.

- Fixed `dither` parameter in *`Image.quantize()`* method for ImageMagick-7.
- Added `Image.combine()` method. [Thanks Fred!]
- Check `__fspath__` attribute for `filename` parameter when calling *`Image.save()`*. [#452]
- Fixed typo in *`ProfileDict`* documentation. [#450 by Thijs Triemstra]
- Fixed typo in *`Resource.c_is_resource`* documentation. [#448]
- Updated broken sentence in *`Image.thumbnail()`* method. [#446]
- Check for `linux_distribution()` as method was removed in Python 3.8. [#456]
- Added *`Image.delay`* property. Previously only available with *`SingleImage`* class.

Version 0.5.8

Released on December 5th, 2019.

- Check `WAND_MAGICK_LIBRARY_SUFFIX` for additional library suffixes. [#436]
- Fixed `MagickCompareImagesLayers()` loading for ImageMagick-6 [#439]
- Fixed incorrect color values for first 5 pixels when exporting to `numpy.array` [#442]
- Updated example in *Image.annotate()* docstring. [#441 by alexgv]
- Fixed *Image.resolution* property to return a tuple of float values. [#444]
- Improved pycache performance by explicitly defining all ImageMagick warnings & errors in *wand.exceptions*. Previously all ImageMagick exceptions were generated dynamically during run-time.

Version 0.5.7

Released on September 3rd, 2019.

- Added *Image.color_decision_list()* method.
- Added *Image.contrast()* method.
- Added *Image.local_contrast()* method.
- Added *Image.ordered_dither()* method.
- Added *Image.random_threshold()* method.
- Added *Image.read_mask()* method. [#433]
- Added *Image.scale()* method.
- Added *Image.sepia_tone()* method.
- Added *Image.swirl()* method.
- Added *Image.write_mask()* method. [#433]
- Converted positional to key-word arguments to allow default values & allow more consistent behavior with CLI operations for the following methods:
  - *Image.blur()*
  - *Image.gaussian_blur()*
  - *Image.selective_blur()*
  - *Image.spread()*
  - *Image.unsharp_mask()*
- Restored #320 fix. [Reported by #435]
- Added `colorspace` & `units` argument to *Image* init. This is useful for defining sRGB ahead of reading CMYKA PDF documents.

Version 0.5.6

Released on August 2nd, 2019.

- Fixed invalid escape sequence warnings [#428]
- Fixed error on Drawing exception handling. [#427]
- Fixed undefined behavior when working with image frames in ImageMagick-7. [#431]
- Added *Image.annotate()* method. [#418]
- Added *Image.level_colors()* method.
- Added *Image.levelize_colors()* method.
- Added *Image.parse_meta_geometry()* method.
- Added *Image.percent_escape()* helper method. [#421]
- Added *Image.ping()* class method. [#425]
- Added `mean_color`, `keep`, & `remove` parameters in *Image.connected_components()* method.

Version 0.5.5

Released on July 8th, 2019.

- Rewrote *Image.contrast_stretch()* method to follow modern CLI behavior.
- Added *Image.chop()* method.
- Added *Image.clahe()* method.
- Added *Image.features()* method.
- Added *Image.forward_fourier_transform()* method.
- Added *Image.inverse_fourier_transform()* method.
- Added *Image.magnify()* method.
- Added `channel` parameter support for the following methods.
  - *Image.adaptive_blur()*
  - *Image.adaptive_sharpen()*
  - *Image.blur()*
  - *Image.brightness_contrast()*
  - *Image.clamp()*
  - *Image.clut()*
  - *Image.equalize()*
  - *Image.gaussian_blur()*
  - *Image.hald_clut()*
  - *Image.noise()*
  - *Image.morphology()*
  - *Image.opaque_paint()*
  - *Image.selective_blur()*

  - *Image.sharpen()*
  - *Image.sigmoidal_contrast()*
  - *Image.solarize()*
  - *Image.statistic()*
  - *Image.unsharp_mask()*
- Added support for new methods introduced with ImageMagick 7.0.8-41. Upgrade to the latest ImageMagick version to take advantage of the following features.
  - *Image.auto_threshold()*
  - *Image.canny()*
  - *Image.complex()*
  - *Image.connected_components()*
  - *Image.hough_lines()*
  - *Image.kuwahara()*
  - *Image.levelize()*
  - *Image.mean_shift()*
  - *Image.polynomial()*
  - *Image.range_threshold()*
  - *Image.seed*
  - *Image.wavelet_denoise()*

Version 0.5.4

Released on May 25th, 2019.

- Rewrote `libc` library loader. [#409]
- Respect `background` parameter in `Image.__init__()` constructor. [#410]
- Fixed *Drawing.get_font_metrics()* not raising internal ImageMagick exception on rendering error. [#411]
- Fixed deleting image artifact value.
- Fixed offset memory calculation in *Image.export_pixels()* & *Image.import_pixels()* methods. [#413]
- Added *Image.auto_gamma()* method.
- Added *Image.auto_level()* method.
- Added *Image.border_color* property.
- Added *Image.brightness_contrast()* method.
- Added *Image.mode()* method.
- Added *Image.motion_blur()* method.
- Added *Image.oil_paint()* method.
- Added *Image.opaque_paint()* method.
- Added *Image.polaroid()* method.

- Added *Image.rendering_intent* property.
- Added *Image.rotational_blur()* method.
- Added *Image.scene* property.
- Added *Image.shear()* method.
- Added *Image.sigmoidal_contrast()* method.
- Added *Image.similarity()* method.
- Added *Image.stegano()* method.
- Added *Image.stereogram()* class method.
- Added *Image.texture()* method.
- Added *Image.thumbnail()* method. [#357 by yoch]
- Added *Image.ticks_per_second* property.

Version 0.5.3

Released on April 20, 2019.

- Fixed alpha channel set to "on" & "off" values for ImageMagick-7. [#404]
- Updated *Image.composite* & *Image.composite_channel* to include optional arguments for composite methods that require extra controls.
- Updated *Image.composite* & *Image.composite_channel* to include optional gravity argument.
- **Support for numpy arrays. [#65]**
  - Added *Image.from_array* class method.
- **Support color map / palette manipulation. [#403]**
  - Added *Image.colors* property.
  - Added *Image.color_map()* method.
  - Added *Image.cycle_color_map()* method.
- Support for `highlight` & `lowlight` has been added to *Image.compare()* method.
- Support for PEP-519 for objects implementing `__fspath__`, in *encode_filename()*.
- Added *Image.adaptive_blur()* method.
- Added *Image.adaptive_resize()* method.
- Added *Image.adaptive_sharpen()* method.
- Added *Image.adaptive_threshold()* method.
- Added *Image.black_threshold()* method.
- Added *Image.blue_shift()* method.
- Added *Image.charcoal()* method.
- Added *Image.color_matrix()* method.
- Added *Image.colorize()* method.
- Added *Image.fuzz* property.

- Added *Image.kurtosis* property.
- Added *Image.kurtosis_channel()* method
- Added *Image.maxima* property.
- Added *Image.mean* property.
- Added *Image.mean_channel()* method
- Added *Image.minima* property.
- Added *Image.noise()* method.
- Added *Image.range_channel()* method
- Added *Image.remap()* method.
- Added *Image.selective_blur()* method.
- Added *Image.skewness* property.
- Added *Image.sketch()* method.
- Added *Image.smush()* method.
- Added *Image.sparse_color()* method.
- Added *Image.splice()* method.
- Added *Image.spread()* method.
- Added *Image.standard_deviation* property.
- Added *Image.statistic()* method.
- Added *Image.tint()* method.

Special thanks to Fred Weinhaus for helping test this release.

Version 0.5.2

Released on March 24, 2019.

- Import `collections.abc` explicitly. [#398 by Stefan Naumann]
- Fixed memory leak in *HistogramDict*. [#397]
- Fixed compression & compression quality bug. [#202 & #278]
- *Image.read()* will raise *WandRuntimeError* if `MagickReadImage()` returns `MagickFalse`, but does not emit exception. [#319]
- Added *Image.implode()* method.
- Added *Image.vignette()* method.
- Added *Image.wave()* method.
- Added *Image.white_threshold()* method.
- Added *Image.blue_primary* property.
- Added *Image.green_primary* property.
- Added *Image.interlace_scheme* property.
- Added *Image.interpolate_method* property.

- Added *Image.red_primary* property.
- Added *Image.white_point* property.

Version 0.5.1

Released on February 15, 2019.

- Added set pixel color via *Image[x, y] = Color('...')*. [#105]
- Added *limits* helper dictionary to allows getting / setting ImageMagick's resource-limit policies. [#97]
- Fixed segmentation violation for win32 & ImageMagick-7. [#389]
- Fixed *AssertError* by moving *SingleImage* sync behavior from `destroy` to context `__exit__`. [#388]
- Fixed memory leak in *get_font_metrics*. [#390]
- Added property setters for *Color* attributes.
- Added *cyan*, *magenta*, *yellow*, & *black* properties for CMYK *Color* instances.
- *Color* instance can be created from HSL values with *from_hsl()* class method.
- Added *Image.compose* property for identifying layer visibility.
- Added *Image.profiles* dictionary attribute. [#249]
- Moved `collections.abc` to *wand.compat.abc* for Python-3.8. [#394 by Tero Vuotila]
- Update *wand.display* to use Python3 compatible `print()` function. [#395 by Tero Vuotila]

Version 0.5.0

Released on January 1, 2019.

- Support for ImageMagick-7.
- Improved support for 32-bit systems.
- Improved support for non-Q16 libraries.
- Removed *README.rst* from setup.py's *data_files*. [#336]
- Improved *EXIF:ORIENTATION* handling. [#364 by M. Skrzypek]
- Tolerate failures while accessing wand.api. [#220 by Utkarsh Upadhyay]
- Added support for Image Artifacts through *Image.artifacts*. [#369]
- Added optional stroke color/width parameters for *Font*.
- Image layers support (#22)
  - Added *Image.coalesce()* method.
  - Added *Image.deconstruct* method.
  - Added *Image.dispose* property.
  - Added *Image.optimize_layers()* method.
  - Added *Image.optimize_transparency()* method.
- Implemented `__array_interface__()` for NumPy [#65]
- Migrated the following methods & attributes from *Image* to *BaseImage* for a more uniformed code-base.

  - *Image.compression*
  - *Image.format*
  - *Image.auto_orient()*
  - *Image.border()*
  - *Image.contrast_stretch()*
  - *Image.gamma()*
  - *Image.level()*
  - *Image.linear_stretch()*
  - *Image.normalize()*
  - *Image.strip()*
  - *Image.transpose()*
  - *Image.transverse()*
  - *Image.trim()*
- Added *Image.clut()* method.
- Added *Image.concat()* method. [#177]
- Added *Image.deskew()* method.
- Added *Image.despeckle()* method.
- Added *Image.edge()* method.
- Added *Image.emboss()* method. [#196]
- Added *Image.enhance()* method. [#132]
- Added *Image.export_pixels()* method.
- Added *Image.import_pixels()* method.
- Added *Image.morphology()* method. [#132]
- Added *Image.posterize()* method.
- Added *Image.shade()* method.
- Added *Image.shadow()* method.
- Added *Image.sharpen()* method. [#132]
- Added *Image.shave()* method.
- Added *Image.unique_colors()* method.
- Method *Drawing.draw()* now accepts *BaseImage* for folks extended classes.
- Added *Image.loop* property. [#227]
- Fixed *SingleImage.delay* property. [#153]
- Attribute *Image.font_antialias* has been deprecated in favor of *Image.antialias*. [#218]
- Fixed ordering of *COMPRESSION_TYPES* based on ImageMagick version. [#309]
- Fixed drawing on *SingleImage*. [#289]
- Fixed wrapping issue for larger offsets when using *gravity* kwarg in *Image.crop()* method. [#367]

3.25.3 0.4 series

Version 0.4.5

Released on November 12, 2018.

- Improve library searching when `MAGICK_HOME` environment variable is set. [#320 by Chase Anderson]
- Fixed misleading *TypeError: object of type 'NoneType' has no len()* during destroy routines. [#346 by Carey Metcalfe]
- Added *`Image.blur()`* method (`MagickBlurImage()`). [#311 by Alexander Karpinsky]
- Added *`Image.extent()`* method (`MagickExtentImage()`). [#233 by Jae-Myoung Yu]
- Added *`Image.resample()`* method (`MagickResampleImage()`). [#244 by Zio Tibia]

Version 0.4.4

Released on October 22, 2016.

- Added *`BaseError`*, *`BaseWarning`*, and *`BaseFatalError`*, base classes for domains. [#292]
- Fixed `TypeError` during parsing version caused by format change of ImageMagick version string (introduced by 6.9.6.2). [#310, Debian bug report #841548]
- Properly fixed again memory-leak when accessing images constructed in *`Image.sequence[]`*. It had still leaked memory in the case an image is not closed using `with` but manual *`wand.resource.Resource.destroy()`*/*`wand.image.Image.close()`* method call. [#237]

Version 0.4.3

Released on June 1, 2016.

- Fixed `repr()` for empty *`Image`* objects. [#265]
- Added *`Image.compare()`* method (`MagickCompareImages()`). [#238, #268 by Gyusun Yeom]
- Added *`Image.page`* and related properties for virtual canvas handling. [#284 by Dan Harrison]
- Added *`Image.merge_layers()`* method (`MagickMergeImageLayers()`). [#281 by Dan Harrison]
- Fixed `OSError` during import `libc.dylib` due to El Capitan's SIP protection. [#275 by Ramesh Dharan]

Version 0.4.2

Released on November 30, 2015.

- Fixed `ImportError` on MSYS2. [#257 by Eon Jeong]
- Added *`Image.quantize()`* method (`MagickQuantizeImage()`). [#152 by Kang Hyojun, #262 by Jeong Yun-Won]
- Added *`Image.transform_colorspace()`* quantize (`MagickTransformImageColorspace()`). [#152 by Adrian Jung, #262 by Jeong YunWon]
- Now ImageMagick DLL can be loaded on Windows even if its location is stored in the registry. [#261 by Roeland Schoukens]
- Added `depth` parameter to *`Image`* constructor. The `depth`, `width` and `height` parameters can be used with the `filename`, `file` and `blob` parameters to load raw pixel data. [#261 by Roeland Schoukens]

Version 0.4.1

Released on August 3, 2015.

- Added *Image.auto_orient()* that fixes orientation by checking EXIF tags.
- Added *Image.transverse()* method (`MagickTransverseImage()`).
- Added *Image.transpose()* method (`MagickTransposeImage()`).
- Added *Image.evaluate()* method.
- Added *Image.frame()* method.
- Added *Image.function()* method.
- Added *Image.fx()* expression method.
- Added `gravity` options in *Image.crop()* method. [#222 by Eric McConville]
- Added *Image.matte_color* property.
- Added *Image.virtual_pixel* property.
- Added *Image.distort()* method.
- Added *Image.contrast_stretch()* method.
- Added *Image.gamma()* method.
- Added *Image.linear_stretch()* method.
- Additional support for *Image.alpha_channel*.
- Additional query functions have been added to *wand.version* API. [#120]
  - Added *configure_options()* function.
  - Added *fonts()* function.
  - Added *formats()* function.
- Additional IPython support. [#117]
  - Render RGB *Color* preview.
  - Display each frame in image *Sequence*.
- Fixed memory-leak when accessing images constructed in *Image.sequence[]*. [#237 by Eric McConville]
- Fixed Windows memory-deallocate errors on *wand.drawing* API. [#226 by Eric McConville]
- Fixed `ImportError` on FreeBSD. [#252 by Pellaeon Lin]

Version 0.4.0

Released on February 20, 2015.

See also:

whatsnew/0.4 This guide introduces what's new in Wand 0.4.

- Complete *wand.drawing* API. The whole work was done by Eric McConville. Huge thanks for his effort! [#194 by Eric McConville]
  - Added *Drawing.arc()* method (*Arc*).
  - Added *Drawing.bezier()* method (*Bezier*).

- Added *Drawing.circle()* method (*Circle*).
- *Color & Matte*
  - Added *wand.drawing.PAINT_METHOD_TYPES* constant.
  - Added *Drawing.color()* method.
  - Added *Drawing matte()* method.
- Added *Drawing.composite()* method (*Composite*).
- Added *Drawing.ellipse()* method (*Ellipse*).
- *Paths*
  - Added *path_start()* method.
  - Added *path_finish()* method.
  - Added *path_close()* method.
  - Added *path_curve()* method.
  - Added *path_curve_to_quadratic_bezier()* method.
  - Added *path_elliptic_arc()* method.
  - Added *path_horizontal_line()* method.
  - Added *path_line()* method.
  - Added *path_move()* method.
  - Added *path_vertical_line()* method.
- Added *Drawing.point()* method (*Point*).
- Added *Drawing.polygon()* method (*Polygon*).
- Added *Drawing.polyline()* method (*Polyline*).
- *Push & Pop*
  - Added *push()* method.
  - Added *push_clip_path()* method.
  - Added *push_defs()* method.
  - Added *push_pattern()* method.
  - Added *clip_path* property.
  - Added *set_fill_pattern_url()* method.
  - Added *set_stroke_pattern_url()* method.
  - Added *pop()* method.
- Added *Drawing.rectangle()* method (*Rectangles*).
- Added *stroke_dash_array* property.
- Added *stroke_dash_offset* property.
- Added *stroke_line_cap* property.
- Added *stroke_line_join* property.
- Added *stroke_miter_limit* property.

  - Added *stroke_opacity* property.
  - Added *stroke_width* property.
  - Added *fill_opacity* property.
  - Added *fill_rule* property.
- Error message of *MissingDelegateError* raised by *Image.liquid_rescale()* became nicer.

3.25.4 0.3 series

Version 0.3.9

Released on December 20, 2014.

- Added `'pdf:use-cropbox'` option to *Image.options* dictionary (and `OPTIONS` constant). [#185 by Christoph Neuroth]
- Fixed a bug that exception message was `bytes` instead of `str` on Python 3.
- The `size` parameter of *Font* class becomes optional. Its default value is 0, which means *autosized*. [#191 by Cha, Hojeong]
- Fixed a bug that *Image.read()* had tried using `MagickReadImageFile()` even when the given file object has no `mode` attribute. [#205 by Stephen J. Fuhry]

Version 0.3.8

Released on August 3, 2014.

- Fixed a bug that transparent background becomes filled with white when SVG is converted to other bitmap image format like PNG. [#184]
- Added *Image.negate()* method. [#174 by Park Joon-Kyu]
- Fixed a segmentation fault on *Image.modulate()* method. [#173 by Ted Fung, #158]
- Added suggestion to install freetype also if Homebrew is used. [#141]
- Now *image/x-gif* also is determined as `animation`. [#181 by Juan-Pablo Scaletti]

Version 0.3.7

Released on March 25, 2014.

- A hotfix of debug prints made at 0.3.6.

Version 0.3.6

Released on March 23, 2014.

- Added *Drawing.rectangle()* method. *Now you can draw rectangles.* [#159]
- Added *Image.compression* property. [#171]
- Added `contextlib.nested()` function to *wand.compat* module.
- Fixed `UnicodeEncodeError` when *Drawing.text()* method gives Unicode `text` argument in Python 2. [#163]

- Now it now allows to use Wand when Python is invoked with the `-OO` flag. [#169 by Samuel Maudo]

Version 0.3.5

Released on September 13, 2013.

- Fix segmentation fault on *`Image.save()`* method. [#150]

Version 0.3.4

Released on September 9, 2013.

- Added *`Image.modulate()`* method. [#134 by Dan P. Smith]
- Added *`Image.colorspace`* property. [#135 by Volodymyr Kuznetsov]
- Added *`Image.unsharp_mask()`* method. [#136 by Volodymyr Kuznetsov]
- Added `'jpeg:sampling-factor'` option to *`Image.options`* dictionary (and `OPTIONS` constant). [#137 by Volodymyr Kuznetsov]
- Fixed ImageMagick shared library resolution on Arch Linux. [#139, #140 by Sergey Tereschenko]
- Added *`Image.sample()`* method. [#142 by Michael Allen]
- Fixed a bug that *`Image.save()`* preserves only one frame of the given animation when file-like object is passed. [#143, #145 by Michael Allen]
- Fixed searching of ImageMagick shared library with HDR support enabled. [#148, #149 by Lipin Dmitriy]

Version 0.3.3

Released on August 4, 2013. It's author's birthday.

- Added *`Image.gaussian_blur()`* method.
- Added *`Drawing.stroke_color`* property. [#129 by Zeray Rice]
- Added *`Drawing.stroke_width`* property. [#130 by Zeray Rice]
- Fixed a memory leak of *`Color`* class. [#127 by Wieland Morgenstern]
- Fixed a bug that *`Image.save()`* to stream truncates data. [#128 by Michael Allen]
- Fixed broken *`display()`* on Python 3. [#126]

Version 0.3.2

Released on July 11, 2013.

- Fixed incorrect encoding of filenames. [#122]
- Fixed key type of *`Image.metadata`* dictionary to `str` from `bytes` in Python 3.
- Fixed CentOS compatibility [#116, #124 by Pierre Vanliefland]
  - Made `DrawSetTextInterlineSpacing()` and `DrawGetTextInterlineSpacing()` optional.
  - Added exception in drawing API when trying to use `DrawSetTextInterlineSpacing()` and `DrawGetTextInterlineSpacing()` functions when they are not available.
  - Added *`WandLibraryVersionError`* class for library versions issues.

Version 0.3.1

Released on June 23, 2013.

- Fixed `ImportError` on Windows.

Version 0.3.0

Released on June 17, 2013.

See also:

whatsnew/0.3 This guide introduces what's new in Wand 0.3.

- Now also works on Python 2.6, 2.7, and 3.2 or higher.
- Added *wand.drawing* module. [#64 by Adrian Jung]
- Added *Drawing.get_font_metrics()* method. [#69, #71 by Cha, Hojeong]
- Added *Image.caption()* method. [#74 by Cha, Hojeong]
- Added optional `color` parameter to *Image.trim()* method.
- Added *Image.border()* method. [2496d37f75d75e9425f95dde07033217dc8afefc by Jae-Myoung Yu]
- Added `resolution` parameter to *Image.read()* method and the constructor of *Image*. [#75 by Andrey Antukh]
- Added *Image.liquid_rescale()* method which does seam carving. See also *Seam carving (also known as content-aware resizing)*.
- Added *Image.metadata* immutable mapping attribute and *Metadata* mapping type for it. [#56 by Michael Elovskikh]
- Added *Image.channel_images* immutable mapping attribute and *ChannelImageDict* mapping for it.
- Added *Image.channel_depths* immutable mapping attribute and *ChannelDepthDict* mapping for it.
- Added *Image.composite_channel()* method.
- Added *Image.read()* method. [#58 by Piotr Florczyk]
- Added *Image.resolution* property. [#58 by Piotr Florczyk]
- Added *Image.blank()* method. [#60 by Piotr Florczyk]
- Fixed several memory leaks. [#62 by Mitch Lindgren]
- Added *ImageProperty* mixin class to maintain a weak reference to the parent image.
- Ranamed `wand.image.COMPOSITE_OPS` to *COMPOSITE_OPERATORS*.
- Now it shows helpful error message when ImageMagick library cannot be found.
- Added IPython-specialized formatter.
- Added *QUANTUM_DEPTH* constant.
- Added these properties to *Color* class:
  - *red_quantum*
  - *green_quantum*
  - *blue_quantum*
  - *alpha_quantum*

  - *red_int8*
  - *green_int8*
  - *blue_int8*
  - *alpha_int8*
- Added *Image.normalize()* method. [#95 by Michael Curry]
- Added *Image.transparent_color()* method. [#98 by Lionel Koenig]
- Started supporting resizing and cropping of GIF images. [#88 by Bear Dong, #112 by Taeho Kim]
- Added *Image.flip()* method.
- Added *Image.flop()* method.
- Added *Image.orientation* property. [88574468a38015669dae903185fb328abdd717c0 by Taeho Kim]
- *wand.resource.DestroyedResourceError* becomes a subtype of *wand.exceptions.WandException*.
- *Color* is now hashable, so can be used as a key of dictionaries, or an element of sets. [#114 by klutzy]
- *Color* has *normalized_string* property.
- *Image* has *histogram* dictionary.
- Added optional `fuzz` parameter to *Image.trim()* method. [#113 by Evaldo Junior]

3.25.5 0.2 series

Version 0.2.4

Released on May 28, 2013.

- Fix `NameError` in *Resource.resource* setter. [#89 forwarded from Debian bug report #699064 by Jakub Wilk]
- Fix the problem of library loading for Mac with Homebrew and Arch Linux. [#102 by Roel Gerrits, #44]

Version 0.2.3

Released on January 25, 2013.

- Fixed a bug that *Image.transparentize()* method (and *Image.watermark()* method which internally uses it) didn't work.
- Fixed segmentation fault occurred when *Color.red*, *Color.green*, or *Color.blue* is accessed.
- Added *Color.alpha* property.
- Fixed a bug that format converting using *Image.format* property or *Image.convert()* method doesn't correctly work to save blob.

Version 0.2.2

Released on September 24, 2012.

- A compatibility fix for FreeBSD. [Patch by Olivier Duchateau]
- Now *Image* can be instantiated without any opening. Instead, it can take `width`/`height` and `background`. [#53 by Michael Elovskikh]
- Added *Image.transform()* method which is a convenience method accepting geometry strings to perform cropping and resizing. [#50 by Mitch Lindgren]
- Added *Image.units* property. [#45 by Piotr Florczyk]
- Now *Image.resize()* method raises a proper error when it fails for any reason. [#41 by Piotr Florczyk]
- Added *Image.type* property. [#33 by Yauhen Yakimovich, #42 by Piotr Florczyk]

Version 0.2.1

Released on August 19, 2012. Beta version.

- Added *Image.trim()* method. [#26 by Jökull Sólberg Auðunsson]
- Added *Image.depth* property. [#31 by Piotr Florczyk]
- Now *Image* can take an optional `format` hint. [#32 by Michael Elovskikh]
- Added *Image.alpha_channel* property. [#35 by Piotr Florczyk]
- The default value of *Image.resize()*'s `filter` option has changed from `'triangle'` to `'undefined'`. [#37 by Piotr Florczyk]
- Added version data of the linked ImageMagick library into *wand.version* module:
  - *MAGICK_VERSION* (`GetMagickVersion()`)
  - *MAGICK_VERSION_INFO* (`GetMagickVersion()`)
  - *MAGICK_VERSION_NUMBER* (`GetMagickVersion()`)
  - *MAGICK_RELEASE_DATE* (`GetMagickReleaseDate()`)
  - *MAGICK_RELEASE_DATE_STRING* (`GetMagickReleaseDate()`)

Version 0.2.0

Released on June 20, 2012. Alpha version.

- Added *Image.transparentize()* method. [#19 by Jeremy Axmacher]
- Added *Image.composite()* method. [#19 by Jeremy Axmacher]
- Added *Image.watermark()* method. [#19 by Jeremy Axmacher]
- Added *Image.quantum_range* property. [#19 by Jeremy Axmacher]
- Added *Image.reset_coords()* method and `reset_coords` option to *Image.rotate()* method. [#20 by Juan Pablo Scaletti]
- Added *Image.strip()* method. [#23 by Dmitry Vukolov]
- Added *Image.compression_quality* property. [#23 by Dmitry Vukolov]
- Now the current version can be found from the command line interface: `python -m wand.version`.

3.25.6 0.1 series

Version 0.1.10

Released on May 8, 2012. Still alpha version.

- So many Windows compatibility issues are fixed. [#14 by John Simon]
- Added *wand.api.libmagick*.
- Fixed a bug that raises `AttributeError` when it's trying to warn. [#16 by Tim Dettrick]
- Now it throws `ImportError` instead of `AttributeError` when the shared library fails to load. [#17 by Kieran Spear]
- Fixed the example usage on index page of the documentation. [#18 by Jeremy Axmacher]

Version 0.1.9

Released on December 23, 2011. Still alpha version.

- Now *wand.version.VERSION_INFO* becomes `tuple` and *wand.version.VERSION* becomes a string.
- Added *Image.background_color* property.
- Added == operator for *Image* type.
- Added `hash()` support of *Image* type.
- Added *Image.signature* property.
- Added *wand.display* module.
- Changed the theme of Sphinx documentation.
- Changed the start example of the documentation.

Version 0.1.8

Released on December 2, 2011. Still alpha version.

- Wrote some guide documentations: *Reading images*, *Writing images* and *Resizing and cropping*.
- Added *Image.rotate()* method for in-place rotation.
- Made *Image.crop()* to raise proper `ValueError` instead of `IndexError` for invalid width/height arguments.
- Changed the type of *Image.resize()* method's `blur` parameter from `numbers.Rational` to `numbers.Real`.
- Fixed a bug of raising `ValueError` when invalid `filter` has passed to *Image.resize()* method.

Version 0.1.7

Released on November 10, 2011. Still alpha version.

- Added *Image.mimetype* property.
- Added *Image.crop()* method for in-place crop.

Version 0.1.6

Released on October 31, 2011. Still alpha version.

- Removed a side effect of *Image.make_blob()* method that changes the image format silently.
- Added *Image.format* property.
- Added *Image.convert()* method.
- Fixed a bug about Python 2.6 compatibility.
- Use the internal representation of `PixelWand` instead of the string representation for *Color* type.

Version 0.1.5

Released on October 28, 2011. Slightly mature alpha version.

- Now *Image* can read Python file objects by `file` keyword argument.
- Now *Image.save()* method can write into Python file objects by `file` keyword argument.
- *Image.make_blob()*'s `format` argument becomes omittable.

Version 0.1.4

Released on October 27, 2011. Hotfix of the malformed Python package.

Version 0.1.3

Released on October 27, 2011. Slightly mature alpha version.

- Pixel getter for *Image*.
- Row getter for *Image*.
- Mac compatibility.
- Windows compatibility.
- 64-bit processor compatibility.

Version 0.1.2

Released on October 16, 2011. Still alpha version.

- *Image* implements iterable interface.
- Added *wand.color* module.
- Added the abstract base class of all Wand resource objects: *wand.resource.Resource*.
- *Image* implements slicing.
- Cropping *Image* using its slicing operator.

Version 0.1.1

Released on October 4, 2011. Still alpha version.

- Now it handles errors and warnings properly and in natural way of Python.
- Added *Image.make_blob()* method.
- Added `blob` parameter into *Image* constructor.
- Added *Image.resize()* method.
- Added *Image.save()* method.
- Added *Image.clone()* method.
- Drawed the pretty logo picture (thanks to Hyojin Choi).

Version 0.1.0

Released on October 1, 2011. Very alpha version.

3.26 Talks and Presentations

3.26.1 Talks in 2012

- Lightning talk at Python Korea November 2012

CHAPTER

FOUR

REFERENCES

4.1 wand — Simple MagickWand API binding for Python

4.1.1 wand.image — Image objects

Opens and manipulates images. Image objects can be used in `with` statement, and these resources will be automatically managed (even if any error happened):

```
with Image(filename='pikachu.png') as i:
    print('width =', i.width)
    print('height =', i.height)
```

wand.image.**ALPHA_CHANNEL_TYPES = ('undefined', 'activate', 'associate', 'background', 'copy', 'deactivate', 'discrete', 'disassociate', 'extract', 'off', 'on', 'opaque', 'remove', 'set', 'shape', 'transparent')**

(`tuple`) The list of *alpha_channel* types.

- `'undefined'`
- `'activate'`
- `'background'`
- `'copy'`
- `'deactivate'`
- `'discrete'` - Only available in ImageMagick-7
- `'extract'`
- `'off'` - Only available in ImageMagick-7
- `'on'` - Only available in ImageMagick-7
- `'opaque'`
- `'reset'` - Only available in ImageMagick-6
- `'set'`
- `'shape'`
- `'transparent'`
- `'flatten'` - Only available in ImageMagick-6
- `'remove'`

See also:

ImageMagick Image Channel Describes the SetImageAlphaChannel method which can be used to modify alpha channel. Also describes AlphaChannelType

wand.image.**AUTO_THRESHOLD_METHODS = ('undefined', 'kapur', 'otsu', 'triangle')**
(tuple) The list of methods used by *Image.auto_threshold()*

- 'undefined'
- 'kapur'
- 'otsu'
- 'triangle'

New in version 0.5.5.

class wand.image.**ArtifactTree**(*image*)
Splay tree to map image artifacts. Values defined here are intended to be used elseware, and will not be written to the encoded image.

For example:

```
# Omit timestamp from PNG file headers.
with Image(filename='input.png') as img:
    img.artifacts['png:exclude-chunks'] = 'tIME'
    img.save(filename='output.png')
```

Parameters **image** (*Image*) – an image instance

Note: You don't have to use this by yourself. Use *Image.artifacts* property instead.

New in version 0.5.0.

class wand.image.**BaseImage**(*wand*)
The abstract base of *Image* (container) and *SingleImage*. That means the most of operations, defined in this abstract class, are possible for both *Image* and *SingleImage*.

New in version 0.3.0.

adaptive_blur(*radius=0.0*, *sigma=0.0*, *channel=None*)
Adaptively blurs the image by decreasing Gaussian as the operator approaches detected edges.

See Example of *Adaptive Blur*.

Parameters

- **radius** (numbers.Real) – size of gaussian aperture.
- **sigma** (numbers.Real) – Standard deviation of the gaussian filter.
- **channel** (basestring) – Apply the blur effect on a specific channel. See *CHANNELS*.

New in version 0.5.3.

Changed in version 0.5.5: Added optional `channel` argument

adaptive_resize(*columns=None*, *rows=None*)
Adaptively resize image by applying Mesh interpolation.

Parameters

- **columns** (numbers.Integral) – width of resized image.

- **rows** (numbers.Integral) – height of resized image.

New in version 0.5.3.

adaptive_sharpen(*radius=0.0*, *sigma=0.0*, *channel=None*)
Adaptively sharpens the image by sharpening more intensely near image edges and less intensely far from edges.

See Example of *Adaptive Sharpen*.

Parameters

- **radius** (numbers.Real) – size of gaussian aperture.
- **sigma** (numbers.Real) – Standard deviation of the gaussian filter.
- **channel** (basestring) – Apply the sharpen effect on a specific channel. See *CHANNELS*.

New in version 0.5.3.

Changed in version 0.5.5: Added optional `channel` argument

adaptive_threshold(*width*, *height*, *offset=0.0*)
Applies threshold for each pixel based on neighboring pixel values.

Parameters

- **width** (numbers.Integral) – size of neighboring pixels on the X-axis.
- **height** (numbers.Integral) – size of neighboring pixels on the Y-axis.
- **offset** (numbers.Real) – normalized number between *0.0* and *quantum_range*. Forces the pixels to black if values are below `offset`.

New in version 0.5.3.

property alpha_channel
(bool) Get state of image alpha channel. It can also be used to enable/disable alpha channel, but with different behavior such as new, copied, or existing.

Behavior of setting *alpha_channel* is defined with the following values:

- **'activate', 'on', or True will enable an images** alpha channel. Existing alpha data is preserved.
- **'deactivate', 'off', or False will disable an images** alpha channel. Any data on the alpha will be preserved.
- **'associate' & 'disassociate' toggle alpha channel flag in** certain image-file specifications.
- 'set' enables and resets any data in an images alpha channel.
- **'opaque' enables alpha/matte channel, and forces full opaque** image.
- **'transparent' enables alpha/matte channel, and forces full** transparent image.
- **'extract' copies data in alpha channel across all other channels,** and disables alpha channel.
- **'copy' calculates the gray-scale of RGB channels,** and applies it to alpha channel.
- **'shape' is identical to 'copy', but will color the resulting** image with the value defined with *background_color*.
- 'remove' will composite *background_color* value.
- **'background' replaces full-transparent color with background** color.

Note: The *alpha_channel* attribute will always return `True` if alpha channel is enabled, and `False` otherwise. Setting this property with a string value from *ALPHA_CHANNEL_TYPES* will resolve to a `bool` after applying channel operations listed above.

With ImageMagick-6, values `'on'` & `'off'` are aliased to `'activate'` & `'deactivate'`. However in ImageMagick-7, both `'on'` & `'off'` have their own behavior.

New in version 0.2.1.

Changed in version 0.4.1: Support for additional setting values. However `Image.alpha_channel` will continue to return `bool` if the current alpha/matte state is enabled.

Changed in version 0.6.0: Setting the alpha channel will apply the change to all frames in the image stack.

property animation

(`bool`) Whether the image is animation or not. It doesn't only mean that the image has two or more images (frames), but all frames are even the same size. It's about image format, not content. It's `False` even if *image/ico* consits of two or more images of the same size.

For example, it's `False` for *image/jpeg*, *image/gif*, *image/ico*.

If *image/gif* has two or more frames, it's `True`. If *image/gif* has only one frame, it's `False`.

New in version 0.3.0.

Changed in version 0.3.8: Became to accept *image/x-gif* as well.

annotate(*text*, *drawing_wand*, *left=0*, *baseline=0*, *angle=0*)

Draws text on an image. This method differs from *caption()* as it uses *Drawing* class to manage the font configuration & style context.

```
from wand.drawing import Drawing
from wand.image import Image

with Image(filename='input.jpg') as img:
    with Drawing() as ctx:
        ctx.font_family = 'Times New Roman, Nimbus Roman No9'
        ctx.font_size = 18
        ctx.text_decoration = 'underline'
        ctx.text_kerning = -1
        img.annotate('Hello World', ctx, left=20, baseline=50)
    img.save(filename='output.jpg')
```

Parameters

- **text** (*wand.drawing.Drawing*) – String to annotate on image.
- **drawing_wand** – Font configuration & style context.
- **left** (`numbers.Real`) – X-axis offset of the text baseline.
- **baseline** (`numbers.Real`) – Y-axis offset of the bottom of the text.
- **angle** (`numbers.Real`) – Degree rotation to draw text at.

New in version 0.5.6.

property antialias

(`bool`) If vectors & fonts will use anti-aliasing.

Changed in version 0.5.0: Previously named *font_antialias*.

auto_gamma()
Adjust the gamma level of an image.

New in version 0.5.4.

auto_level()
Scale the minimum and maximum values to a full quantum range.

New in version 0.5.4.

auto_orient()
Adjusts an image so that its orientation is suitable for viewing (i.e. top-left orientation). If available it uses `MagickAutoOrientImage()` (was added in ImageMagick 6.8.9+) if you have an older magick library, it will use `_auto_orient()` method for fallback.

New in version 0.4.1.

auto_threshold(*method='kapur'*)
Automatically performs threshold method to reduce grayscale data down to a binary black & white image. Included algorithms are Kapur, Otsu, and Triangle methods.

> **Warning:** This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters **method** (`basestring`) – Which threshold method to apply. See *AUTO_THRESHOLD_METHODS*. Defaults to `'kapur'`.

Raises *WandLibraryVersionError* – if function is not available on system's library.

New in version 0.5.5.

property background_color
(*wand.color.Color*) The image background color. It can also be set to change the background color.

New in version 0.1.9.

Changed in version 0.6.7: Allow property to be set before image read.

black_threshold(*threshold*)
Forces all pixels above a given color as black. Leaves pixels above threshold unaltered.

Parameters **threshold** (`Color`) – Color to be referenced as a threshold.

New in version 0.5.3.

property blue_primary
(`tuple`) The chromatic blue primary point for the image. With ImageMagick-6 the primary value is `(x, y)` coordinates; however, ImageMagick-7 has `(x, y, z)`.

New in version 0.5.2.

blue_shift(*factor=1.5*)
Mutes colors of the image by shifting blue values.

See Example of *Blue Shift*

Parameters **factor** (`numbers.Real`) – Amount to adjust values.

New in version 0.5.3.

blur(*radius=0.0*, *sigma=0.0*, *channel=None*)
Blurs the image. Convolve the image with a gaussian operator of the given `radius` and standard deviation (`sigma`). For reasonable results, the `radius` should be larger than `sigma`. Use a `radius` of 0 and *blur()* selects a suitable `radius` for you.

See Example of *Blur*.

Parameters

- **radius** (numbers.Real) – the radius of the, in pixels, not counting the center pixel. Default is `0.0`.
- **sigma** (numbers.Real) – the standard deviation of the, in pixels. Default value is `0.0`.
- **channel** (basestring) – Optional color channel to apply blur. See *CHANNELS*.

New in version 0.4.5.

Changed in version 0.5.5: Added optional `channel` argument.

Changed in version 0.5.7: Positional arguments `radius` & `sigman` have been converted to key-word arguments.

border(*color*, *width*, *height*, *compose='copy'*)
Surrounds the image with a border.

Parameters

- **bordercolor** – the border color pixel wand
- **width** (numbers.Integral) – the border width
- **height** (numbers.Integral) – the border height
- **compose** (basestring) – Use composite operator when applying frame. Only used if called with ImageMagick 7+.

New in version 0.3.0.

Changed in version 0.5.0: Added `compose` paramater, and ImageMagick 7 support.

property border_color
(*wand.color.Color*) The image border color. Used for special effects like *polaroid()*.

New in version 0.5.4.

brightness_contrast(*brightness=0.0*, *contrast=0.0*, *channel=None*)
Converts `brightness` & `contrast` parameters into a slope & intercept, and applies a polynomial function.

Parameters

- **brightness** (numbers.Real) – between `-100.0` and `100.0`. Default is `0.0` for unchanged.
- **contrast** (numbers.Real) – between `-100.0` and `100.0`. Default is `0.0` for unchanged.
- **channel** – Isolate a single color channel to apply contrast. See *CHANNELS*.

New in version 0.5.4.

Changed in version 0.5.5: Optional `channel` argument added.

canny(*radius=0.0*, *sigma=1.0*, *lower_percent=0.1*, *upper_percent=0.3*)
Detect edges by leveraging a multi-stage Canny algorithm.

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters

- **radius** (numbers.Real) – Size of gaussian filter.
- **sigma** (numbers.Real) – Standard deviation of gaussian filter.
- **lower_percent** (numbers.Real) – Normalized lower threshold. Values between 0.0 (0%) and 1.0 (100%). The default value is 0.1 or 10%.
- **upper_percent** (numbers.Real) – Normalized upper threshold. Values between 0.0 (0%) and 1.0 (100%). The default value is 0.3 or 30%.

Raises *WandLibraryVersionError* – if function is not available on system's library.

New in version 0.5.5.

caption(*text*, *left=0*, *top=0*, *width=None*, *height=None*, *font=None*, *gravity=None*)
Writes a caption `text` into the position.

Parameters

- **text** (basestring) – text to write
- **left** (numbers.Integral) – x offset in pixels
- **top** (numbers.Integral) – y offset in pixels
- **width** (numbers.Integral) – width of caption in pixels. default is *width* of the image
- **height** (numbers.Integral) – height of caption in pixels. default is *height* of the image
- **font** (*wand.font.Font*) – font to use. default is *font* of the image
- **gravity** (basestring) – text placement gravity. uses the current *gravity* setting of the image by default

New in version 0.3.0.

cdl(*ccc*)
Alias for *color_decision_list()*.

New in version 0.5.7.

charcoal(*radius*, *sigma*)
Transform an image into a simulated charcoal drawing.

See Example of *Charcoal*.

Parameters

- **radius** (numbers.Real) – The size of the Gaussian operator.
- **sigma** (numbers.Real) – The standard deviation of the Gaussian.

New in version 0.5.3.

chop(*width=None*, *height=None*, *x=None*, *y=None*, *gravity=None*)
Removes a region of an image, and reduces the image size accordingly.

Parameters

- **width** (numbers.Integral) – Size of region.
- **height** (numbers.Integral) – Size of region.

- **x** (numbers.Integral) – Offset on the X-axis.
- **y** (numbers.Integral) – Offset on the Y-axis.
- **gravity** (basestring) – Helper to auto-calculate offset. See *GRAVITY_TYPES*.

New in version 0.5.5.

Changed in version 0.6.8: Added `gravity` argument.

clahe(*width*, *height*, *number_bins*, *clip_limit*)
Contrast limited adaptive histogram equalization.

> **Warning:** The CLAHE method is only available with ImageMagick-7.

Parameters

- **width** (numbers.Integral) – Tile division width.
- **height** (numbers.Integral) – Tile division height.
- **number_bins** (numbers.Real) – Histogram bins.
- **clip_limit** (numbers.Real) – contrast limit.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

clamp(*channel=None*)
Restrict color values between 0 and quantum range. This is useful when applying arithmetic operations that could result in color values over/under-flowing.

Parameters **channel** (basestring) – Optional color channel.

New in version 0.5.0.

Changed in version 0.5.5: Added `channel` argument.

clone()
Clones the image. It is equivalent to call *Image* with `image` parameter.

```
with img.clone() as cloned:
    # manipulate the cloned image
    pass
```

Returns the cloned new image

Return type *Image*

New in version 0.1.1.

clut(*image*, *method='undefined'*, *channel=None*)
Replace color values by referencing another image as a Color Look Up Table.

Parameters

- **image** (*wand.image.BaseImage*) – Color Look Up Table image.
- **method** (basestring) – Pixel Interpolate method. Only available with ImageMagick-7. See *PIXEL_INTERPOLATE_METHODS*

- **channel** (basestring) – Optional color channel to target. See *CHANNELS*

New in version 0.5.0.

Changed in version 0.5.5: Added optional `channel` argument.

`coalesce`()

Rebuilds image sequence with each frame size the same as first frame, and composites each frame atop of previous.

Note: Only affects GIF, and other formats with multiple pages/layers.

New in version 0.5.0.

`color_decision_list`(*ccc*)

Applies color correction from a Color Correction Collection (CCC) xml string. An example of xml:

```
<ColorCorrectionCollection xmlns="urn:ASC:CDL:v1.2">
    <ColorCorrection id="cc03345">
        <SOPNode>
            <Slope> 0.9 1.2 0.5 </Slope>
            <Offset> 0.4 -0.5 0.6 </Offset>
            <Power> 1.0 0.8 1.5 </Power>
        </SOPNode>
        <SATNode>
            <Saturation> 0.85 </Saturation>
        </SATNode>
    </ColorCorrection>
</ColorCorrectionCollection>
```

Parameters **ccc** (basestring) – A XML string of the CCC contents.

New in version 0.5.7.

`color_map`(*index*, *color=None*)

Get & Set a color at a palette index. If `color` is given, the color at the index location will be set & returned. Omitting the `color` argument will only return the color value at index.

Valid indexes are between `0` and total *colors* of the image.

Note: Ensure the image type is set to `'palette'` before calling the *color_map()* method. For example:

```
with Image(filename='graph.png') as img:
    img.type = 'palette'
    palette = [img.color_map(idx) for idx in range(img.colors)]
    # ...
```

Parameters

- **index** (numbers.Integral) – The color postion of the image palette.
- **color** (*wand.color.Color*) – Optional color to _set_ at the given index.

Returns Color at index.

Return type *wand.color.Color*

New in version 0.5.3.

color_matrix(*matrix*)

Adjust color values by applying a matrix transform per pixel.

Matrix should be given as 2D list, with a max size of 6x6.

An example of 3x3 matrix:

```
matrix = [
    [1.0, 0.0, 0.0],
    [0.0, 1.0, 0.0],
    [0.0, 0.0, 1.0],
]
```

Which would translate RGB color channels by calculating the following:

$$red' = 1.0 * red + 0.0 * green + 0.0 * blue$$
$$green' = 0.0 * red + 1.0 * green + 0.0 * blue$$
$$blue' = 0.0 * red + 0.0 * green + 1.0 * blue$$

For RGB colorspace images, the rows & columns are laid out as:

	Red	Green	Blue	n/a	Alpha	Offset
Red'	1	0	0	0	0	0
Green'	0	1	0	0	0	0
Blue'	0	0	1	0	0	0
n/a	0	0	0	0	0	0
Alpha'	0	0	0	0	0	0
Offset'	0	0	0	0	0	0

Or for a CMYK colorspace image:

	Cyan	Yellow	Magenta	Black	Alpha	Offset
Cyan'	1	0	0	0	0	0
Yellow'	0	1	0	0	0	0
Magenta'	0	0	1	0	0	0
Black'	0	0	0	1	0	0
Alpha'	0	0	0	0	0	0
Offset'	0	0	0	0	0	0

See color-matrix for examples.

See Example of *Color Matrix*.

Parameters **matrix** (collections.abc.Sequence) – 2D List of doubles.

New in version 0.5.3.

color_threshold(*start=None*, *stop=None*)

Forces all pixels in color range to white, and all other pixels to black.

Note: This method is only works with ImageMagick-7.0.10, or later.

Parameters

- **start** (*wand.color.Color*) – Color to begin color range.
- **stop** (*wand.color.Color*) – Color to end color range.

New in version 0.6.4.

colorize(*color=None*, *alpha=None*)
Blends a given fill color over the image. The amount of blend is determined by the color channels given by the `alpha` argument.

See Example of *Colorize*.

Parameters

- **color** (*wand.color.Color*) – Color to paint image with.
- **alpha** (*wand.color.Color*) – Defines how to blend color.

New in version 0.5.3.

property colors
(`numbers.Integral`) Count of unique colors used within the image. This is READ ONLY property.

New in version 0.5.3.

property colorspace
(`basestring`) The image colorspace.

Defines image colorspace as in *COLORSPACE_TYPES* enumeration.

It may raise `ValueError` when the colorspace is unknown.

New in version 0.3.4.

combine(*channel='rgb_channels'*, *colorspace='rgb'*)
Creates an image where each color channel is assigned by a grayscale image in a sequence.

Warning: If your using ImageMagick-6, use `channel` argument to control the color-channel order. With ImageMagick-7, the `channel` argument has been replaced with `colorspace`.

For example:

```
for wand.image import Image

with Image() as img:
    img.read(filename='red_channel.png')
    img.read(filename='green_channel.png')
    img.read(filename='blue_channel.png')
    img.combine(colorspace='rgb')
    img.save(filename='output.png')
```

Parameters

- **channel** (`basestring`) – Determines the colorchannel ordering of the sequence. Only used for ImageMagick-6. See *CHANNELS*.
- **colorspace** (`basestring`) – Determines the colorchannel ordering of the sequence. Only used for ImageMagick-7. See *COLORSPACE_TYPES*.

New in version 0.5.9.

compare(*image*, *metric='undefined'*, *highlight=None*, *lowlight=None*)

Compares an image with another, and returns a reconstructed image & computed distortion. The reconstructed image will show the differences colored with `highlight`, and similarities with `lowlight`.

If you need the computed distortion between to images without a image being reconstructed, use *get_image_distortion()* method.

Set *fuzz* property to adjust pixel-compare thresholds.

For example:

```
from wand.image import Image

with Image(filename='input.jpg') as base:
    with Image(filename='subject.jpg') as img:
        base.fuzz = base.quantum_range * 0.20  # Threshold of 20%
        result_image, result_metric = base.compare(img)
        with result_image:
            result_image.save(filename='diff.jpg')
```

Parameters

- **image** (*wand.image.Image*) – The reference image
- **metric** (`basestring`) – The metric type to use for comparing. See *COMPARE_METRICS*
- **highlight** (*Color* or `basestring`) – Set the color of the delta pixels in the resulting difference image.
- **lowlight** (*Color* or `basestring`) – Set the color of the similar pixels in the resulting difference image.

Returns The difference image(*wand.image.Image*), the computed distortion between the images (`numbers.Real`)

Return type `tuple` (*Image*, `numbers.Real`)

New in version 0.4.3.

Changed in version 0.5.3: Added support for `highlight` & `lowlight`.

complex(*operator='undefined'*, *snr=None*)

Performs complex mathematics against two images in a sequence, and generates a new image with two results.

See also:

forward_fourier_transform() & *inverse_fourier_transform()*

```
from wand.image import Image

with Image(filename='real_part.png') as imgA:
    with Image(filename='imaginary_part.png') as imgB:
        imgA.sequence.append(imgB)
    with imgA.complex('conjugate') as results:
        results.save(filename='output-%02d.png')
```

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters

- **operator** (basestring) – Define which mathematic operator to perform. See COMPLEX_OPERATORS.
- **snr** (basestring) – Optional SNR parameter for 'divide' operator.

Raises *WandLibraryVersionError* – If ImageMagick library does not support this function.

New in version 0.5.5.

property compose

(basestring) The type of image compose. It's a string from *COMPOSITE_OPERATORS* list. It also can be set.

New in version 0.5.1.

composite(*image*, *left=None*, *top=None*, *operator='over'*, *arguments=None*, *gravity=None*)

Places the supplied image over the current image, with the top left corner of image at coordinates left, top of the current image. The dimensions of the current image are not changed.

Parameters

- **image** (*wand.image.Image*) – the image placed over the current image
- **left** (numbers.Integral) – the x-coordinate where *image* will be placed
- **top** (numbers.Integral) – the y-coordinate where *image* will be placed
- **operator** (basestring) – the operator that affects how the composite is applied to the image. available values can be found in the *COMPOSITE_OPERATORS* list. Default is 'over'.
- **arguments** (basestring) – Additional numbers given as a geometry string, or comma delimited values. This is needed for 'blend', 'displace', 'dissolve', and 'modulate' operators.
- **gravity** – Calculate the top & left values based on gravity value from *GRAVITY_TYPES*.

Type gravity: basestring

New in version 0.2.0.

Changed in version 0.5.3: The operator can be set, as well as additional composite arguments.

Changed in version 0.5.3: Optional gravity argument was added.

composite_channel(*channel*, *image*, *operator*, *left=None*, *top=None*, *arguments=None*, *gravity=None*)

Composite two images using the particular channel.

Parameters

- **channel** – the channel type. available values can be found in the *CHANNELS* mapping
- **image** (*Image*) – the composited source image. (the receiver image becomes the destination)
- **operator** (basestring) – the operator that affects how the composite is applied to the image. available values can be found in the *COMPOSITE_OPERATORS* list
- **left** (numbers.Integral) – the column offset of the composited source image
- **top** (numbers.Integral) – the row offset of the composited source image
- **arguments** (basestring) – Additional numbers given as a geometry string, or comma delimited values. This is needed for 'blend', 'displace', 'dissolve', and 'modulate' operators.

- **gravity** – Calculate the `top` & `left` values based on gravity value from *GRAVITY_TYPES*.

Type gravity: `basestring`

Raises `ValueError` – when the given `channel` or `operator` is invalid

New in version 0.3.0.

Changed in version 0.5.3: Support for optional composite arguments has been added.

Changed in version 0.5.3: Optional `gravity` argument was added.

property compression

(`basestring`) The type of image compression. It's a string from *COMPRESSION_TYPES* list. It also can be set.

New in version 0.3.6.

Changed in version 0.5.2: Setting *compression* now sets both *image_info* and *images* in the internal image stack.

property compression_quality

(`numbers.Integral`) Compression quality of this image.

New in version 0.2.0.

Changed in version 0.5.2: Setting *compression_quality* now sets both *image_info* and *images* in the internal image stack.

concat(*stacked=False*)

Concatenates images in stack into a single image. Left-to-right by default, top-to-bottom if `stacked` is True.

Parameters **stacked** (`bool`) – stack images in a column, or in a row (default)

New in version 0.5.0.

connected_components(***kwargs*)

Evaluates binary image, and groups connected pixels into objects. This method will also return a list of *ConnectedComponentObject* instances that will describe an object's features.

```
from wand.image import Image

with Image(filename='objects.gif') as img:
    objects = img.connected_components()
for cc_obj in objects:
    print("{0._id}: {0.size} {0.offset}".format(cc_obj))

#=> 0: (256, 171) (0, 0)
#=> 2: (120, 135) (104, 18)
#=> 3: (50, 36) (129, 44)
#=> 4: (21, 23) (0, 45)
#=> 1: (4, 10) (252, 0)
```

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

Tip: Set *fuzz* property to increase pixel matching by reducing tolerance of color-value comparisons:

```
from wand.image import Image
from wand.version import QUANTUM_RANGE

with Image(filename='objects.gif') as img:
    img.fuzz = 0.1 * QUANTUM_RANGE  # 10%
    objects = img.connected_components()
```

Parameters

- **angle_threshold** (basestring) – Optional argument that merges any region with an equivalent ellipse smaller than a given value. Requires ImageMagick-7.0.9-24, or greater.
- **area_threshold** (basestring) – Optional argument to merge objects under an area size.
- **background_id** (basestring) – Optional argument to identify which object should be the background. Requires ImageMagick-7.0.9-24, or greater.
- **circularity_threshold** (basestring) – Optional argument that merges any region smaller than value defined as: `4*pi*area/perimeter^2`. Requires ImageMagick-7.0.9-24, or greater.
- **connectivity** (numbers.Integral) – Either `4`, or `8`. A value of `4` will evaluate each pixels top-bottom, & left-right neighbors. A value of `8` will use the same pixels as with `4`, but will also include the four corners of each pixel. Default value of `4`.
- **diameter_threshold** (basestring) – Optional argument to merge any region under a given value. A region is defined as: `sqr(4*area/pi)`. Requires ImageMagick-7.0.9-24.
- **eccentricity_threshold** – Optional argument to merge any region with ellipse eccentricity under a given value. Requires ImageMagick-7.0.9-24, or greater.
- **keep** (basestring) – Comma separated list of object IDs to isolate, the reset are converted to transparent.
- **keep_colors** (basestring) – Semicolon separated list of objects to keep by their color value. Requires ImageMagick-7.0.9-24, or greater.
- **keep_top** (basestring) – Keeps only the top number of objects by area. Requires ImageMagick-7.0.9-24, or greater.
- **major_axis_threshold** (basestring) – Optional argument to merge any ellipse with a major axis smaller then given value. Requires ImageMagick-7.0.9-24, or greater.
- **mean_color** (bool) – Optional argument. Replace object color with mean color of the source image.
- **minor_axis_threshold** (basestring) – Optional argument to merge any ellipse with a minor axis smaller then given value. Requires ImageMagick-7.0.9-24, or greater.
- **perimeter_threshold** – Optional argument to merge any region with a perimeter smaller than the given value. Requires ImageMagick-7.0.9-24, or greater.
- **remove** (basestring) – Comma separated list of object IDs to ignore, and convert to transparent.
- **remove_colors** (basestring) – Semicolon separated list of objects to remove by there color. Requires ImageMagick-7.0.9-24, or greater.

Returns A list of *ConnectedComponentObject*.

Return type list [*ConnectedComponentObject*]

Raises *WandLibraryVersionError* – If ImageMagick library does not support this method.

New in version 0.5.5.

Changed in version 0.5.6: Added `mean_color`, `keep`, & `remove` optional arguments.

Changed in version 0.6.4: Added `angle_threshold`, `circularity_threshold`, `diameter_threshold`, `eccentricity_threshold`, `keep_colors`, `major_axis_threshold`, `minor_axis_threshold`, `perimeter_threshold`, and `remove_colors` optional arguments.

contrast(*sharpen=True*)

Enhances the difference between lighter & darker values of the image. Set `sharpen` to `False` to reduce contrast.

Parameters **sharpen** (`bool`) – Increase, or decrease, contrast. Default is `True` for increased contrast.

New in version 0.5.7.

contrast_stretch(*black_point=0.0*, *white_point=None*, *channel=None*)

Enhance contrast of image by adjusting the span of the available colors.

Parameters

- **black_point** (`numbers.Real`) – black point between 0.0 and 1.0. default is 0.0
- **white_point** (`numbers.Real`) – white point between 0.0 and 1.0. Defaults to the same value given to the `black_point` argument.
- **channel** (*CHANNELS*) – optional color channel to apply contrast stretch

Raises `ValueError` – if `channel` is not in *CHANNELS*

New in version 0.4.1.

Changed in version 0.5.5: The `white_point` argument will now default to the value given by the `black_point` argument.

convex_hull(*background=None*)

Find the smallest convex polygon, and returns a list of points.

Note: Requires ImageMagick-7.0.10 or greater.

You can pass the list of points directly to *Drawing.polygon()* method to draw the convex hull shape on the image.

```
from wand.image import Image
from wand.drawing import Drawing

with Image(filename='kdf_black.png') as img:
  points = img.convex_hull()
  with Drawing() as ctx:
    ctx.fill_color = 'transparent'
    ctx.stroke_color = 'red'
    ctx.polygon(points=points)
    ctx(img)
img.save(filename='kdf_black_convex_hull.png')
```

Parameters **background** (basestring or *Color*) – Define which color value to evaluate as the background.

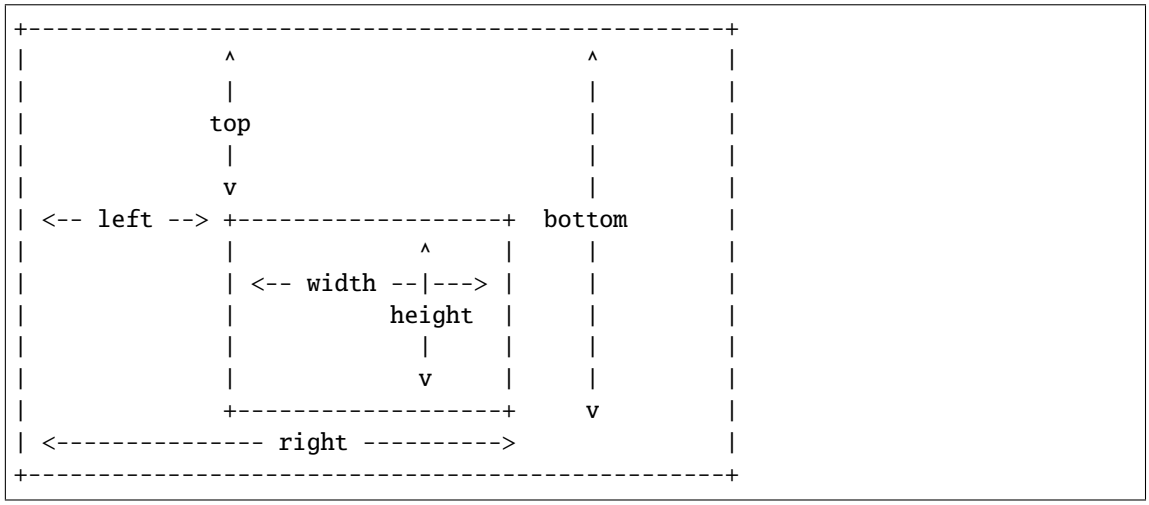
Returns list of points

Return type list [tuple (float, float)]

New in version 0.6.4.

crop(*left=0*, *top=0*, *right=None*, *bottom=None*, *width=None*, *height=None*, *reset_coords=True*, *gravity=None*)

Crops the image in-place.



Parameters

- **left** (numbers.Integral) – x-offset of the cropped image. default is 0
- **top** (numbers.Integral) – y-offset of the cropped image. default is 0
- **right** (numbers.Integral) – second x-offset of the cropped image. default is the *width* of the image. this parameter and **width** parameter are exclusive each other
- **bottom** (numbers.Integral) – second y-offset of the cropped image. default is the *height* of the image. this parameter and **height** parameter are exclusive each other

- **width** (numbers.Integral) – the *width* of the cropped image. default is the *width* of the image. this parameter and `right` parameter are exclusive each other
- **height** (numbers.Integral) – the *height* of the cropped image. default is the *height* of the image. this parameter and `bottom` parameter are exclusive each other
- **reset_coords** (bool) – optional flag. If set, after the rotation, the coordinate frame will be relocated to the upper-left corner of the new image. By default is *True*.
- **gravity** (*GRAVITY_TYPES*) – optional flag. If set, will calculate the `top` and `left` attributes. This requires both *width* and *height* parameters to be included.

Raises ValueError – when one or more arguments are invalid

Note: If you want to crop the image but not in-place, use slicing operator.

Changed in version 0.4.1: Added `gravity` option. Using `gravity` along with `width` & `height` to auto-adjust `left` & `top` attributes.

Changed in version 0.1.8: Made to raise `ValueError` instead of `IndexError` for invalid `width`/`height` arguments.

New in version 0.1.7.

cycle_color_map(*offset=1*)
Shift the image color-map by a given offset.

Parameters **offset** (numbers.Integral) – number of steps to rotate index by.

New in version 0.5.3.

decipher(*passphrase*)
Decrypt ciphered pixels into original values.

Note: *ImageError* will be thrown if the system's ImageMagick library was compiled without cipher support.

Parameters **passphrase** (basestring) – the secret passphrase to decrypt with.

New in version 0.6.3.

deconstruct()
Iterates over internal image stack, and adjust each frame size to minimum bounding region of any changes from the previous frame.

New in version 0.5.0.

property delay
(numbers.Integral) The number of ticks between frames.

New in version 0.5.9.

property depth
(numbers.Integral) The depth of this image.

New in version 0.2.1.

deskew(*threshold*)
Attempts to remove skew artifacts common with most scanning & optical import devices.

Params threshold limit between foreground & background. Use a real number between *0.0* & *1.0* to match CLI's percent argument.

New in version 0.5.0.

despeckle()
Applies filter to reduce noise in image.

See Example of *Despeckle*.

New in version 0.5.0.

dirty = None
(bool) Whether the image is changed or not.

property dispose
(basestring) Controls how the image data is handled during animations. Values are from *DISPOSE_TYPES* list, and can also be set.

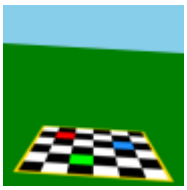
See also:

Dispose Images section in Animation Basics article.

New in version 0.5.0.

distort(*method*, *arguments*, *best_fit=False*)
Distorts an image using various distorting methods.

```
from wand.image import Image
from wand.color import Color

with Image(filename='checks.png') as img:
    img.virtual_pixel = 'background'
    img.background_color = Color('green')
    img.matte_color = Color('skyblue')
    arguments = (0, 0, 20, 60,
                 90, 0, 70, 63,
                 0, 90, 5, 83,
                 90, 90, 85, 88)
    img.distort('perspective', arguments)
    img.save(filename='checks_perspective.png')
```



Use *virtual_pixel*, *background_color*, and *matte_color* properties to control the behavior of pixels rendered outside of the image boundaries.

Use *interpolate_method* to control how images scale-up.

Distortion viewport, and scale, can be defined by using *Image.artifacts* dictionary. For example:

```
img.artifacts['distort:viewport'] = '44x44+15+0'
img.artifacts['distort:scale'] = '10'
```

See Additional examples of *Distortion*.

Parameters

- **method** (basestring) – Distortion method name from *DISTORTION_METHODS*
- **arguments** (collections.abc.Sequence) – List of distorting float arguments unique to distortion method
- **best_fit** (bool) – Attempt to resize resulting image fit distortion. Defaults False

New in version 0.4.1.

edge(*radius=0.0*)
Applies convolution filter to detect edges.

See Example of *Edge*.

Parameters **radius** (numbers.Real) – aperture of detection filter.

New in version 0.5.0.

emboss(*radius=0.0*, *sigma=0.0*)
Applies convolution filter against Gaussians filter.

Note: The *radius* value should be larger than *sigma* for best results.

See Example of *Emboss*.

Parameters

- **radius** (numbers.Real) – filter aperture size.
- **sigma** (numbers.Real) – standard deviation.

New in version 0.5.0.

encipher(*passphrase*)
Encrypt plain pixels into ciphered values.

Note: *ImageError* will be thrown if the system's ImageMagick library was compiled without cipher support.

Parameters **passphrase** (basestring) – the secret passphrase to encrypt with.

New in version 0.6.3.

Changed in version 0.6.8: Fixed C-API call.

enhance()
Applies digital filter to reduce noise.

See Example of *Enhance*.

New in version 0.5.0.

equalize(*channel=None*)
Equalizes the image histogram

Parameters **channel** (basestring) – Optional channel. See *CHANNELS*.

New in version 0.3.10.

Changed in version 0.5.5: Added optional `channel` argument.

evaluate(*operator=None*, *value=0.0*, *channel=None*)
Apply arithmetic, relational, or logical expression to an image.

Percent values must be calculated against the quantum range of the image:

```
fifty_percent = img.quantum_range * 0.5
img.evaluate(operator='set', value=fifty_percent)
```

See Example of *Evaluate Expression*.

Parameters

- **operator** (*EVALUATE_OPS*) – Type of operation to calculate
- **value** (numbers.Real) – Number to calculate with `operator`
- **channel** (*CHANNELS*) – Optional channel to apply operation on.

Raises

- ***TypeError*** – When `value` is not numeric.
- **ValueError** – When `operator`, or `channel` are not defined in constants.

New in version 0.4.1.

export_pixels(*x=0*, *y=0*, *width=None*, *height=None*, *channel_map='RGBA'*, *storage='char'*)
Export pixel data from a raster image to a list of values.

The `channel_map` tells ImageMagick which color channels to export, and what order they should be written as – per pixel. Valid entries for `channel_map` are:

- `'R'` - Red channel
- `'G'` - Green channel
- `'B'` - Blue channel
- `'A'` - Alpha channel (`0` is transparent)
- `'O'` - Alpha channel (`0` is opaque)
- `'C'` - Cyan channel
- `'Y'` - Yellow channel
- `'M'` - Magenta channel
- `'K'` - Black channel
- `'I'` - Intensity channel (only for grayscale)
- `'P'` - Padding

See *STORAGE_TYPES* for a list of valid `storage` options. This tells ImageMagick what type of data it should calculate & write to. For example; a storage type of `'char'` will write a 8-bit value between 0 ~ 255, a storage type of `'short'` will write a 16-bit value between 0 ~ 65535, and a `'integer'` will write a 32-bit value between 0 ~ 4294967295.

Note: By default, the entire image will be exported as `'char'` storage with each pixel mapping Red, Green, Blue, & Alpha channels.

Parameters

- **x** (numbers.Integral) – horizontal starting coordinate of raster.
- **y** (numbers.Integral) – vertical starting coordinate of raster.
- **width** (numbers.Integral) – horizontal length of raster.
- **height** (numbers.Integral) – vertical length of raster.
- **channel_map** (basestring) – a string listing the channel data format for each pixel.
- **storage** (basestring) – what data type each value should be calculated as.

Returns list of values.

Return type collections.abc.Sequence

New in version 0.5.0.

extent(*width=None*, *height=None*, *x=None*, *y=None*, *gravity=None*)

Adjust the canvas size of the image. Use `x` & `y` to offset the image's relative placement in the canvas, or `gravity` helper for quick position placement.

Parameters

- **width** (numbers.Integral) – the target width of the extended image. Default is the *width* of the image.
- **height** (numbers.Integral) – the target height of the extended image. Default is the *height* of the image.
- **x** (numbers.Integral) – the x-axis offset of the extended image. Default is 0, and can not be used with `gravity`.
- **y** (numbers.Integral) – the `y` offset of the extended image. Default is 0, and can not be used with `gravity`.
- **gravity** (basestring) – position of the item extent when not using `x` & `y`. See *GRAVITY_TYPES*.

New in version 0.4.5.

Changed in version 0.6.8: Added `gravity` argument.

features(*distance*)

Calculate directional image features for each color channel. Feature metrics including:

- angular second moment
- contrast
- correlation
- variance sum of squares
- inverse difference moment
- sum average
- sum variance

- sum entropy
- entropy
- difference variance
- difference entropy
- information measures of correlation 1
- information measures of correlation 2
- maximum correlation coefficient

With each metric containing horizontal, vertical, left & right diagonal values.

```
from wand.image import Image

with Image(filename='rose:') as img:
    channel_features = img.features(distance=32)
    for channels, features in channel_features.items():
        print(channels)
        for feature, directions in features.items():
            print('  ', feature)
            for name, value in directions.items():
                print('    ', name, value)
```

Parameters **distance** (numbers.Integral) – Define the distance if pixels to calculate.

Returns a dict mapping each color channel with a dict of each feature.

Return type dict

New in version 0.5.5.

fft(*magnitude=True*)
Alias for *forward_fourier_transform()*.

New in version 0.5.7.

flip()
Creates a vertical mirror image by reflecting the pixels around the central x-axis. It manipulates the image in place.

See Example of *Flip and flop*.

New in version 0.3.0.

flop()
Creates a horizontal mirror image by reflecting the pixels around the central y-axis. It manipulates the image in place.

See Example of *Flip and flop*.

New in version 0.3.0.

property font
(*wand.font.Font*) The current font options.

property font_antialias
Deprecated since version 0.5.0: Use *antialias* instead.

property font_path
(basestring) The path of the current font. It also can be set.

property font_size
(numbers.Real) The font size. It also can be set.

property format
(basestring) The image format.

If you want to convert the image format, just reset this property:

```
assert isinstance(img, wand.image.Image)
img.format = 'png'
```

It may raise ValueError when the format is unsupported.

See also:

ImageMagick Image Formats ImageMagick uses an ASCII string known as *magick* (e.g. GIF) to identify file formats, algorithms acting as formats, built-in patterns, and embedded profile types.

New in version 0.1.6.

forward_fourier_transform(*magnitude=True*)
Performs a discrete Fourier transform. The image stack is replaced with the results. Either a pair of magnitude & phase images, or real & imaginary (HDRI).

```
from wand.image import Image
from wand.version import QUANTUM_RANGE

with Image(filename='source.png') as img:
    img.forward_fourier_transform()
    img.depth = QUANTUM_RANGE
    img.save(filename='fft_%02d.png')
```

See also:

inverse_fourier_transform() & *complex()*

Note: ImageMagick must have HDRI support to compute real & imaginary components (i.e. magnitude=False).

Parameters **magnitude** (bool) – If True, generate magnitude & phase, else real & imaginary. Default True

New in version 0.5.5.

frame(*matte=None*, *width=1*, *height=1*, *inner_bevel=0*, *outer_bevel=0*, *compose='over'*)
Creates a bordered frame around image. Inner & outer bevel can simulate a 3D effect.

Parameters

- **matte** (*wand.color.Color*) – color of the frame
- **width** (numbers.Integral) – total size of frame on x-axis
- **height** (numbers.Integral) – total size of frame on y-axis
- **inner_bevel** (numbers.Real) – inset shadow length
- **outer_bevel** (numbers.Real) – outset highlight length

- **compose** (basestring) – Optional composite operator. Default `'over'`, and only available with ImageMagick-7.

New in version 0.4.1.

Changed in version 0.5.6: Added optional `compose` parameter.

function(*function*, *arguments*, *channel=None*)

Apply an arithmetic, relational, or logical expression to an image.

Defaults entire image, but can isolate affects to single color channel by passing *CHANNELS* value to `channel` parameter.

Note: Support for function methods added in the following versions of ImageMagick.

- `'polynomial'` >= 6.4.8-8
- `'sinusoid'` >= 6.4.8-8
- `'arcsin'` >= 6.5.3-1
- `'arctan'` >= 6.5.3-1

See Example of *Function Expression*.

Parameters

- **function** (basestring) – a string listed in *FUNCTION_TYPES*
- **arguments** (collections.abc.Sequence) – a sequence of doubles to apply against `function`
- **channel** (basestring) – optional *CHANNELS*, defaults all

Raises

- **ValueError** – when a `function`, or `channel` is not defined in there respected constant
- ***TypeError*** – if `arguments` is not a sequence

New in version 0.4.1.

property fuzz

(numbers.Real) The normalized real number between `0.0` and *quantum_range*. This property influences the accuracy of *compare()*.

New in version 0.5.3.

fx(*expression*, *channel=None*)

Manipulate each pixel of an image by given expression.

FX will preserver current wand instance, and return a new instance of *Image* containing affected pixels.

Defaults entire image, but can isolate affects to single color channel by passing *CHANNELS* value to `channel` parameter.

See also:

The anatomy of FX expressions can be found at http://www.imagemagick.org/script/fx.php

See Example of *FX*.

Parameters

- **expression** (basestring) – The entire FX expression to apply
- **channel** (*CHANNELS*) – Optional channel to target.

Returns A new instance of an image with expression applied

Return type *Image*

New in version 0.4.1.

Changed in version 0.6.9: Will raise `WandRuntimeError` if method is unable to generate a new image & doesn't throw an exception.

gamma(*adjustment_value=1.0*, *channel=None*)
Gamma correct image.

Specific color channels can be correct individual. Typical values range between 0.8 and 2.3.

See Example of *Gamma*.

Parameters

- **adjustment_value** (numbers.Real) – value to adjust gamma level. Default *1.0*
- **channel** (basestring) – optional channel to apply gamma correction

Raises

- ***TypeError*** – if `gamma_point` is not a `numbers.Real`
- **ValueError** – if `channel` is not in *CHANNELS*

New in version 0.4.1.

gaussian_blur(*radius=0.0*, *sigma=0.0*, *channel=None*)
Blurs the image. We convolve the image with a gaussian operator of the given `radius` and standard deviation (`sigma`). For reasonable results, the `radius` should be larger than `sigma`. Use a `radius` of 0 and *blur()* selects a suitable `radius` for you.

See Example of *Gaussian Blur*.

Parameters

- **radius** (numbers.Real) – the radius of the, in pixels, not counting the center pixel
- **sigma** (numbers.Real) – the standard deviation of the, in pixels
- **channel** (basestring) – Optional color channel to target. See *CHANNELS*

New in version 0.3.3.

Changed in version 0.5.5: Added `channel` argument.

Changed in version 0.5.7: Positional arguments `radius` & `sigma` have been converted to keyword arguments.

get_image_distortion(*image*, *metric='undefined'*)
Compares two images, and return the specified distortion metric.

This method is faster than *compare()* method as ImageMagick will not need to reconstruct an image.

Parameters

- **image** (*wand.image.BaseImage*) – Image to reference.
- **metric** (basestring) – Compare disortion metric to use. See *COMPARE_METRICS*.

Returns Computed value of the distortion metric used.

Return type numbers.Real

New in version 0.6.6.

property gravity
(basestring) The text placement gravity used when annotating with text. It's a string from *GRAVITY_TYPES* list. It also can be set.

property green_primary
(tuple) The chromatic green primary point for the image. With ImageMagick-6 the primary value is `(x, y)` coordinates; however, ImageMagick-7 has `(x, y, z)`.

New in version 0.5.2.

hald_clut(*image*, *channel=None*)
Replace color values by referencing a Higher And Lower Dimension (HALD) Color Look Up Table (CLUT). You can generate a HALD image by using ImageMagick's *hald:* protocol.

```
with Image(filename='rose:') as img:
    with Image(filename='hald:3') as hald:
        hald.gamma(1.367)
        img.hald_clut(hald)
```

Parameters

- **image** (*wand.image.BaseImage*) – The HALD color matrix.
- **channel** (basestring) – Optional color channel to target. See *CHANNELS*

New in version 0.5.0.

Changed in version 0.5.5: Added `channel` argument.

property height
(numbers.Integral) The height of this image.

property histogram
(*HistogramDict*) The mapping that represents the histogram. Keys are *Color* objects, and values are the number of pixels.

Tip: True-color photos can have millions of color values. If performance is more valuable than accuracy, remember to *quantize()* the image before generating a *HistogramDict*.

with Image(filename='hd_photo.jpg') as img: img.quantize(255, 'RGB', 0, False, False) hist = img.histogram

---

New in version 0.3.0.

hough_lines(*width*, *height=None*, *threshold=40*)
Identify lines within an image. Use *canny()* to reduce image to a binary edge before calling this method.

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters

- **width** (numbers.Integral) – Local maxima of neighboring pixels.
- **height** (numbers.Integral) – Local maxima of neighboring pixels.

- **threshold** (numbers.Integral) – Line count to limit. Default to 40.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

ift(*phase*, *magnitude=True*)
Alias for *inverse_fourier_transform()*.

New in version 0.5.7.

implode(*amount=0.0*, *method='undefined'*)
Creates a "imploding" effect by pulling pixels towards the center of the image.

See Example of *Implode*.

Parameters

- **amount** (numbers.Real) – Normalized degree of effect between *0.0* & *1.0*.
- **method** (basestring) – Which interpolate method to apply to effected pixels. See *PIXEL_INTERPOLATE_METHODS* for a list of options. Only available with ImageMagick-7.

New in version 0.5.2.

import_pixels(*x=0*, *y=0*, *width=None*, *height=None*, *channel_map='RGB'*, *storage='char'*, *data=None*)
Import pixel data from a byte-string to the image. The instance of *Image* must already be allocated with the correct size.

The `channel_map` tells ImageMagick which color channels to export, and what order they should be written as – per pixel. Valid entries for `channel_map` are:

- `'R'` - Red channel
- `'G'` - Green channel
- `'B'` - Blue channel
- `'A'` - Alpha channel (`0` is transparent)
- `'O'` - Alpha channel (`0` is opaque)
- `'C'` - Cyan channel
- `'Y'` - Yellow channel
- `'M'` - Magenta channel
- `'K'` - Black channel
- `'I'` - Intensity channel (only for grayscale)
- `'P'` - Padding

See *STORAGE_TYPES* for a list of valid `storage` options. This tells ImageMagick what type of data it should calculate & write to. For example; a storage type of `'char'` will write a 8-bit value between 0 ~ 255, a storage type of `'short'` will write a 16-bit value between 0 ~ 65535, and a `'integer'` will write a 32-bit value between 0 ~ 4294967295.

Note: By default, the entire image will be exported as `'char'` storage with each pixel mapping Red, Green, Blue, & Alpha channels.

Parameters

- **x** (numbers.Integral) – horizontal starting coordinate of raster.
- **y** (numbers.Integral) – vertical starting coordinate of raster.
- **width** (numbers.Integral) – horizontal length of raster.
- **height** (numbers.Integral) – vertical length of raster.
- **channel_map** (basestring) – a string listing the channel data format for each pixel.
- **storage** (basestring) – what data type each value should be calculated as.

New in version 0.5.0.

property interlace_scheme
(basestring) The interlace used by the image. See *INTERLACE_TYPES*.

New in version 0.5.2.

Changed in version 0.6.2: The *interlace_scheme* property now points to the image. Previously was pointing to the MagickWand.

property interpolate_method
(basestring) The interpolation method of the image. See *PIXEL_INTERPOLATE_METHODS*.

New in version 0.5.2.

inverse_fourier_transform(*phase*, *magnitude=True*)
Applies the inverse of a discrete Fourier transform. The image stack is replaced with the results. Either a pair of magnitude & phase images, or real & imaginary (HDRI).

```
from wand.image import Image

with Image(filename='magnitude.png') as img:
    with Image(filename='phase.png') as phase:
        img.inverse_fourier_transform(phase)
    img.save(filename='output.png')
```

See also:

forward_fourier_transform() & *complex()*

Note: ImageMagick must have HDRI support to compute real & imaginary components (i.e. magnitude=False).

Parameters

- **phase** (*BaseImage*) – Second part (image) of the transform. Either the phase, or the imaginary part.
- **magnitude** (bool) – If True, accept magnitude & phase input, else real & imaginary. Default True

New in version 0.5.5.

iterator_first()
Sets the internal image-stack iterator to the first image. Useful for prepending an image at the start of the stack.

New in version 0.6.2.

`iterator_get`()
Returns the position of the internal image-stack index.

Return type int

New in version 0.6.2.

`iterator_last`()
Sets the internal image-stack iterator to the last image. Useful for appending an image to the end of the stack.

New in version 0.6.2.

`iterator_length`()
Get the count of images in the image-stack.

Return type int

New in version 0.6.2.

`iterator_next`()
Steps the image-stack index forward by one

Return type bool

New in version 0.6.2.

`iterator_previous`()
Steps the image-stack index back by one.

Return type bool

New in version 0.6.2.

`iterator_reset`()
Reset internal image-stack iterator. Useful for iterating over the image-stack without allocating *Sequence*.

New in version 0.6.2.

`iterator_set`(*index*)
Sets the index of the internal image-stack.

Return type bool

New in version 0.6.2.

`kmeans`(*number_colors=None*, *max_iterations=100*, *tolerance=0.01*)
Reduces the number of colors in an image by appling the K-means clustering algorithm.

Note: Requires ImageMagick-7.0.10-37, or later.

Parameters

- **number_colors** (numbers.Integral) – the target number of colors to use as seeds.
- **max_iterations** (numbers.Integral) – maximum number of iterations needed until convergence. Default 100.
- **tolerance** (numbers.Real) – maximum tolerance between distrotion iterations. Default 0.01

New in version 0.6.4.

property kurtosis
(numbers.Real) The kurtosis of the image.

Tip: If you want both *kurtosis* & *skewness*, it would be faster to call *kurtosis_channel()* directly.

New in version 0.5.3.

kurtosis_channel(*channel='default_channels'*)
Calculates the kurtosis and skewness of the image.

```
from wand.image import Image

with Image(filename='input.jpg') as img:
    kurtosis, skewness = img.kurtosis_channel()
```

Parameters **channel** (basestring) – Select which color channel to evaluate. See *CHANNELS*. Default 'default_channels'.

Returns Tuple of *kurtosis* & *skewness* values.

Return type tuple

New in version 0.5.3.

kuwahara(*radius=1.0*, *sigma=None*)
Edge preserving noise reduction filter.

https://en.wikipedia.org/wiki/Kuwahara_filter

If sigma is not given, the value will be calculated as:

sigma = radius - 0.5

To match original algorithm's behavior, increase radius value by one:

myImage.kuwahara(myRadius + 1, mySigma)

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

See Example of *Kuwahara*.

Parameters

- **radius** (numbers.Real) – Size of the filter aperture.
- **sigma** (numbers.Real) – Standard deviation of Gaussian filter.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

label(*text*, *left=None*, *top=None*, *font=None*, *gravity=None*, *background_color='transparent'*)
Writes a label text into the position on top of the existing canvas. This method doesn't autofit text like *caption()*. Use left & top, or gravity, to position the text.

Parameters

- **text** (basestring) – text to write.

- **left** (numbers.Integral) – x offset in pixels.
- **top** (numbers.Integral) – y offset in pixels.
- **font** (*wand.font.Font*) – font to use. default is *font* of the image.
- **gravity** (basestring) – text placement gravity.

New in version 0.6.8.

property length_of_bytes
(numbers.Integral) The original size, in bytes, of the image read. This will return *0* if the image was modified in a way that would invalidate the original length value.

New in version 0.5.4.

level(*black=0.0*, *white=None*, *gamma=1.0*, *channel=None*)
Adjusts the levels of an image by scaling the colors falling between specified black and white points to the full available quantum range.

If only `black` is given, `white` will be adjusted inward.

See Example of *Level*.

Parameters

- **black** (numbers.Real) – Black point, as a percentage of the system's quantum range. Defaults to 0.
- **white** (numbers.Real) – White point, as a percentage of the system's quantum range. Defaults to 1.0.
- **gamma** (numbers.Real) – Optional gamma adjustment. Values > 1.0 lighten the image's midtones while values < 1.0 darken them.
- **channel** (*CHANNELS*) – The channel type. Available values can be found in the *CHANNELS* mapping. If `None`, normalize all channels.

Note: Images may not be affected if the `white` value is equal to or less than the `black` value.

New in version 0.4.1.

level_colors(*black_color*, *white_color*, *channel=None*)
Maps given colors to "black" & "white" values.

Warning: This class method is only available with ImageMagick 7.0.8-54, or greater.

Parameters

- **black_color** (Color) – linearly map given color as "black" point.
- **white_color** (Color) – linearly map given color as "white" point.
- **channel** (basestring) – target a specific color-channel to levelize.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.6.

levelize(*black=0.0*, *white=None*, *gamma=1.0*, *channel=None*)
Reverse of *level()*, this method compresses the range of colors between `black` & `white` values.

If only `black` is given, `white` will be adjusted inward.

> **Warning:** This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters

- **black** (numbers.Real) – Black point, as a percentage of the system's quantum range. Defaults to 0.
- **white** (numbers.Real) – White point, as a percentage of the system's quantum range. Defaults to 1.0.
- **gamma** (numbers.Real) – Optional gamma adjustment. Values > 1.0 lighten the image's midtones while values < 1.0 darken them.
- **channel** (*CHANNELS*) – The channel type. Available values can be found in the *CHANNELS* mapping. If `None`, normalize all channels.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

levelize_colors(*black_color*, *white_color*, *channel=None*)
Reverse of *level_colors()*, and creates a de-contrasting gradient of given colors. This works best with grayscale images.

> **Warning:** This class method is only available with ImageMagick 7.0.8-54, or greater.

Parameters

- **black_color** (Color) – tint map given color as "black" point.
- **white_color** (Color) – tint map given color as "white" point.
- **channel** (basestring) – target a specific color-channel to levelize.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.6.

linear_stretch(*black_point=0.0*, *white_point=1.0*)
Enhance saturation intensity of an image.

Parameters

- **black_point** (numbers.Real) – Black point between 0.0 and 1.0. Default 0.0
- **white_point** (numbers.Real) – White point between 0.0 and 1.0. Default 1.0

New in version 0.4.1.

liquid_rescale(*width*, *height*, *delta_x=0*, *rigidity=0*)
Rescales the image with seam carving, also known as image retargeting, content-aware resizing, or liquid rescaling.

Parameters

- **width** (numbers.Integral) – the width in the scaled image
- **height** (numbers.Integral) – the height in the scaled image
- **delta_x** (numbers.Real) – maximum seam transversal step. 0 means straight seams. default is 0
- **rigidity** (numbers.Real) – introduce a bias for non-straight seams. default is 0

Raises *wand.exceptions.MissingDelegateError* – when ImageMagick isn't configured `--with-lqr` option.

Note: This feature requires ImageMagick to be configured `--with-lqr` option. Or it will raise *MissingDelegateError*:

See also:

Seam carving — Wikipedia The article which explains what seam carving is on Wikipedia.

local_contrast(*radius=10*, *strength=12.5*)
Increase light-dark transitions within image.

Warning: This class method is only available with ImageMagick 6.9.3, or greater.

Parameters

- **radius** (numbers.Real) – The size of the Gaussian operator. Default value is `10.0`.
- **strength** (numbers.Real) – Percentage of blur mask to apply. Values can be between `0.0` and `100` with a default of `12.5`.

New in version 0.5.7.

property loop
(numbers.Integral) Number of frame iterations. A value of `0` will loop forever.

magnify()
Quickly double an image in size. This is a convenience method. Use *resize()*, *resample()*, or *sample()* for more control.

New in version 0.5.5.

property matte_color
(*wand.color.Color*) The color value of the matte channel. This can also be set.

New in version 0.4.1.

property maxima
(numbers.Real) The maximum quantum value within the image. Value between 0.0 and *quantum_range*

Tip: If you want both *maxima* & *minima*, it would be faster to call *range_channel()* directly.

New in version 0.5.3.

property mean
(numbers.Real) The mean of the image, and have a value between 0.0 and *quantum_range*

Tip: If you want both *mean* & *standard_deviation*, it would be faster to call *mean_channel()* directly.

New in version 0.5.3.

mean_channel(*channel='default_channels'*)
Calculates the mean and standard deviation of the image.

```
from wand.image import Image

with Image(filename='input.jpg') as img:
    mean, stddev = img.mean_channel()
```

Parameters **channel** (basestring) – Select which color channel to evaluate. See *CHANNELS*. Default 'default_channels'.

Returns Tuple of *mean* & *standard_deviation* values. The mean value will be between 0.0 & *quantum_range*

Return type tuple

New in version 0.5.3.

mean_shift(*width*, *height*, *color_distance=0.1*)
Recalculates pixel value by comparing neighboring pixels within a color distance, and replacing with a mean value. Works best with Gray, YCbCr, YIQ, or YUV colorspaces.

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters

- **width** (numbers.Integral) – Size of the neighborhood window in pixels.
- **height** (numbers.Integral) – Size of the neighborhood window in pixels.
- **color_distance** (numbers.Real) – Include pixel values within this color distance.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

merge_layers(*method*)
Composes all the image layers from the current given image onward to produce a single image of the merged layers.

The initial canvas's size depends on the given ImageLayerMethod, and is initialized using the first images background color. The images are then composited onto that image in sequence using the given composition that has been assigned to each individual image. The method must be set with a value from *IMAGE_LAYER_METHOD* that is acceptable to this operation. (See ImageMagick documentation for more details.)

Parameters **method** (basestring) – the method of selecting the size of the initial canvas.

New in version 0.4.3.

property minima

(numbers.Real) The minimum quantum value within the image. Value between 0.0 and *quantum_range*

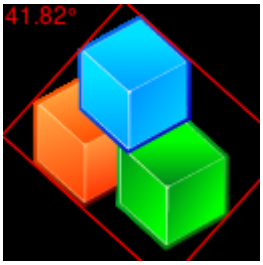
Tip: If you want both *maxima* & *minima*, it would be faster to call *range_channel()* directly.

New in version 0.5.3.

minimum_bounding_box(*orientation=None*)

Find the minmum bounding box within the image. Use properties *fuzz* & *background_color* to influence bounding box thresholds.

```
from wand.image import Image
from wand.drawing import Drawing

with Image(filename='kdf_black.png') as img:
    img.fuzz = img.quantum_range * 0.1
    img.background_color = 'black'
    mbr = img.minimum_bounding_box()
    with Drawing() as ctx:
        ctx.fill_color = 'transparent'
        ctx.stroke_color = 'red'
        ctx.polygon(points=mbr['points'])
        ctx.fill_color = 'red'
        ctx.stroke_color = 'transparent'
        ctx.text(1, 10, '{0:.4g}°'.format(mbr['angle']))
        ctx(img)
    img.save(filename='kdf_black_mbr.png')
```



Note: Requires ImageMagick-7.0.10 or later.

Parameters **orientation** (basestring) – sets the image orientation. Values can be 'landscape', or 'portrait'.

Returns a directory of MBR properties & corner points.

Return type dict { "points": list [tuple (float, float)], "area": float, "width": float, "height": float, "angle": float, "unrotate": float }

New in version 0.6.4.

mode(*width*, *height=None*)
Replace each pixel with the mathematical mode of the neighboring colors. This is an alias of the *statistic()* method.

Parameters

- **width** (numbers.Integral) – Number of neighboring pixels to include in mode.
- **height** (numbers.Integral) – Optional height of neighboring pixels, defaults to the same value as width.

New in version 0.5.4.

modulate(*brightness=100.0*, *saturation=100.0*, *hue=100.0*)
Changes the brightness, saturation and hue of an image. We modulate the image with the given brightness, saturation and hue.

Parameters

- **brightness** (numbers.Real) – percentage of brightness
- **saturation** (numbers.Real) – percentage of saturation
- **hue** (numbers.Real) – percentage of hue rotation

Raises ValueError – when one or more arguments are invalid

New in version 0.3.4.

morphology(*method=None*, *kernel=None*, *iterations=1*, *channel=None*)
Manipulate pixels based on the shape of neighboring pixels.

The method determines what type of effect to apply to matching kernel shapes. Common methods can be add/remove, or lighten/darken pixel values.

The kernel describes the shape of the matching neighbors. Common shapes are provided as "built-in" kernels. See :const`KERNEL_INFO_TYPES` for examples. The format for built-in kernels is:

```
label:geometry
```

Where *label* is the kernel name defined in *KERNEL_INFO_TYPES*, and *:geometry* is an optional geometry size. For example:

```
with Image(filename='rose:') as img:
    img.morphology(method='dilate', kernel='octagon:3x3')
    # or simply
    img.morphology(method='edgein', kernel='octagon')
```

Custom kernels can be applied by following a similar format:

```
geometry:args
```

Where *geometry* is the size of the custom kernel, and *args* list a comma separated list of values. For example:

```
custom_kernel='5x3:nan,1,1,1,nan 1,1,1,1,1 nan,1,1,1,nan'
with Image(filename='rose:') as img:
    img.morphology(method='dilate', kernel=custom_kernel)
```

Parameters

- **method** (basestring) – effect function to apply. See *MORPHOLOGY_METHODS* for a list of methods.
- **kernel** (basestring) – shape to evaluate surrounding pixels. See *KERNEL_INFO_TYPES* for a list of built-in shapes.
- **iterations** (numbers.Integral) – Number of times a morphology method should be applied to the image. Default 1. Use -1 for unlimited iterations until the image is unchanged by the method operator.
- **channel** (*basestring*) – Optional color channel to target. See *CHANNELS*

New in version 0.5.0.

Changed in version 0.5.5: Added channel argument.

motion_blur(*radius=0.0*, *sigma=0.0*, *angle=0.0*, *channel=None*)

Apply a Gaussian blur along an angle direction. This simulates motion movement.

See Example of *Motion Blur*.

Parameters

- **radius** (numbers.Real) – Aperture size of the Gaussian operator.
- **sigma** (numbers.Real) – Standard deviation of the Gaussian operator.
- **angle** (numbers.Real) – Apply the effect along this angle.

New in version 0.5.4.

negate(*grayscale=False*, *channel=None*)

Negate the colors in the reference image.

Parameters

- **grayscale** (bool) – if set, only negate grayscale pixels in the image.
- **channel** (basestring) – the channel type. available values can be found in the *CHANNELS* mapping. If None, negate all channels.

New in version 0.3.8.

noise(*noise_type='uniform'*, *attenuate=1.0*, *channel=None*)

Adds noise to image.

See Example of *Add Noise*.

Parameters

- **noise_type** (basestring) – type of noise to apply. See *NOISE_TYPES*.
- **attenuate** (numbers.Real) – rate of distribution. Only available in ImageMagick-7. Default is 1.0.
- **channel** (basestring) – Optionally target a color channel to apply noise to. See *CHANNELS*.

New in version 0.5.3.

Changed in version 0.5.5: Added optional channel argument.

normalize(*channel=None*)

Normalize color channels.

Parameters **channel** (basestring) – the channel type. available values can be found in the *CHANNELS* mapping. If None, normalize all channels.

oil_paint(*radius=0.0*, *sigma=0.0*)
Simulates an oil painting by replace each pixel with most frequent surrounding color.

Parameters

- **radius** (numbers.Real) – The size of the surrounding neighbors.
- **sigma** (numbers.Real) – The standard deviation used by the Gaussian operator. This is only available with ImageMagick-7.

New in version 0.5.4.

opaque_paint(*target=None*, *fill=None*, *fuzz=0.0*, *invert=False*, *channel=None*)
Replace any color that matches `target` with `fill`. Use `fuzz` to control the threshold of the target match. The `invert` will replace all colors *but* the pixels matching the `target` color.

Parameters

- **target** (*wand.color.Color*) – The color to match.
- **fill** (*wand.color.Color*) – The color to paint with.
- **fuzz** (class:*numbers.Real*) – Normalized real number between *0.0* and *quantum_range*. Default is *0.0*.
- **invert** (bool) – Replace all colors that do not match target. Default is False.
- **channel** (basestring) – Optional color channel to target. See *CHANNELS*

New in version 0.5.4.

Changed in version 0.5.5: Added `channel` paramater.

optimize_layers()
Attempts to crop each frame to the smallest image without altering the animation. For best results, call *Image.coalesce()* before manipulating any frames. For timing accuracy, any *SingleImage.delay* overwrites must be applied after optimizing layers.

Note: This will only affect `GIF` image formates.

New in version 0.5.0.

optimize_transparency()
Iterates over frames, and sets transparent values for each pixel unchanged by previous frame.

Note: This will only affect `GIF` image formates.

New in version 0.5.0.

options = None
(*OptionDict*) The mapping of internal option settings.

New in version 0.3.0.

Changed in version 0.3.4: Added `'jpeg:sampling-factor'` option.

Changed in version 0.3.9: Added `'pdf:use-cropbox'` option.

ordered_dither(*threshold_map='threshold'*, *channel=None*)
Executes a ordered-based dither operations based on predetermined threshold maps.

Map	Alias	Description
threshold	1x1	Threshold 1x1 (non-dither)
checks	2x1	Checkerboard 2x1 (dither)
o2x2	2x2	Ordered 2x2 (dispersed)
o3x3	3x3	Ordered 3x3 (dispersed)
o4x4	4x4	Ordered 4x4 (dispersed)
o8x8	8x8	Ordered 8x8 (dispersed)
h4x4a	4x1	Halftone 4x4 (angled)
h6x6a	6x1	Halftone 6x6 (angled)
h8x8a	8x1	Halftone 8x8 (angled)
h4x4o		Halftone 4x4 (orthogonal)
h6x6o		Halftone 6x6 (orthogonal)
h8x8o		Halftone 8x8 (orthogonal)
h16x16o		Halftone 16x16 (orthogonal)
c5x5b	c5x5	Circles 5x5 (black)
c5x5w		Circles 5x5 (white)
c6x6b	c6x6	Circles 6x6 (black)
c6x6w		Circles 6x6 (white)
c7x7b	c7x7	Circles 7x7 (black)
c7x7w		Circles 7x7 (white)

Parameters

- **threshold_map** (basestring) – Name of threshold dither to use, followed by optional arguments.
- **channel** (basestring) – Optional argument to apply dither to specific color channel. See *CHANNELS*.

New in version 0.5.7.

property orientation
(basestring) The image orientation. It's a string from *ORIENTATION_TYPES* list. It also can be set.

New in version 0.3.0.

property page
The dimensions and offset of this Wand's page as a 4-tuple: `(width, height, x, y)`.

```
with Image(filename='wizard:') as img:
    img.page = (595, 842, 0, 0)
```

Note that since it is based on the virtual canvas, it may not equal the dimensions of an image. See the ImageMagick documentation on the virtual canvas for more information.

This attribute can also be set by using a named papersize. For example:

```
with Image(filename='wizard:') as img:
    img.page = 'a4'
```

New in version 0.4.3.

Changed in version 0.6.4: Added support for setting by papersize.

property page_height
(numbers.Integral) The height of the page for this wand.

New in version 0.4.3.

property page_width
(numbers.Integral) The width of the page for this wand.

New in version 0.4.3.

property page_x
(numbers.Integral) The X-offset of the page for this wand.

New in version 0.4.3.

property page_y
(numbers.Integral) The Y-offset of the page for this wand.

New in version 0.4.3.

parse_meta_geometry(*geometry*)
Helper method to translate geometry format, and calculate meta-characters against image dimensions.

See "Image Geometry" definitions & examples for more info: https://imagemagick.org/script/command-line-processing.php#geometry

Parameters **geometry** (basestring) – user string following ImageMagick's geometry format.

Returns Calculated width, height, offset-x, & offset-y.

Return type tuple

Raises **ValueError** – If given geometry can not be parsed.

New in version 0.5.6.

percent_escape(*string_format*)
Convenience method that expands ImageMagick's Percent Escape characters into image attribute values.

```
with wand.image import Image

with Image(filename='tests/assets/sasha.jpg') as img:
    print(img.percent_escape('%f %wx%h'))
    #=> sasha.jpg 204x247
```

Note: Not all percent escaped values can be populated as I/O operations are managed by Python, and not the CLI utility.

Parameters **string_format** (basestring) – The precent escaped string to be translated.

Returns String of expanded values.

Return type basestring

New in version 0.5.6.

polaroid(*angle=0.0*, *caption=None*, *font=None*, *method='undefined'*)
Creates a special effect simulating a Polaroid photo.

See Example of *Polaroid*.

Parameters

- **angle** (numbers.Real) – applies a shadow effect along this angle.
- **caption** (basestring) – Writes a message at the bottom of the photo's border.
- **font** (*wand.font.Font*) – Specify font style.
- **method** (basestring) – Interpolation method. ImageMagick-7 only.

New in version 0.5.4.

polynomial(*arguments*)

Replace image with the sum of all images in a sequence by calculating the pixel values a coefficient-weight value, and a polynomial-exponent.

For example:

```
with Image(filename='rose:') as img:
    img.polynomial(arguments=[0.5, 1.0])
```

The output image will be calculated as:

$$output = 0.5 * image^{1.0}$$

This can work on multiple images in a sequence by calculating across each frame in the image stack.

```
with Image(filename='2frames.gif') as img:
    img.polynomial(arguments=[0.5, 1.0, 0.25, 1.25])
```

Where the results would be calculated as:

$$output = 0.5 * frame1^{1.0} + 0.25 * frame2^{1.25}$$

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters **arguments** (collections.abc.Sequence) – A list of real numbers where at least two numbers (weight & exponent) are need for each image in the sequence.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

posterize(*levels=None*, *dither='no'*)

Reduce color levels per channel.

Parameters

- **levels** (numbers.Integral) – Number of levels per channel.
- **dither** (*basestring*) – Dither method to apply. See *DITHER_METHODS*.

New in version 0.5.0.

quantize(*number_colors*, *colorspace_type=None*, *treedepth=0*, *dither=False*, *measure_error=False*)

quantize analyzes the colors within a sequence of images and chooses a fixed number of colors to represent the image. The goal of the algorithm is to minimize the color difference between the input and output image while minimizing the processing time.

Parameters

- **number_colors** (numbers.Integral) – The target number of colors to reduce the image.
- **colorspace_type** (basestring) – Available value can be found in the *COLORSPACE_TYPES*. Defaults *colorspace*.
- **treedepth** (numbers.Integral) – A value between 0 & 8 where 0 will allow ImageMagick to calculate the optimal depth with Log4(number_colors). Default value is 0.
- **dither** (bool, or basestring) – Perform dither operation between neighboring pixel values. If using ImageMagick-6, this can be a value of True, or False. With ImageMagick-7, use a string from *DITHER_METHODS*. Default False.
- **measure_error** (bool) – Include total quantization error of all pixels in an image & quantized value.

New in version 0.4.2.

Changed in version 0.5.9: Fixed ImageMagick-7 dither argument, and added keyword defaults.

property quantum_range

(*:class: `numbers.Integral*) The maximum value of a color channel that is supported by the imagemagick library.

New in version 0.2.0.

random_threshold(*low=0.0*, *high=1.0*, *channel=None*)

Performs a random dither to force a pixel into a binary black & white state. Each color channel operarates independently from each other.

Parameters

- **low** (numbers.Real) – bottom threshold. Any pixel value below the given value will be rendered "0", or no value. Given threshold value can be between 0.0 & 1.0, or 0 & *quantum_range*.
- **high** (numbers.Real) – top threshold. Any pixel value above the given value will be rendered as max quantum value. Given threshold value can be between 0.0 & 1.0, or 0 & *quantum_range*.
- **channel** (basestring) – Optional argument to apply dither to specific color channel. See *CHANNELS*.

New in version 0.5.7.

range_channel(*channel='default_channels'*)

Calculate the minimum and maximum of quantum values in image.

```
from wand.image import Image

with Image(filename='input.jpg') as img:
    minima, maxima = img.range_channel()
```

Parameters **channel** (basestring) – Select which color channel to evaluate. See *CHANNELS*. Default 'default_channels'.

Returns Tuple of *minima* & *maxima* values. Each value will be between 0.0 & *quantum_range*.

Return type tuple

New in version 0.5.3.

range_threshold(*low_black=0.0*, *low_white=None*, *high_white=None*, *high_black=None*)
Applies soft & hard thresholding.

For a soft thresholding, parameters should be monotonically increasing:

with Image(filename='text.png') as img: img.range_threshold(0.2, 0.4, 0.6, 0.8)

For a hard thresholding, parameters should be the same:

with Image(filename='text.png') as img: img.range_threshold(0.4, 0.4, 0.6, 0.6)

> **Warning:** This class method is only available with ImageMagick 7.0.8-41, or greater.

Parameters

- **low_black** (numbers.Real) – Define the minimum threshold value.
- **low_white** (numbers.Real) – Define the minimum threshold value.
- **high_white** (numbers.Real) – Define the maximum threshold value.
- **high_black** (numbers.Real) – Define the maximum threshold value.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

read_mask(*clip_mask=None*)
Sets the read mask where the gray values of the clip mask are used to blend during composite operations. Call this method with a `None` argument to clear any previously set masks.

This method is also useful for *compare()* method for limiting region of interest.

> **Warning:** This method is only available with ImageMagick-7.

Parameters **clip_mask** (*BaseImage*) – Image to reference as blend mask.

New in version 0.5.7.

property red_primary
(tuple) The chromatic red primary point for the image. With ImageMagick-6 the primary value is `(x, y)` coordinates; however, ImageMagick-7 has `(x, y, z)`.

New in version 0.5.2.

region(*width=None*, *height=None*, *x=None*, *y=None*, *gravity=None*)
Extract an area of the image. This is the same as *crop()*, but returns a new instance of *Image* without altering the original source image.

```
from wand.image import Image

with Image(filename='input.jpg') as img:
    with img.region(width=100, height=100, x=0, y=0) as area:
        pass
```

Parameters

- **width** (numbers.Integral) – Area size on the X-axis. Default value is the image's *page_width*.
- **height** (numbers.Integral) – Area size on the Y-axis.Default value is the image's *page_height*.
- **x** (numbers.Integral) – X-axis offset. This number can be negative. Default value is the image's *page_x*.
- **y** (numbers.Integral) – Y-axis offset. This number can be negative. Default value is the image's *page_y*.
- **region** (basestring) – Helper attribute to set x & y offset. See *GRAVITY_TYPES*.

Returns New instance of Wand.

Return type *Image*

New in version 0.6.8.

remap(*affinity=None*, *method='no'*)

Rebuild image palette with closest color from given affinity image.

See Example of *Remap*.

Parameters

- **affinity** (*BaseImage*) – reference image.
- **method** (basestring) – dither method. See *DITHER_METHODS*. Default is 'no' dither.

New in version 0.5.3.

property rendering_intent

(basestring) PNG rendering intent. See *RENDERING_INTENT_TYPES* for valid options.

New in version 0.5.4.

resample(*x_res=None*, *y_res=None*, *filter='undefined'*, *blur=1*)

Adjust the number of pixels in an image so that when displayed at the given Resolution or Density the image will still look the same size in real world terms.

Note: This method will automatically *coalesce()* & resample all frames in a GIF animation. For other image formats, *resample()* will only effect the current image in the stack. Use *iterator_reset()* & *iterator_next()* to traverse the image stack to resample all images in a multi-layer document.

```
with Image(filename='input.tiff') as img:
    img.iterator_reset()
    while True:
        img.resample(128, 128)
        if not img.iterator_next():
            break
```

Parameters

- **x_res** (numbers.Real) – the X resolution (density) in the scaled image. default is the original resolution.
- **y_res** (numbers.Real) – the Y resolution (density) in the scaled image. default is the original resolution.

- **filter** (basestring, numbers.Integral) – a filter type to use for resizing. choose one in *FILTER_TYPES*. default is 'undefined' which means IM will try to guess best one to use.
- **blur** (numbers.Real) – the blur factor where > 1 is blurry, < 1 is sharp. default is 1

New in version 0.4.5.

reset_coords()
Reset the coordinate frame of the image so to the upper-left corner is (0, 0) again (crop and rotate operations change it).

New in version 0.2.0.

reset_sequence()
Abstract method prototype. See *wand.image.Image.reset_sequence()*.

New in version 0.6.0.

resize(*width=None*, *height=None*, *filter='undefined'*, *blur=1*)
Resizes the image.

Parameters

- **width** (numbers.Integral) – the width in the scaled image. default is the original width
- **height** (numbers.Integral) – the height in the scaled image. default is the original height
- **filter** (basestring, numbers.Integral) – a filter type to use for resizing. choose one in *FILTER_TYPES*. default is 'undefined' which means IM will try to guess best one to use
- **blur** (numbers.Real) – the blur factor where > 1 is blurry, < 1 is sharp. default is 1

Changed in version 0.2.1: The default value of filter has changed from 'triangle' to 'undefined' instead.

Changed in version 0.1.8: The blur parameter changed to take numbers.Real instead of numbers.Rational.

New in version 0.1.1.

property resolution
(tuple) Resolution of this image.

New in version 0.3.0.

Changed in version 0.5.8: Resolution returns a tuple of float values to match ImageMagick's behavior.

roll(*x=0*, *y=0*)
Shifts all pixels over by an X/Y offset.

Parameters

- **x** (numbers.Integral) – Number of columns to roll over. Negative value will roll pixels from right-to-left, and positive value will roll pixels from left-to-right. Default value: 0.
- **y** (numbers.Integral) – Number of rows to roll over. Negative value will roll pixels from bottom-to-top, and positive value will roll pixels from top-to-bottm. Default value: 0.

New in version 0.6.8.

rotate(*degree*, *background=None*, *reset_coords=True*)
Rotates the image right. It takes a background color for degree that isn't a multiple of 90.

See Example of *Rotation*.

Parameters

- **degree** (numbers.Real) – a degree to rotate. multiples of 360 affect nothing
- **background** (*wand.color.Color*) – an optional background color. default is transparent
- **reset_coords** (bool) – optional flag. If set, after the rotation, the coordinate frame will be relocated to the upper-left corner of the new image. By default is *True*.

New in version 0.2.0: The `reset_coords` parameter.

New in version 0.1.8.

rotational_blur(*angle=0.0*, *channel=None*)

Blur an image in a radius around the center of an image.

> **Warning:** Requires ImageMagick-6.8.8 or greater.

See Example of *Rotational Blur*.

Parameters

- **angle** (numbers.Real) – Degrees of rotation to blur with.
- **channel** (basestring) – Optional channel to apply the effect against. See *CHANNELS* for a list of possible values.

Raises ***WandLibraryVersionError*** – If system's version of ImageMagick does not support this method.

New in version 0.5.4.

sample(*width=None*, *height=None*)

Resizes the image by sampling the pixels. It's basically quicker than *resize()* except less quality as a trade-off.

Parameters

- **width** (numbers.Integral) – the width in the scaled image. default is the original width
- **height** (numbers.Integral) – the height in the scaled image. default is the original height

New in version 0.3.4.

property sampling_factors

(tuple) Factors used in sampling data streams. This can be set by given it a string `"4:2:2"`, or tuple of numbers `(2, 1, 1)`. However the given string value will be parsed to aspect ratio (i.e. `"4:2:2"` internally becomes `"2,1"`).

Note: This property is only used by YUV, DPX, & EXR encoders. For JPEG & TIFF set `"jpeg:sampling-factor"` on `Image.options` dictionary:

```
with Image(filename='input.jpg') as img:
    img.options['jpeg:sampling-factor'] = '2x1'
```

New in version 0.6.3.

scale(*columns=1*, *rows=1*)
Increase image size by scaling each pixel value by given `columns` and `rows`.

Parameters

- **columns** (`numbers.Integral`) – The number of columns, in pixels, to scale the image horizontally.
- **rows** (`numbers.Integral`) – The number of rows, in pixels, to scale the image vertically.

New in version 0.5.7.

property scene
(`numbers.Integral`) The scene number of the current frame within an animated image.

New in version 0.5.4.

property seed
(`numbers.Integral`) The seed for random number generator.

> **Warning:** This property is only available with ImageMagick 7.0.8-41, or greater.

New in version 0.5.5.

selective_blur(*radius=0.0*, *sigma=0.0*, *threshold=0.0*, *channel=None*)
Blur an image within a given threshold.

For best effects, use a value between 10% and 50% of *quantum_range*

```
from wand.image import Image

with Image(filename='photo.jpg') as img:
    # Apply 8x3 blur with a 10% threshold
    img.selective_blur(8.0, 3.0, 0.1 * img.quantum_range)
```

See Example of *Selective Blur*.

Parameters

- **radius** (`numbers.Real`) – Size of gaussian aperture.
- **sigma** (`numbers.Real`) – Standard deviation of gaussian operator.
- **threshold** (`numbers.Real`) – Only pixels within contrast threshold are effected. Value should be between `0.0` and *quantum_range*.
- **channel** (`basestring`) – Optional color channel to target. See *CHANNELS*

New in version 0.5.3.

Changed in version 0.5.5: Added `channel` argument.

sepia_tone(*threshold=0.8*)
Creates a Sepia Tone special effect similar to a darkroom chemical toning.

See Example of *Sepia Tone*.

Parameters **threshold** (`numbers.Real`) – The extent of the toning. Value can be between `0` & *quantum_range*, or `0` & `1.0`. Default value is `0.8` or "80%".

New in version 0.5.7.

sequence = None
(collections.abc.Sequence) The list of *SingleImage*s that the image contains.

New in version 0.3.0.

shade(*gray=False*, *azimuth=0.0*, *elevation=0.0*)
Creates a 3D effect by simulating a light from an elevated angle.

See Example of *Shade*.

Parameters

- **gray** (bool) – Isolate the effect on pixel intensity. Default is False.
- **azimuth** (numbers.Real) – Angle from x-axis.
- **elevation** (numbers.Real) – Amount of pixels from the z-axis.

New in version 0.5.0.

shadow(*alpha=0.0*, *sigma=0.0*, *x=0*, *y=0*)
Generates an image shadow.

Parameters

- **alpha** (numbers.Real) – Ratio of transparency.
- **sigma** (numbers.Real) – Standard deviation of the gaussian filter.
- **x** (numbers.Integral) – x-offset.
- **y** (numbers.Integral) – y-offset.

New in version 0.5.0.

sharpen(*radius=0.0*, *sigma=0.0*, *channel=None*)
Applies a gaussian effect to enhance the sharpness of an image.

Note: For best results, ensure `radius` is larger than `sigma`.

Defaults values of zero will have ImageMagick attempt to auto-select suitable values.

See Example of *Sharpen*.

Parameters

- **radius** (numbers.Real) – size of gaussian aperture.
- **sigma** (numbers.Real) – Standard deviation of the gaussian filter.
- **channel** (basestring) – Optional color channel to target. See *CHANNELS*.

New in version 0.5.0.

Changed in version 0.5.5: Added `channel` argument.

shave(*columns=0*, *rows=0*)
Remove pixels from the edges.

Parameters

- **columns** (numbers.Integral) – amount to shave off both sides of the x-axis.
- **rows** (numbers.Integral) – amount to shave off both sides of the y-axis.

New in version 0.5.0.

shear(*background='WHITE'*, *x=0.0*, *y=0.0*)
Shears the image to create a parallelogram, and fill the space created with a `background` color.

Parameters

- **background** (*wand.color.Color*) – Color to fill the void created by shearing the image.
- **x** (`numbers.Real`) – Slide the image along the X-axis.
- **y** (`numbers.Real`) – Slide the image along the Y-axis.

New in version 0.5.4.

sigmoidal_contrast(*sharpen=True*, *strength=0.0*, *midpoint=0.0*, *channel=None*)
Modifies the contrast of the image by applying non-linear sigmoidal algorithm.

```
with Image(filename='photo.jpg') as img:
    img.sigmoidal_contrast(sharpen=True,
                           strength=3,
                           midpoint=0.65 * img.quantum_range)
```

Parameters

- **sharpen** (`bool`) – Increase the contrast when `True` (default), else reduces contrast.
- **strength** (`numbers.Real`) – How much to adjust the contrast. Where a value of `0.0` has no effect, `3.0` is typical, and `20.0` is extreme.
- **midpoint** (`numbers.Real`) – Normalized value between *0.0* & *quantum_range*
- **channel** (`basestring`) – Optional color channel to target. See *CHANNELS*.

New in version 0.5.4.

Changed in version 0.5.5: Added `channel` argument.

property signature
(`str`) The SHA-256 message digest for the image pixel stream.

New in version 0.1.9.

similarity(*reference*, *threshold=0.0*, *metric='undefined'*)
Scan image for best matching `reference` image, and return location & similarity.

Use parameter `threshold` to stop subimage scanning if the matching similarity value is below the given value. This is the same as the CLI `-similarity-threshold` option.

This method will always return a location & the lowest computed similarity value. Users are responsible for checking the similarity value to determine if a matching location is valid. Traditionally, a similarity value greater than *0.3183099* is considered dissimilar.

```
from wand.image import Image

dissimilarity_threshold = 0.318
similarity_threshold = 0.05
with Image(filename='subject.jpg') as img:
    with Image(filename='object.jpg') as reference:
        location, diff = img.similarity(reference,
                                        similarity_threshold)
        if diff > dissimilarity_threshold:
            print('Images too dissimilar to match')
```

(continues on next page)

(continued from previous page)

```
        elif diff <= similarity_threshold:
            print('First match @ {left}x{top}'.format(**location))
        else:
            print('Best match @ {left}x{top}'.format(**location))
```

Warning: This operation can be slow to complete.

Parameters

- **reference** (*wand.image.Image*) – Image to search for.
- **threshold** (numbers.Real) – Stop scanning if reference similarity is below given threshold. Value can be between `0.0` and *quantum_range*. Default is `0.0`.
- **metric** (basestring) – specify which comparison algorithm to use. See *COMPARE_METRICS* for a list of values. Only used by ImageMagick-7.

Returns List of location & similarity value. Location being a dictionary of `width`, `height`, `left`, & `top`. The similarity value is the compare distance, so a value of `0.0` means an exact match.

Return type tuple (dict, numbers.Real)

New in version 0.5.4: has been added.

property size

(tuple) The pair of (*width*, *height*).

Note: When working with animations, or other layer-based image formats, the *width* & *height* properties are referencing the last frame read into the image stack. To get the *size* of the entire animated images, call *Image.coalesce()* method immediately after reading the image.

sketch(*radius=0.0*, *sigma=0.0*, *angle=0.0*)

Simulates a pencil sketch effect. For best results, `radius` value should be larger than `sigma`.

See Example of *Sketch*.

Parameters

- **radius** (numbers.Real) – size of Gaussian aperture.
- **sigma** (numbers.Real) – standard deviation of the Gaussian operator.
- **angle** (numbers.Real) – direction of blur.

New in version 0.5.3.

property skewness

(numbers.Real) The skewness of the image.

Tip: If you want both *kurtosis* & *skewness*, it would be faster to call *kurtosis_channel()* directly.

New in version 0.5.3.

smush(*stacked=False*, *offset=0*)

Appends all images together. Similar behavior to *concat()*, but with an optional offset between images.

Parameters

- **stacked** (bool) – If True, will join top-to-bottom. If False, join images from left-to-right (default).
- **offset** (numbers.Integral) – Minimum space (in pixels) between each join.

New in version 0.5.3.

solarize(*threshold=0.0*, *channel=None*)
Simulates extreme overexposure.

See Example of *Solarize*.

Parameters

- **threshold** (numbers.Real) – between `0.0` and *quantum_range*.
- **channel** (basestring) – Optional color channel to target. See *CHANNELS*

New in version 0.5.3.

Changed in version 0.5.5: Added `channel` argument.

sparse_color(*method*, *colors*, *channel_mask=7*)
Interpolates color values between points on an image.

The `colors` argument should be a dict mapping *Color* keys to coordinate tuples.

For example:

```
from wand.color import Color
from wand.image import Image

colors = {
    Color('RED'): (10, 50),
    Color('YELLOW'): (174, 32),
    Color('ORANGE'): (74, 123)
}
with Image(filename='input.png') as img:
    img.sparse_color('bilinear', colors)
```

The available interpolate methods are:

- `'barycentric'`
- `'bilinear'`
- `'shepards'`
- `'voronoi'`
- `'inverse'`
- `'manhattan'`

You can control which color channels are effected by building a custom channel mask. For example:

```
from wand.image import Image, CHANNELS

with Image(filename='input.png') as img:
    colors = {
        img[50, 50]: (50, 50),
```

(continues on next page)

(continued from previous page)

```
        img[100, 50]: (100, 50),
        img[50, 75]: (50, 75),
        img[100, 100]: (100, 100)
    }
    # Only apply Voronoi to Red & Alpha channels
    mask = CHANNELS['red'] | CHANNELS['alpha']
    img.sparse_color('voronoi', colors, channel_mask=mask)
```

Parameters

- **method** (basestring) – Interpolate method. See *SPARSE_COLOR_METHODS*
- **colors** (abc.Mapping { *Color*: (int, int) }) – A dictionary of *Color* keys mapped to an (x, y) coordinate tuple.
- **channel_mask** (numbers.Integral) – Isolate specific color channels to apply interpolation. Default to RGB channels.

New in version 0.5.3.

splice(*width=None*, *height=None*, *x=None*, *y=None*, *gravity=None*)

Partitions image by splicing a `width` x `height` rectangle at (`x`, `y`) offset coordinate. The space inserted will be replaced by the *background_color* value.

Parameters

- **width** (numbers.Integral) – number of pixel columns.
- **height** (numbers.Integral) – number of pixel rows.
- **x** (numbers.Integral) – offset on the X-axis.
- **y** (numbers.Integral) – offset on the Y-axis.

New in version 0.5.3.

spread(*radius=0.0*, *method='undefined'*)

Randomly displace pixels within a defined radius.

See Example of *Spread*.

Parameters

- **radius** (numbers.Real) – Distance a pixel can be displaced from source. Default value is `0.0`, which will allow ImageMagick to auto select a radius.
- **method** – Interpolation method. Only available with ImageMagick-7. See *PIXEL_INTERPOLATE_METHODS*.

New in version 0.5.3.

Changed in version 0.5.7: Added default value to `radius`.

property standard_deviation

(numbers.Real) The standard deviation of the image.

Tip: If you want both *mean* & *standard_deviation*, it would be faster to call *mean_channel()* directly.

New in version 0.5.3.

statistic(*stat='undefined'*, *width=None*, *height=None*, *channel=None*)
Replace each pixel with the statistic results from neighboring pixel values. The `width` & `height` defines the size, or aperture, of the neighboring pixels.

See Example of *Statistic*.

Parameters

- **stat** (`basestring`) – The type of statistic to calculate. See *STATISTIC_TYPES*.
- **width** (`numbers.Integral`) – The size of neighboring pixels on the X-axis.
- **height** (`numbers.Integral`) – The size of neighboring pixels on the Y-axis.
- **channel** (`basestring`) – Optional color channel to target. See *CHANNELS*

New in version 0.5.3.

Changed in version 0.5.5: Added optional `channel` argument.

stegano(*watermark*, *offset=0*)
Hide a digital watermark of an image within the image.

```
from wand.image import Image

# Embed watermark
with Image(filename='source.png') as img:
    with Image(filename='gray_watermark.png') as watermark:
        print('watermark size (for recovery)', watermark.size)
        img.stegano(watermark)
    img.save(filename='public.png')

# Recover watermark
with Image(width=w, height=h, pseudo='stegano:public.png') as img:
    img.save(filename='recovered_watermark.png')
```

Parameters

- **watermark** (*wand.image.Image*) – Image to hide within image.
- **offset** (`numbers.Integral`) – Start embedding image after a number of pixels.

New in version 0.5.4.

strip()
Strips an image of all profiles and comments.

New in version 0.2.0.

swirl(*degree=0.0*, *method='undefined'*)
Swirls pixels around the center of the image. The larger the degree the more pixels will be effected.

See Example of *Swirl*.

Parameters

- **degree** (`numbers.Real`) – Defines the amount of pixels to be effected. Value between `-360.0` and `360.0`.
- **method** (`basestring`) – Controls interpolation of the effected pixels. Only available for ImageMagick-7. See *PIXEL_INTERPOLATE_METHODS*.

New in version 0.5.7.

texture(*tile*)
Repeat tile-image across the width & height of the image.

```
from wand.image import Image

with Image(width=100, height=100) as canvas:
    with Image(filename='tile.png') as tile:
        canvas.texture(tile)
    canvas.save(filename='output.png')
```

Parameters **tile** (*Image*) – image to repeat across canvas.

New in version 0.5.4.

threshold(*threshold=0.5*, *channel=None*)
Changes the value of individual pixels based on the intensity of each pixel compared to threshold. The result is a high-contrast, two color image. It manipulates the image in place.

Parameters

- **threshold** (numbers.Real) – threshold as a factor of quantum. A normalized float between 0.0 and 1.0.
- **channel** (basestring) – the channel type. available values can be found in the *CHANNELS* mapping. If None, threshold all channels.

New in version 0.3.10.

thumbnail(*width=None*, *height=None*)
Changes the size of an image to the given dimensions and removes any associated profiles. The goal is to produce small low cost thumbnail images suited for display on the web.

Parameters

- **width** (numbers.Integral) – the width in the scaled image. default is the original width
- **height** (numbers.Integral) – the height in the scaled image. default is the original height

New in version 0.5.4.

property ticks_per_second
(numbers.Integral) Internal clock for animated images. .. versionadded:: 0.5.4

tint(*color=None*, *alpha=None*)
Applies a color vector to each pixel in the image.

See Example of *Tint*.

Parameters

- **color** (*Color*) – Color to calculate midtone.
- **alpha** (*Color*) – Determine how to blend.

New in version 0.5.3.

transform(*crop=''*, *resize=''*)
Transforms the image using MagickTransformImage(), which is a convenience function accepting geometry strings to perform cropping and resizing. Cropping is performed first, followed by resizing. Either or both arguments may be omitted or given an empty string, in which case the corresponding action will not be performed. Geometry specification strings are defined as follows:

A geometry string consists of a size followed by an optional offset. The size is specified by one of the options below, where **bold** terms are replaced with appropriate integer values:

scale% Height and width both scaled by specified percentage.

scale-x%xscale-y% Height and width individually scaled by specified percentages. Only one % symbol is needed.

width Width given, height automagically selected to preserve aspect ratio.

xheight Height given, width automagically selected to preserve aspect ratio.

widthxheight Maximum values of width and height given; aspect ratio preserved.

widthxheight! Width and height emphatically given; original aspect ratio ignored.

widthxheight> Shrinks images with dimension(s) larger than the corresponding width and/or height dimension(s).

widthxheight< Enlarges images with dimensions smaller than the corresponding width and/or height dimension(s).

area@ Resize image to have the specified area in pixels. Aspect ratio is preserved.

X:Y Resize at a given aspect ratio. Common aspect ratios may include `4:3` for video/tv, `3:2` for 35mm film, `16:9` for HDTV, and `2.39:1` for cinema. Aspect ratio can be used with the crop parameter, but is only available with ImageMagick version 7.0.8 or greater.

The offset, which only applies to the cropping geometry string, is given by `{+-}`**x**`{+-}`**y**, that is, one plus or minus sign followed by an **x** offset, followed by another plus or minus sign, followed by a **y** offset. Offsets are in pixels from the upper left corner of the image. Negative offsets will cause the corresponding number of pixels to be removed from the right or bottom edge of the image, meaning the cropped size will be the computed size minus the absolute value of the offset.

For example, if you want to crop your image to 300x300 pixels and then scale it by 2x for a final size of 600x600 pixels, you can call:

```
image.transform('300x300', '200%')
```

This method is a fairly thin wrapper for the C API, and does not perform any additional checking of the parameters except insofar as verifying that they are of the correct type. Thus, like the C API function, the method is very permissive in terms of what it accepts for geometry strings; unrecognized strings and trailing characters will be ignored rather than raising an error.

Parameters

- **crop** (`basestring`) – A geometry string defining a subregion of the image to crop to
- **resize** (`basestring`) – A geometry string defining the final size of the image

See also:

ImageMagick Geometry Specifications Cropping and resizing geometry for the `transform` method are specified according to ImageMagick's geometry string format. The ImageMagick documentation provides more information about geometry strings.

New in version 0.2.2.

Changed in version 0.5.0: Will call *crop()* followed by *resize()* in the event that `MagickTransformImage()` is not available.

Deprecated since version 0.6.0: Use *crop()* and *resize()* instead.

transform_colorspace(*colorspace_type*)
Transform image's colorspace.

Parameters **colorspace_type** (basestring) – colorspace_type. available value can be found in the *COLORSPACE_TYPES*

New in version 0.4.2.

transparent_color(*color*, *alpha*, *fuzz=0*, *invert=False*)
Makes the color `color` a transparent color with a tolerance of fuzz. The `alpha` parameter specify the transparency level and the parameter `fuzz` specify the tolerance.

Parameters

- **color** (*wand.color.Color*) – The color that should be made transparent on the image, color object
- **alpha** (numbers.Real) – the level of transparency: 1.0 is fully opaque and 0.0 is fully transparent.
- **fuzz** (numbers.Real) – By default target must match a particular pixel color exactly. However, in many cases two colors may differ by a small amount. The fuzz member of image defines how much tolerance is acceptable to consider two colors as the same. For example, set fuzz to 10 and the color red at intensities of 100 and 102 respectively are now interpreted as the same color for the color.
- **invert** (bool) – Boolean to tell to paint the inverse selection.

New in version 0.3.0.

Changed in version 0.6.3: Parameter `fuzz` type switched from Integral to Real.

transparentize(*transparency*)
Makes the image transparent by subtracting some percentage of the black color channel. The `transparency` parameter specifies the percentage.

Parameters **transparency** (numbers.Real) – the percentage fade that should be performed on the image, from 0.0 to 1.0

New in version 0.2.0.

transpose()
Creates a vertical mirror image by reflecting the pixels around the central x-axis while rotating them 90-degrees.

New in version 0.4.1.

transverse()
Creates a horizontal mirror image by reflecting the pixels around the central y-axis while rotating them 270-degrees.

New in version 0.4.1.

trim(*color=None*, *fuzz=0.0*, *reset_coords=False*, *percent_background=None*, *background_color=None*)
Remove solid border from image. Uses top left pixel as a guide by default, or you can also specify the `color` to remove.

Parameters

- **color** (*Color*) – the border color to remove. if it's omitted top left pixel is used by default
- **fuzz** (numbers.Real) – Defines how much tolerance is acceptable to consider two colors as the same. Value can be between `0.0`, and *quantum_range*.
- **reset_coords** (bool) – Reset coordinates after trimming image. Default `False`.

- **percent_background** (numbers.Real) – Sets how aggressive the trim operation will be. A value of *0.0* will trim to the minimal bounding box of all matching color, and *1.0* to the most outer edge.
- **background_color** – Local alias to *background_color*, and has the same effect as defining `color` parameter – but much faster.

New in version 0.2.1.

Changed in version 0.3.0: Optional `color` and `fuzz` parameters.

Changed in version 0.5.2: The `color` parameter may accept color-compliant strings.

Changed in version 0.6.0: Optional `reset_coords` parameter added.

Changed in version 0.6.4: Optional `percent_background` & `background_color` parameters have been added.

property type
(basestring) The image type.

Defines image type as in *IMAGE_TYPES* enumeration.

It may raise ValueError when the type is unknown.

New in version 0.2.2.

unique_colors()
Discards all duplicate pixels, and rebuilds the image as a single row.

New in version 0.5.0.

property units
(basestring) The resolution units of this image.

unsharp_mask(*radius=0.0*, *sigma=1.0*, *amount=1.0*, *threshold=0.0*, *channel=None*)
Sharpens the image using unsharp mask filter. We convolve the image with a Gaussian operator of the given `radius` and standard deviation (`sigma`). For reasonable results, `radius` should be larger than `sigma`. Use a radius of 0 and *unsharp_mask()* selects a suitable radius for you.

See Example of *Unsharp Mask*.

Parameters

- **radius** (numbers.Real) – the radius of the Gaussian, in pixels, not counting the center pixel
- **sigma** (numbers.Real) – the standard deviation of the Gaussian, in pixels
- **amount** (numbers.Real) – the percentage of the difference between the original and the blur image that is added back into the original
- **threshold** (numbers.Real) – the threshold in pixels needed to apply the difference amount.
- **channel** (basestring) – Optional color channel to target. See *CHANNELS*

New in version 0.3.4.

Changed in version 0.5.5: Added optional `channel` argument.

Changed in version 0.5.7: Added default values to match CLI behavior.

vignette(*radius=0.0*, *sigma=0.0*, *x=0*, *y=0*)
Creates a soft vignette style effect on the image.

See Example of *Vignette*.

Parameters

- **radius** (numbers.Real) – the radius of the Gaussian blur effect.
- **sigma** (numbers.Real) – the standard deviation of the Gaussian effect.
- **x** (numbers.Integral) – Number of pixels to offset inward from the top & bottom of the image before drawing effect.
- **y** (numbers.Integral) – Number of pixels to offset inward from the left & right of the image before drawing effect.

New in version 0.5.2.

property virtual_pixel
(basestring) The virtual pixel of image. This can also be set with a value from *VIRTUAL_PIXEL_METHOD* ... versionadded:: 0.4.1

property wand
Internal pointer to the MagickWand instance. It may raise *ClosedImageError* when the instance has destroyed already.

watermark(*image*, *transparency=0.0*, *left=0*, *top=0*)
Transparentized the supplied image and places it over the current image, with the top left corner of image at coordinates left, top of the current image. The dimensions of the current image are not changed.

Parameters

- **image** (*wand.image.Image*) – the image placed over the current image
- **transparency** (numbers.Real) – the percentage fade that should be performed on the image, from 0.0 to 1.0
- **left** (numbers.Integral) – the x-coordinate where *image* will be placed
- **top** (numbers.Integral) – the y-coordinate where *image* will be placed

New in version 0.2.0.

wave(*amplitude=0.0*, *wave_length=0.0*, *method='undefined'*)
Creates a ripple effect within the image.

See Example of *Wave*.

Parameters

- **amplitude** (numbers.Real) – height of wave form.
- **wave_length** (numbers.Real) – width of wave form.
- **method** (basestring) – pixel interpolation method. Only available with ImageMagick-7. See *PIXEL_INTERPOLATE_METHODS*

New in version 0.5.2.

wavelet_denoise(*threshold=0.0*, *softness=0.0*)
Removes noise by applying a wavelet transform.

Warning: This class method is only available with ImageMagick 7.0.8-41, or greater.

See Example of *Wavelet Denoise*.

Parameters

- **threshold** (numbers.Real) – Smoothing limit.
- **softness** (numbers.Real) – Attenuate of the smoothing threshold.

Raises *WandLibraryVersionError* – If system's version of ImageMagick does not support this method.

New in version 0.5.5.

white_balance()
Uses LAB colorspace to apply a white balance to the image.

Note: Requires ImageMagick-7.0.10-37 or later.

New in version 0.6.4.

property white_point
(tuple) The chromatic white point for the image. With ImageMagick-6 the primary value is `(x, y)` coordinates; however, ImageMagick-7 has `(x, y, z)`.

New in version 0.5.2.

white_threshold(*threshold*)
Forces all pixels above a given color as white. Leaves pixels below threshold unaltered.

Parameters **threshold** (Color) – Color to be referenced as a threshold.

New in version 0.5.2.

property width
(numbers.Integral) The width of this image.

write_mask(*clip_mask=None*)
Sets the write mask which prevents pixel-value updates to the image. Call this method with a None argument to clear any previously set masks.

Warning: This method is only available with ImageMagick-7.

Parameters **clip_mask** (*BaseImage*) – Image to reference as blend mask.

New in version 0.5.7.

wand.image.**CHANNELS** = {'all_channels': 134217727, 'alpha': 16, 'black': 8, 'blue': 4, 'composite_channels': 31, 'cyan': 1, 'default_channels': 134217727, 'gray': 1, 'gray_channels': 1, 'green': 2, 'index': 32, 'magenta': 2, 'meta': 256, 'opacity': 16, 'readmask': 64, 'red': 1, 'rgb': 7, 'rgb_channels': 7, 'sync_channels': 131072, 'true_alpha': 256, 'undefined': 0, 'write_mask': 128, 'yellow': 4}
(dict) The dictionary of channel types.

- 'undefined'
- 'red'
- 'gray'
- 'cyan'
- 'green'
- 'magenta'

- 'blue'
- 'yellow'
- 'alpha'
- 'opacity'
- 'black'
- 'index'
- 'composite_channels'
- 'all_channels'
- 'sync_channels'
- 'default_channels'

See also:

ImageMagick Color Channels Lists the various channel types with descriptions of each

Changed in version 0.5.5: Deprecated `true_alpha`, `rgb_channels`, and `gray_channels` values in favor of MagickCore channel parser.

wand.image.**COLORSPACE_TYPES = ('undefined', 'cmy', 'cmyk', 'gray', 'hcl', 'hclp', 'hsb', 'hsi', 'hsl', 'hsv', 'hwb', 'lab', 'lch', 'lchab', 'lchuv', 'log', 'lms', 'luv', 'ohta', 'rec601ycbcr', 'rec709ycbcr', 'rgb', 'scrgb', 'srgb', 'transparent', 'xyy', 'xyz', 'ycbcr', 'ycc', 'ydbdr', 'yiq', 'ypbpr', 'yuv')**

(`tuple`) The list of colorspaces.

- 'undefined'
- 'rgb'
- 'gray'
- 'transparent'
- 'ohta'
- 'lab'
- 'xyz'
- 'ycbcr'
- 'ycc'
- 'yiq'
- 'ypbpr'
- 'yuv'
- 'cmyk'
- 'srgb'
- 'hsb'
- 'hsl'
- 'hwb'
- 'rec601luma' - Only available with ImageMagick-6

- 'rec601ycbcr'
- 'rec709luma' - Only available with ImageMagick-6
- 'rec709ycbcr'
- 'log'
- 'cmy'
- 'luv'
- 'hcl'
- 'lch'
- 'lms'
- 'lchab'
- 'lchuv'
- 'scrgb'
- 'hsi'
- 'hsv'
- 'hclp'
- 'xyy' - Only available with ImageMagick-7
- 'ydbdr'

See also:

ImageMagick Color Management Describes the ImageMagick color management operations

New in version 0.3.4.

wand.image.**COMPARE_METRICS = ('undefined', 'absolute', 'fuzz', 'mean_absolute', 'mean_error_per_pixel', 'mean_squared', 'normalized_cross_correlation', 'peak_absolute', 'peak_signal_to_noise_ratio', 'perceptual_hash', 'root_mean_square', 'structural_similarity', 'structural_dissimilarity')**

(tuple) The list of compare metric types used by *Image.compare()* and *Image.similarity()* methods.

- 'undefined'
- 'absolute'
- 'fuzz'
- 'mean_absolute'
- 'mean_error_per_pixel'
- 'mean_squared'
- 'normalized_cross_correlation'
- 'peak_absolute'
- 'peak_signal_to_noise_ratio'
- 'perceptual_hash' - Available with ImageMagick-7
- 'root_mean_square'
- 'structural_similarity' - Available with ImageMagick-7

- 'structural_dissimilarity' - Available with ImageMagick-7

See also:

ImageMagick Compare Operations

New in version 0.4.3.

Changed in version 0.5.4: - Remapped MetricType enum.

wand.image.**COMPOSITE_OPERATORS** = ('undefined', 'alpha', 'atop', 'blend', 'blur', 'bumpmap', 'change_mask', 'clear', 'color_burn', 'color_dodge', 'colorize', 'copy_black', 'copy_blue', 'copy', 'copy_cyan', 'copy_green', 'copy_magenta', 'copy_alpha', 'copy_red', 'copy_yellow', 'darken', 'darken_intensity', 'difference', 'displace', 'dissolve', 'distort', 'divide_dst', 'divide_src', 'dst_atop', 'dst', 'dst_in', 'dst_out', 'dst_over', 'exclusion', 'hard_light', 'hard_mix', 'hue', 'in', 'intensity', 'lighten', 'lighten_intensity', 'linear_burn', 'linear_dodge', 'linear_light', 'luminize', 'mathematics', 'minus_dst', 'minus_src', 'modulate', 'modulus_add', 'modulus_subtract', 'multiply', 'no', 'out', 'over', 'overlay', 'pegtop_light', 'pin_light', 'plus', 'replace', 'saturate', 'screen', 'soft_light', 'src_atop', 'src', 'src_in', 'src_out', 'src_over', 'threshold', 'vivid_light', 'xor', 'stereo')

(tuple) The list of composition operators

- 'undefined'
- 'alpha' - Only available with ImageMagick-7
- 'atop'
- 'blend'
- 'blur'
- 'bumpmap'
- 'change_mask'
- 'clear'
- 'color_burn'
- 'color_dodge'
- 'colorize'
- 'copy_black'
- 'copy_blue'
- 'copy'
- 'copy_alpha' - Only available with ImageMagick-7
- 'copy_cyan'
- 'copy_green'
- 'copy_magenta'
- 'copy_opacity' - Only available with ImageMagick-6
- 'copy_red'
- 'copy_yellow'
- 'darken'
- 'darken_intensity'

- 'difference'
- 'displace'
- 'dissolve'
- 'distort'
- 'divide_dst'
- 'divide_src'
- 'dst_atop'
- 'dst'
- 'dst_in'
- 'dst_out'
- 'dst_over'
- 'exclusion'
- 'hard_light'
- 'hard_mix'
- 'hue'
- 'in'
- 'intensity' - Only available with ImageMagick-7
- 'lighten'
- 'lighten_intensity'
- 'linear_burn'
- 'linear_dodge'
- 'linear_light'
- 'luminize'
- 'mathematics'
- 'minus_dst'
- 'minus_src'
- 'modulate'
- 'modulus_add'
- 'modulus_subtract'
- 'multiply'
- 'no'
- 'out'
- 'over'
- 'overlay'
- 'pegtop_light'
- 'pin_light'

- 'plus'
- 'replace'
- 'saturate'
- 'screen'
- 'soft_light'
- 'src_atop'
- 'src'
- 'src_in'
- 'src_out'
- 'src_over'
- 'threshold'
- 'vivid_light'
- 'xor'
- 'stereo'

Changed in version 0.3.0: Renamed from COMPOSITE_OPS to *COMPOSITE_OPERATORS*.

Changed in version 0.5.6: Operators have been updated to reflect latest changes in C-API. For ImageMagick-6, 'add' has been renamed to 'modulus_add', 'subtract' has been renamed to 'modulus_subtract', 'divide' has been split into 'divide_dst' & 'divide_src', and 'minus' has been split into 'minus_dst' & 'minus_src'.

See also:

Compositing Images **ImageMagick v6 Examples** Image composition is the technique of combining images that have, or do not have, transparency or an alpha channel. This is usually performed using the IM **composite** command. It may also be performed as either part of a larger sequence of operations or internally by other image operators.

ImageMagick Composition Operators Demonstrates the results of applying the various composition composition operators.

wand.image.**COMPRESSION_TYPES** = ('undefined', 'b44a', 'b44', 'bzip', 'dxt1', 'dxt3', 'dxt5', 'fax', 'group4', 'jbig1', 'jbig2', 'jpeg2000', 'jpeg', 'losslessjpeg', 'lzma', 'lzw', 'no', 'piz', 'pxr24', 'rle', 'zip', 'zips')

(tuple) The list of Image.compression types.

- 'undefined'
- 'b44a'
- 'b44'
- 'bzip'
- 'dxt1'
- 'dxt3'
- 'dxt5'
- 'fax'
- 'group4'

- 'jbig1'
- 'jbig2'
- 'jpeg2000'
- 'jpeg'
- 'losslessjpeg'
- 'lzma'
- 'lzw'
- 'no'
- 'piz'
- 'pxr24'
- 'rle'
- 'zip'
- 'zips'

New in version 0.3.6.

Changed in version 0.5.0: Support for ImageMagick-6 & ImageMagick-7

class wand.image.**ChannelDepthDict**(*image*)
The mapping table of channels to their depth.

Parameters **image** (*Image*) – an image instance

Note: You don't have to use this by yourself. Use *Image.channel_depths* property instead.

New in version 0.3.0.

class wand.image.**ChannelImageDict**(*image*)
The mapping table of separated images of the particular channel from the image.

Parameters **image** (*Image*) – an image instance

Note: You don't have to use this by yourself. Use *Image.channel_images* property instead.

New in version 0.3.0.

exception wand.image.**ClosedImageError**
An error that rises when some code tries access to an already closed image.

class wand.image.**ConnectedComponentObject**(*cc_object=None*)
Generic Python wrapper to translate `CCObjectInfo` structure into a class describing objects found within an image. This class is generated by *Image.connected_components()* method.

New in version 0.5.5.

Changed in version 0.6.3: Added *merge* & *metric* for ImageMagick 7.0.10

Changed in version 0.6.8: Added `key` property for ImageMagick 7.1.0

area = None
(numbers.Real) Quantity of pixels that make-up the objects shape.

center_x = None
(numbers.Real) X offset of objects centroid.

center_y = None
(numbers.Real) Y offset of objects centroid.

property centroid
(tuple (*center_x*, *center_y*)) Center of object.

clone_from_cc_object_info(*cc_object*)
Copy data from CCObjectInfo.

clone_from_extra_70A_info(*cc_object*)
Copy the additional values from CCObjectInfo structure. This is the *merge* & *metric* properties added in ImageMagick 7.0.10.

New in version 0.6.3.

clone_from_extra_710_info(*cc_object*)
Copy additional value from CCObjectInfo structure for properties added to ImageMagick 7.1.0.

New in version 0.6.8.

height = None
(numbers.Integral) Height of objects minimum bounding rectangle.

left = None
(numbers.Integral) X offset of objects minimum bounding rectangle.

mean_color = None
(*Color*) The average color of the shape.

merge = None
(bool) Object merge flag. Only avaliable after ImageMagick-7.0.10. ..versionadded:: 0.6.3

metric = None
(list) List of doubles used by metric. Only avaliable after ImageMagick-7.0.10. ..versionadded:: 0.6.3

property offset
(tuple (*left*, *top*)) Position of objects minimum bounding rectangle.

property size
(tuple (*width*, *height*)) Minimum bounding rectangle.

top = None
(numbers.Integral) Y offset of objects minimum bounding rectangle.

width = None
(numbers.Integral) Width of objects minimum bounding rectangle.

wand.image.**DISPOSE_TYPES = ('undefined', 'none', 'background', 'previous')**
(tuple) The list of *BaseImage.dispose* types.

- 'undefined'
- 'none'
- 'background'
- 'previous'

New in version 0.5.0.

wand.image.**DISTORTION_METHODS = ('undefined', 'affine', 'affine_projection', 'scale_rotate_translate', 'perspective', 'perspective_projection', 'bilinear_forward', 'bilinear_reverse', 'polynomial', 'arc', 'polar', 'depolar', 'cylinder_2_plane', 'plane_2_cylinder', 'barrel', 'barrel_inverse', 'shepards', 'resize', 'sentinel', 'rigidaffine')**

(tuple) The list of *BaseImage.distort()* methods.

- 'undefined'
- 'affine'
- 'affine_projection'
- 'scale_rotate_translate'
- 'perspective'
- 'perspective_projection'
- 'bilinear_forward'
- 'bilinear_reverse'
- 'polynomial'
- 'arc'
- 'polar'
- 'depolar'
- 'cylinder_2_plane'
- 'plane_2_cylinder'
- 'barrel'
- 'barrel_inverse'
- 'shepards'
- 'resize'
- 'sentinel'
- 'rigidaffine' - Only available with ImageMagick-7.0.10, or later.

New in version 0.4.1.

wand.image.**DITHER_METHODS = ('undefined', 'no', 'riemersma', 'floyd_steinberg')**

(tuple) The list of Dither methods. Used by *Image.posterize()* and *Image.remap()* methods.

- 'undefined'
- 'no'
- 'riemersma'
- 'floyd_steinberg'

New in version 0.5.0.

wand.image.**EVALUATE_OPS = ('undefined', 'abs', 'add', 'addmodulus', 'and', 'cosine', 'divide', 'exponential', 'gaussiannoise', 'impulsenoise', 'laplaciannoise', 'leftshift', 'log', 'max', 'mean', 'median', 'min', 'multiplicativenoise', 'multiply', 'or', 'poissonnoise', 'pow', 'rightshift', 'rootmeansquare', 'set', 'sine', 'subtract', 'sum', 'thresholdblack', 'threshold', 'thresholdwhite', 'uniformnoise', 'xor', 'inverse_log')**

(tuple) The list of evaluation operators. Used by *Image.evaluate()* method.

- 'undefined'
- 'abs'
- 'add'
- 'addmodulus'
- 'and'
- 'cosine'
- 'divide'
- 'exponential'
- 'gaussiannoise'
- 'impulsenoise'
- 'inverse_log' - Added with ImageMagick-7.0.10-24
- 'laplaciannoise'
- 'leftshift'
- 'log'
- 'max'
- 'mean'
- 'median'
- 'min'
- 'multiplicativenoise'
- 'multiply'
- 'or'
- 'poissonnoise'
- 'pow'
- 'rightshift'
- 'set'
- 'sine'
- 'subtract'
- 'sum'
- 'threshold'
- 'thresholdblack'
- 'thresholdwhite'
- 'uniformnoise'
- 'xor'

See also:

ImageMagick Image Evaluation Operators Describes the MagickEvaluateImageChannel method and lists the various evaluations operators

wand.image.**FILTER_TYPES = ('undefined', 'point', 'box', 'triangle', 'hermite', 'hanning', 'hamming', 'blackman', 'gaussian', 'quadratic', 'cubic', 'catrom', 'mitchell', 'jinc', 'sinc', 'sincfast', 'kaiser', 'welsh', 'parzen', 'bohman', 'bartlett', 'lagrange', 'lanczos', 'lanczossharp', 'lanczos2', 'lanczos2sharp', 'robidoux', 'robidouxsharp', 'cosine', 'spline', 'sentinel')**

(tuple) The list of filter types. Used by *Image.resample()* and *Image.resize()* methods.

- 'undefined'
- 'point'
- 'box'
- 'triangle'
- 'hermite'
- 'hanning'
- 'hamming'
- 'blackman'
- 'gaussian'
- 'quadratic'
- 'cubic'
- 'catrom'
- 'mitchell'
- 'jinc'
- 'sinc'
- 'sincfast'
- 'kaiser'
- 'welsh'
- 'parzen'
- 'bohman'
- 'bartlett'
- 'lagrange'
- 'lanczos'
- 'lanczossharp'
- 'lanczos2'
- 'lanczos2sharp'
- 'robidoux'
- 'robidouxsharp'
- 'cosine'
- 'spline'
- 'sentinel'

See also:

ImageMagick Resize Filters Demonstrates the results of resampling images using the various resize filters and blur settings available in ImageMagick.

wand.image.**FUNCTION_TYPES = ('undefined', 'arcsin', 'arctan', 'polynomial', 'sinusoid')**
(tuple) The list of *Image.function* types.

- 'undefined'
- 'arcsin'
- 'arctan'
- 'polynomial'
- 'sinusoid'

wand.image.**GRAVITY_TYPES = ('forget', 'north_west', 'north', 'north_east', 'west', 'center', 'east', 'south_west', 'south', 'south_east', 'static')**
(tuple) The list of *gravity* types.

- 'forget'
- 'north_west'
- 'north'
- 'north_east'
- 'west'
- 'center'
- 'east'
- 'south_west'
- 'south'
- 'south_east'

New in version 0.3.0.

class wand.image.**HistogramDict**(*image*)
Specialized mapping object to represent color histogram. Keys are colors, and values are the number of pixels.

Parameters **image** (*BaseImage*) – the image to get its histogram

New in version 0.3.0.

wand.image.**IMAGE_LAYER_METHOD = ('undefined', 'coalesce', 'compareany', 'compareclear', 'compareoverlay', 'dispose', 'optimize', 'optimizeimage', 'optimizeplus', 'optimizetrans', 'removedups', 'removezero', 'composite', 'merge', 'flatten', 'mosaic', 'trimbounds')**
(tuple) The list of methods for *merge_layers()* and *compare_layers()*.

- 'undefined'
- 'coalesce'
- 'compareany' - Only used for *compare_layers()*.
- 'compareclear' - Only used for *compare_layers()*.
- 'compareoverlay' - Only used for *compare_layers()*.
- 'dispose'
- 'optimize'

- 'optimizeimage'
- 'optimizeplus'
- 'optimizetrans'
- 'removedups'
- 'removezero'
- 'composite'
- 'merge' - Only used for *merge_layers()*.
- 'flatten' - Only used for *merge_layers()*.
- 'mosaic' - Only used for *merge_layers()*.
- 'trimbounds' - Only used for *merge_layers()*.

New in version 0.4.3.

wand.image.**IMAGE_TYPES = ('undefined', 'bilevel', 'grayscale', 'grayscalealpha', 'palette', 'palettealpha', 'truecolor', 'truecoloralpha', 'colorseparation', 'colorseparationalpha', 'optimize', 'palettebilevelalpha')**

(tuple) The list of image types

- 'undefined'
- 'bilevel'
- 'grayscale'
- 'grayscalealpha' - Only available with ImageMagick-7
- 'grayscalematte' - Only available with ImageMagick-6
- 'palette'
- 'palettealpha' - Only available with ImageMagick-7
- 'palettematte' - Only available with ImageMagick-6
- 'truecolor'
- 'truecoloralpha' - Only available with ImageMagick-7
- 'truecolormatte' - Only available with ImageMagick-6
- 'colorseparation'
- 'colorseparationalpha' - Only available with ImageMagick-7
- 'colorseparationmatte' - Only available with ImageMagick-6
- 'optimize'
- 'palettebilevelalpha' - Only available with ImageMagick-7
- 'palettebilevelmatte' - Only available with ImageMagick-6

See also:

ImageMagick Image Types Describes the MagickSetImageType method which can be used to set the type of an image

wand.image.**INTERLACE_TYPES = ('undefined', 'no', 'line', 'plane', 'partition', 'gif', 'jpeg', 'png')**

(tuple) The list of interlace schemes.

- 'undefined'
- 'no'
- 'line'
- 'plane'
- 'partition'
- 'gif'
- 'jpeg'
- 'png'

New in version 0.5.2.

class wand.image.**Image**(*image=None*, *blob=None*, *file=None*, *filename=None*, *pseudo=None*, *background=None*, *colorspace=None*, *depth=None*, *extract=None*, *format=None*, *height=None*, *interlace=None*, *resolution=None*, *sampling_factors=None*, *units=None*, *width=None*)

An image object.

Parameters

- **image** (*Image*) – makes an exact copy of the image
- **blob** (bytes) – opens an image of the blob byte array
- **file** (*file object*) – opens an image of the file object
- **filename** (basestring) – opens an image of the filename string. Additional *Read Modifiers* are supported.
- **format** (basestring) – forces filename to buffer. format to help ImageMagick detect the file format. Used only in blob or file cases
- **width** (numbers.Integral) – the width of new blank image or an image loaded from raw data.
- **height** (numbers.Integral) – the height of new blank image or an image loaded from raw data.
- **depth** (numbers.Integral) – the depth used when loading raw data.
- **background** (*wand.color.Color*) – an optional background color. default is transparent
- **resolution** (collections.abc.Sequence, numbers.Integral) – set a resolution value (dpi), useful for vectorial formats (like pdf)
- **colorspace** (basestring) – sets the stack's default colorspace value before reading any images. See *COLORSPACE_TYPES*.
- **units** (basestring) – paired with resolution for defining an image's pixel density. See *UNIT_TYPES*.

New in version 0.1.5: The file parameter.

New in version 0.1.1: The blob parameter.

New in version 0.2.1: The format parameter.

New in version 0.2.2: The `width`, `height`, `background` parameters.

New in version 0.3.0: The `resolution` parameter.

New in version 0.4.2: The `depth` parameter.

Changed in version 0.4.2: The `depth`, `width` and `height` parameters can be used with the `filename`, `file` and `blob` parameters to load raw pixel data.

New in version 0.5.0: The `pseudo` parameter.

Changed in version 0.5.4: Read constructor no longer sets "transparent" background by default. Use the `background` paramater to specify canvas color when reading in image.

Changed in version 0.5.7: Added the `colorspace` & `units` parameter.

Changed in version 0.6.3: Added `sampling_factors` parameter for working with YUV streams.

[left:right, top:bottom]

Crops the image by its `left`, `right`, `top` and `bottom`, and then returns the cropped one.

```
with img[100:200, 150:300] as cropped:
    # manipulated the cropped image
    pass
```

Like other subscriptable objects, default is 0 or its width/height:

```
img[:, :]        #--> just clone
img[:100, 200:]  #--> equivalent to img[0:100, 200:img.height]
```

Negative integers count from the end (width/height):

```
img[-70:-50, -20:-10]
#--> equivalent to img[width-70:width-50, height-20:height-10]
```

Returns the cropped image

Rtype *Image*

New in version 0.1.2.

property animation

(`bool`) Whether the image is animation or not. It doesn't only mean that the image has two or more images (frames), but all frames are even the same size. It's about image format, not content. It's `False` even if *image/ico* consits of two or more images of the same size.

For example, it's `False` for *image/jpeg*, *image/gif*, *image/ico*.

If *image/gif* has two or more frames, it's `True`. If *image/gif* has only one frame, it's `False`.

New in version 0.3.0.

Changed in version 0.3.8: Became to accept *image/x-gif* as well.

artifacts = None

(*ArtifactTree*) A dict mapping to image artifacts. Similar to *metadata*, but used to alter behavior of various internal operations.

New in version 0.5.0.

blank(*width*, *height*, *background=None*)

Creates blank image.

Parameters

- **width** (numbers.Integral) – the width of new blank image.
- **height** (numbers.Integral) – the height of new blank image.
- **background** (*wand.color.Color*) – an optional background color. default is transparent

Returns blank image

Return type *Image*

New in version 0.3.0.

channel_depths = None
(*ChannelDepthDict*) The mapping of channels to their depth. Read only.

New in version 0.3.0.

channel_images = None
(*ChannelImageDict*) The mapping of separated channels from the image.

```
with image.channel_images['red'] as red_image:
    display(red_image)
```

clear()
Clears resources associated with the image, leaving the image blank, and ready to be used with new image.

New in version 0.3.0.

close()
Closes the image explicitly. If you use the image object in with statement, it was called implicitly so don't have to call it.

Note: It has the same functionality of destroy() method.

compare_layers(*method*)
Generates new images showing the delta pixels between layers. Similar pixels are converted to transparent. Useful for debugging complex animations.

```
with img.compare_layers('compareany') as delta:
    delta.save(filename='framediff_%02d.png')
```

Note: May not work as expected if animations are already optimized.

Parameters **method** (basestring) – Can be 'compareany', 'compareclear', or 'compareoverlay'

Returns new image stack.

Return type *Image*

New in version 0.5.0.

convert(*format*)
Converts the image format with the original image maintained. It returns a converted image instance which is new.

```
with img.convert('png') as converted:
    converted.save(filename='converted.png')
```

Parameters **format** (basestring) – image format to convert to

Returns a converted image

Return type *Image*

Raises **ValueError** – when the given format is unsupported

New in version 0.1.6.

data_url()

Generate a base64 data-url string from the loaded image. Useful for converting small graphics into ASCII strings for HTML/CSS web development.

Returns a data-url formated string.

Return type basestring

New in version 0.6.3.

classmethod from_array(*array*, *channel_map=None*, *storage=None*)

Create an image instance from a numpy array, or any other datatype that implements __array_interface__ protocol.

```
import numpy
from wand.image import Image

matrix = numpy.random.rand(100, 100, 3)
with Image.from_array(matrix) as img:
    img.save(filename='noise.png')
```

Use the optional channel_map & storage arguments to specify the order of color channels & data size. If channel_map is omitted, this method will will guess "RGB", or "CMYK" based on array shape. If storage is omitted, this method will reference the array's typestr value, and raise a ValueError if storage-type can not be mapped.

Float values must be normalized between *0.0* and *1.0*, and signed integers should be converted to unsigned values between *0* and max value of type.

Instances of *Image* can also be exported to numpy arrays:

```
with Image(filename='rose:') as img:
    matrix = numpy.array(img)
```

Parameters

- **array** (numpy.array) – Numpy array of pixel values.
- **channel_map** (basestring) – Color channel layout.
- **storage** (basestring) – Datatype per pixel part.

Returns New instance of an image.

Return type *Image*

New in version 0.5.3.

Changed in version 0.6.0: Input `array` now expects the `shape` property to be defined as `` `( 'height', 'width', 'channels' )` ``.

image_add(*image*)
Copies a given image on to the image stack. By default, the added image will be append at the end of the stack, or immediately after the current image iterator defined by *iterator_set()*. Use *iterator_reset()* before calling this method to insert the new image before existing images on the stack.

Parameters **image** (*Image*) – raster to add.

New in version 0.6.7.

image_get()
Generate & return a clone of a single image at the current image-stack index.

New in version 0.6.7.

image_remove()
Remove an image from the image-stack at the current index.

New in version 0.6.7.

image_set(*image*)
Overwrite current image on the image-stack with given image.

Parameters **image** (*wand.image.Image*) – Wand instance of images to write to stack.

New in version 0.6.7.

image_swap(*i*, *j*)
Swap two images on the image-stack.

Parameters

- **i** (numbers.Integral) – image index to replace with `j`
- **j** (numbers.Integral) – image index to replace with `i`

New in version 0.6.7.

make_blob(*format=None*)
Makes the binary string of the image.

Parameters **format** (basestring) – the image format to write e.g. `'png'`, `'jpeg'`. it is omittable

Returns a blob (bytes) string

Return type bytes

Raises **ValueError** – when `format` is invalid

Changed in version 0.1.6: Removed a side effect that changes the image `format` silently.

New in version 0.1.5: The `format` parameter became optional.

New in version 0.1.1.

metadata = None
(*Metadata*) The metadata mapping of the image. Read only.

New in version 0.3.0.

property mimetype
(basestring) The MIME type of the image e.g. 'image/jpeg', 'image/png'.

New in version 0.1.7.

montage(*font=None*, *tile=None*, *thumbnail=None*, *mode='unframe'*, *frame=None*)
Generates a new image containing thumbnails if each previous image read.

```
with Image() as img:
    for file_path in ['first.png', 'second.png', 'third.png']:
        with Image(filename=file_path) as item:
            img.options['label'] = file_path
            img.image_add(item)
    style = Font('monospace', 24, 'green')
    img.montage(font=style, tile='3x1', thumbnail='15x15')
    img.save(filename='montage.png')
```

Parameters

- **font** (*Font*) – Define font style to use when labeling each thumbnail. Thumbnail labeling will only be rendered if 'label' value in options dict is defined.
- **tile** (basestring) – The number of thunbnails per rows & column on a page. Example: "6x4".
- **thumbnail** (basestring) – Preferred image size. Montage will attempt to generate a thumbnail to match the geometry. This can also define the border size on each thumbnail. Example: "120x120x+4+3>".
- **mode** (basestring) – Which effect to render. Options include "frame", "unframe", and "concatenate". Default "frame".
- **frame** (basestring) – Define ornamental boarder around each thrumbnail. The color of the frame is defined by the image's matte color. Example: "15x15+3+3".

New in version 0.6.8.

classmethod ping(*file=None*, *filename=None*, *blob=None*, ***kwargs*)
Ping image header into Image() object, but without any pixel data. This is useful for inspecting image meta-data without decoding the whole image.

Parameters

- **blob** (bytes) – reads an image from the blob byte array
- **file** (*file object*) – reads an image from the file object
- **filename** (basestring) – reads an image from the filename string
- **resolution** (collections.abc.Sequence, numbers.Integral) – set a resolution value (DPI), useful for vector formats (like PDF)
- **format** (basestring) – suggest image file format when reading from a blob, or file property.

New in version 0.5.6.

profiles = None
(*ProfileDict*) The mapping of image profiles.

New in version 0.5.1.

pseudo(*width*, *height*, *pseudo='xc:'*)
Creates a new image from ImageMagick's internal protocol coders.

Parameters

- **width** (numbers.Integral) – Total columns of the new image.
- **height** (numbers.Integral) – Total rows of the new image.
- **pseudo** (basestring) – The protocol & arguments for the pseudo image.

New in version 0.5.0.

read(*file=None*, *filename=None*, *blob=None*, *background=None*, *colorspace=None*, *depth=None*, *extract=None*, *format=None*, *height=None*, *interlace=None*, *resolution=None*, *sampling_factors=None*, *units=None*, *width=None*)
Read new image into Image() object.

Parameters

- **blob** (bytes) – reads an image from the `blob` byte array
- **file** (*file object*) – reads an image from the `file` object
- **filename** (basestring) – reads an image from the `filename` string. Additional *Read Modifiers* are supported.
- **background** (Color, basestring) – set default background color.
- **colorspace** (basestring) – set default colorspace. See *COLORSPACE_TYPES*.
- **depth** (numbers.Integral) – sets bits per color sample. Usually `8`, or `16`.
- **format** (basestring) – sets which image decoder to read with. Helpful when reading `blob` data with ambiguous headers.
- **height** (numbers.Integral) – used with `width` to define the canvas size. Useful for reading image streams.
- **interlace** (basestring) – Defines the interlacing scheme for raw data streams. See *INTERLACE_TYPES*.
- **resolution** (collections.abc.Sequence, numbers.Integral) – set a resolution value (DPI), useful for vectorial formats (like PDF)
- **sampling_factors** (collections.abc.Sequence, basestring) – set up/down stampling factors for YUV data stream. Usually `"4:2:2"`
- **units** (basestring) – used with `resolution`, can either be `'pixelperinch'`, or `'pixelpercentimeter'`.
- **width** (numbers.Integral) – used with `height` to define the canvas size. Useful for reading image streams.

New in version 0.3.0.

Changed in version 0.5.7: Added `units` parameter.

Changed in version 0.6.3: Added, or documented, optional pre-read parameters: `background`, `colorspace`, `depth`, `format`, `height`, `interlace`, `sampling_factors`, & `width`.

reset_sequence()
Remove any previously allocated *SingleImage* instances in `sequence` attribute.

New in version 0.6.0.

save(*file=None*, *filename=None*, *adjoin=True*)
Saves the image into the `file` or `filename`. It takes only one argument at a time.

Parameters

- **file** (*file object*) – a file object to write to
- **filename** (`basestring`) – a filename string to write to
- **adjoin** (`bool`) – write all images to a single multi-image file. Only available if file format supports frames, layers, & etc.

New in version 0.1.1.

Changed in version 0.1.5: The `file` parameter was added.

Changed in version 6.0.0: The `adjoin` parameter was added.

classmethod stereogram(*left*, *right*)
Create a new stereogram image from two existing images.

See Example of *Stereogram*.

Parameters

- **left** (*wand.image.Image*) – Left-eye image.
- **right** (*wand.image.Image*) – Right-eye image.

New in version 0.5.4.

class wand.image.**ImageProperty**(*image*)
The mixin class to maintain a weak reference to the parent *Image* object.

New in version 0.3.0.

property image
(*Image*) The parent image.

It ensures that the parent *Image*, which is held in a weak reference, still exists. Returns the dereferenced *Image* if it does exist, or raises a *ClosedImageError* otherwise.

Exc *ClosedImageError* when the parent Image has been destroyed

class wand.image.**Iterator**(*image=None*, *iterator=None*)
Row iterator for *Image*. It shouldn't be instantiated directly; instead, it can be acquired through *Image* instance:

```
assert isinstance(image, wand.image.Image)
iterator = iter(image)
```

It doesn't iterate every pixel, but rows. For example:

```
for row in image:
    for col in row:
        assert isinstance(col, wand.color.Color)
        print(col)
```

Every row is a `collections.abc.Sequence` which consists of one or more *wand.color.Color* values.

Parameters **image** (*Image*) – the image to get an iterator

New in version 0.1.3.

clone()
Clones the same iterator.

next(*x=None*)
Return the next item from the iterator. When exhausted, raise StopIteration

wand.image.**KERNEL_INFO_TYPES = ('undefined', 'unity', 'gaussian', 'dog', 'log', 'blur', 'comet', 'binomial', 'laplacian', 'sobel', 'frei_chen', 'roberts', 'prewitt', 'compass', 'kirsch', 'diamond', 'square', 'rectangle', 'octagon', 'disk', 'plus', 'cross', 'ring', 'peaks', 'edges', 'corners', 'diagonals', 'line_ends', 'line_junctions', 'ridges', 'convex_hull', 'thin_se', 'skeleton', 'chebyshev', 'manhattan', 'octagonal', 'euclidean', 'user_defined')**
(tuple) The list of builtin kernels.

- 'undefined'
- 'unity'
- 'gaussian'
- 'dog'
- 'log'
- 'blur'
- 'comet'
- 'laplacian'
- 'sobel'
- 'frei_chen'
- 'roberts'
- 'prewitt'
- 'compass'
- 'kirsch'
- 'diamond'
- 'square'
- 'rectangle'
- 'octagon'
- 'disk'
- 'plus'
- 'cross'
- 'ring'
- 'peaks'
- 'edges'
- 'corners'
- 'diagonals'
- 'line_ends'
- 'line_junctions'
- 'ridges'

- 'convex_hull'
- 'thin_se'
- 'skeleton'
- 'chebyshev'
- 'manhattan'
- 'octagonal'
- 'euclidean'
- 'user_defined'
- 'binomial'

wand.image.**MORPHOLOGY_METHODS = ('undefined', 'convolve', 'correlate', 'erode', 'dilate', 'erode_intensity', 'dilate_intensity', 'iterative_distance', 'open', 'close', 'open_intensity', 'close_intensity', 'smooth', 'edgein', 'edgeout', 'edge', 'tophat', 'bottom_hat', 'hit_and_miss', 'thinning', 'thicken', 'distance', 'voronoi')**

(tuple) The list of morphology methods.

- 'undefined'
- 'convolve'
- 'correlate'
- 'erode'
- 'dilate'
- 'erode_intensity'
- 'dilate_intensity'
- 'distance'
- 'open'
- 'close'
- 'open_intensity'
- 'close_intensity'
- 'smooth'
- 'edgein'
- 'edgeout'
- 'edge'
- 'tophat'
- 'bottom_hat'
- 'hit_and_miss'
- 'thinning'
- 'thicken'
- 'voronoi'
- 'iterative_distance'

class wand.image.**Metadata**(*image*)

Class that implements dict-like read-only access to image metadata like EXIF or IPTC headers. Most WRITE encoders will ignore properties assigned here.

Parameters **image** (*Image*) – an image instance

Note: You don't have to use this by yourself. Use *Image.metadata* property instead.

New in version 0.3.0.

wand.image.**NOISE_TYPES = ('undefined', 'uniform', 'gaussian', 'multiplicative_gaussian', 'impulse', 'laplacian', 'poisson', 'random')**

(tuple) The list of noise types used by *Image.noise()* method.

- 'undefined'
- 'uniform'
- 'gaussian'
- 'multiplicative_gaussian'
- 'impulse'
- 'laplacian'
- 'poisson'
- 'random'

New in version 0.5.3.

wand.image.**ORIENTATION_TYPES = ('undefined', 'top_left', 'top_right', 'bottom_right', 'bottom_left', 'left_top', 'right_top', 'right_bottom', 'left_bottom')**

(tuple) The list of *orientation* types.

New in version 0.3.0.

class wand.image.**OptionDict**(*image*)

Free-form mutable mapping of global internal settings.

New in version 0.3.0.

Changed in version 0.5.0: Remove key check to OPTIONS. Image properties are specific to vendor, and this library should not attempt to manage the 100+ options in a whitelist.

wand.image.**PAPERSIZE_MAP** = {'10x13': (720, 936), '10x14': (720, 1008), '11x17': (792, 1224), '2A0': (3370, 4768), '4A0': (4768, 6741), '4x6': (288, 432), '5x7': (360, 504), '7x9': (504, 648), '8x10': (576, 720), '9x11': (648, 792), '9x12': (648, 864), 'a0': (2384, 3370), 'a1': (1684, 2384), 'a10': (74, 105), 'a2': (1191, 1684), 'a3': (842, 1191), 'a4': (595, 842), 'a4small': (595, 842), 'a5': (420, 595), 'a6': (298, 420), 'a7': (210, 298), 'a8': (147, 210), 'a9': (105, 147), 'archC': (1296, 1728), 'archa': (648, 864), 'archb': (864, 1296), 'archd': (1728, 2592), 'arche': (2592, 3456), 'b0': (2920, 4127), 'b1': (2064, 2920), 'b10': (91, 127), 'b2': (1460, 2064), 'b3': (1032, 1460), 'b4': (729, 1032), 'b5': (516, 729), 'b6': (363, 516), 'b7': (258, 363), 'b8': (181, 258), 'b9': (127, 181), 'c0': (2599, 3676), 'c1': (1837, 2599), 'c2': (1298, 1837), 'c3': (918, 1296), 'c4': (649, 918), 'c5': (459, 649), 'c6': (323, 459), 'c7': (230, 323), 'csheet': (1224, 1584), 'dsheet': (1584, 2448), 'esheet': (2448, 3168), 'executive': (540, 720), 'flsa': (612, 936), 'flse': (612, 936), 'folio': (612, 936), 'halfletter': (396, 612), 'isob0': (2835, 4008), 'isob1': (2004, 2835), 'isob10': (88, 125), 'isob2': (1417, 2004), 'isob3': (1001, 1417), 'isob4': (709, 1001), 'isob5': (499, 709), 'isob6': (354, 499), 'isob7': (249, 354), 'isob8': (176, 249), 'isob9': (125, 176), 'jisb0': (1030, 1456), 'jisb1': (728, 1030), 'jisb2': (515, 728), 'jisb3': (364, 515), 'jisb4': (257, 364), 'jisb5': (182, 257), 'jisb6': (128, 182), 'ledger': (1224, 792), 'legal': (612, 1008), 'letter': (612, 792), 'lettersmall': (612, 792), 'monarch': (279, 540), 'quarto': (610, 780), 'statement': (396, 612), 'tabloid': (792, 1224)}

(dict) Map of papersize names to page sizes. Each page size is a width & height tuple at a 72dpi resolution.

```
from wand.image import Image, PAPERSIZE_MAP

w, h = PAPERSIZE_MAP["a4"]
with Image(width=w, height=h, background="white") as img:
    img.save(filename="a4_page.png")
```

New in version 0.6.4.

wand.image.**PIXEL_INTERPOLATE_METHODS** = ('undefined', 'average', 'average9', 'average16', 'background', 'bilinear', 'blend', 'catrom', 'integer', 'mesh', 'nearest', 'spline')

(tuple) List of interpolate pixel methods (ImageMagick-7 only.)

- 'undefined'
- 'average'
- 'average9'
- 'average16'
- 'background'
- 'bilinear'
- 'blend'
- 'catrom'
- 'integer'
- 'mesh'
- 'nearest'
- 'spline'

New in version 0.5.0.

class wand.image.**ProfileDict**(*image*)

The mapping table of embedded image profiles.

Use this to get, set, and delete whole profile payloads on an image. Each payload is a raw binary string.

For example:

```
with Image(filename='photo.jpg') as img:
    # Extract EXIF
    with open('exif.bin', 'wb') as payload:
        payload.write(img.profiles['exif'])
    # Import ICC
    with open('color_profile.icc', 'rb') as payload:
        img.profiles['icc'] = payload.read()
    # Remove XMP
    del imp.profiles['xmp']
```

See also:

Embedded Image Profiles for a list of supported profiles.

New in version 0.5.1.

wand.image.**RENDERING_INTENT_TYPES = ('undefined', 'saturation', 'perceptual', 'absolute', 'relative')**

(tuple) List of rendering intent types used for *Image.rendering_intent* property.

- 'undefined'
- 'saturation'
- 'perceptual'
- 'absolute'
- 'relative'

New in version 0.5.4.

wand.image.**SPARSE_COLOR_METHODS = {'barycentric': 1, 'bilinear': 7, 'inverse': 19, 'manhattan': 20, 'shepards': 16, 'undefined': 0, 'voronoi': 18}**

(tuple) List of sparse color methods used by *Image.sparse_color()*

- 'undefined'
- 'barycentric'
- 'bilinear'
- 'shepards'
- 'voronoi'
- 'inverse'
- 'manhattan'

New in version 0.5.3.

wand.image.**STATISTIC_TYPES = ('undefined', 'gradient', 'maximum', 'mean', 'median', 'minimum', 'mode', 'nonpeak', 'root_mean_square', 'standard_deviation')**

(tuple) The list of statistic types used by *Image.statistic()*.

- 'undefined'
- 'gradient'

- 'maximum'
- 'mean'
- 'median'
- 'minimum'
- 'mode'
- 'nonpeak'
- 'root_mean_square'
- 'standard_deviation'

New in version 0.5.3.

wand.image.**STORAGE_TYPES = ('undefined', 'char', 'double', 'float', 'integer', 'long', 'quantum', 'short')**

(tuple) The list of pixel storage types.

- 'undefined'
- 'char'
- 'double'
- 'float'
- 'integer'
- 'long'
- 'quantum'
- 'short'

New in version 0.5.0.

wand.image.**UNIT_TYPES = ('undefined', 'pixelsperinch', 'pixelspercentimeter')**

(tuple) The list of resolution unit types.

- 'undefined'
- 'pixelsperinch'
- 'pixelspercentimeter'

See also:

ImageMagick Image Units Describes the MagickSetImageUnits method which can be used to set image units of resolution

wand.image.**VIRTUAL_PIXEL_METHOD = ('undefined', 'background', 'dither', 'edge', 'mirror', 'random', 'tile', 'transparent', 'mask', 'black', 'gray', 'white', 'horizontal_tile', 'vertical_tile', 'horizontal_tile_edge', 'vertical_tile_edge', 'checker_tile')**

(tuple) The list of *virtual_pixel* types.

- 'undefined'
- 'background'
- 'constant' - Only available with ImageMagick-6
- 'dither'
- 'edge'

- 'mirror'
- 'random'
- 'tile'
- 'transparent'
- 'mask'
- 'black'
- 'gray'
- 'white'
- 'horizontal_tile'
- 'vertical_tile'
- 'horizontal_tile_edge'
- 'vertical_tile_edge'
- 'checker_tile'

New in version 0.4.1.

wand.image.**manipulative**(*function*)
Mark the operation manipulating itself instead of returning new one.

4.1.2 wand.color — Colors

New in version 0.1.2.

class wand.color.**Color**(*string=None*, *raw=None*)
Color value.

Unlike any other objects in Wand, its resource management can be implicit when it used outside of `with` block. In these case, its resource are allocated for every operation which requires a resource and destroyed immediately. Of course it is inefficient when the operations are much, so to avoid it, you should use color objects inside of `with` block explicitly e.g.:

```
red_count = 0
with Color('#f00') as red:
    with Image(filename='image.png') as img:
        for row in img:
            for col in row:
                if col == red:
                    red_count += 1
```

Parameters **string** (basestring) – a color name string e.g. 'rgb(255, 255, 255)', '#fff', 'white'. see ImageMagick Color Names doc also

Changed in version 0.3.0: *Color* objects become hashable.

Changed in version 0.5.1: Color channel properties can now be set.

Changed in version 0.5.1: Added *cyan*, *magenta*, *yellow*, & *black* properties for CMYK *Color* instances.

Changed in version 0.5.1: Method *Color.from_hsl()* can create a RGB color from hue, saturation, & lightness values.

See also:

ImageMagick Color Names The color can then be given as a color name (there is a limited but large set of these; see below) or it can be given as a set of numbers (in decimal or hexadecimal), each corresponding to a channel in an RGB or RGBA color model. HSL, HSLA, HSB, HSBA, CMYK, or CMYKA color models may also be specified. These topics are briefly described in the sections below.

`== (other)`
Equality operator.

Param other a color another one

Type color *Color*

Returns `True` only if two images equal.

Rtype `bool`

`property alpha`
(`numbers.Real`) Alpha value, from 0.0 to 1.0.

`property alpha_int8`
(`numbers.Integral`) Alpha value as 8bit integer which is a common style. From 0 to 255.

New in version 0.3.0.

`property alpha_quantum`
(`numbers.Integral`) Alpha value. Scale depends on *QUANTUM_DEPTH*.

New in version 0.3.0.

`property black`
(`numbers.Real`) Black, or `'K'`, color channel in CMYK colorspace. Unused by RGB colorspace.

New in version 0.5.1.

`property black_int8`
(`numbers.Integral`) Black value as 8bit integer which is a common style. From 0 to 255.

New in version 0.5.1.

`property black_quantum`
(`numbers.Integral`) Black. Scale depends on *QUANTUM_DEPTH*.

New in version 0.5.1.

`property blue`
(`numbers.Real`) Blue, from 0.0 to 1.0.

`property blue_int8`
(`numbers.Integral`) Blue as 8bit integer which is a common style. From 0 to 255.

New in version 0.3.0.

`property blue_quantum`
(`numbers.Integral`) Blue. Scale depends on *QUANTUM_DEPTH*.

New in version 0.3.0.

`static c_equals`(*a*, *b*)
Raw level version of equality test function for two pixels.

Parameters

- **a** (`ctypes.c_void_p`) – a pointer to PixelWand to compare

- **b** (ctypes.c_void_p) – a pointer to PixelWand to compare

Returns True only if two pixels equal

Return type bool

Note: It's only for internal use. Don't use it directly. Use == operator of *Color* instead.

---

property cyan
(numbers.Real) Cyan color channel in CMYK colorspace. Unused by RGB colorspace.

New in version 0.5.1.

property cyan_int8
(numbers.Integral) Cyan value as 8bit integer which is a common style. From 0 to 255.

New in version 0.5.1.

property cyan_quantum
(numbers.Integral) Cyan. Scale depends on *QUANTUM_DEPTH*.

New in version 0.5.1.

dirty = None
(bool) Whether the color has changed or not.

classmethod from_hsl(*hue=0.0*, *saturation=0.0*, *lightness=0.0*)
Creates a RGB color from HSL values. The hue, saturation, and lightness must be normalized between 0.0 & 1.0.

```
h=0.75  # 270 Degrees
s=1.0   # 100 Percent
l=0.5   # 50 Percent
with Color.from_hsl(hue=h, saturation=s, lightness=l) as color:
    print(color)  #=> srgb(128,0,255)
```

Parameters

- **hue** (numbers.Real) – a normalized double between 0.0 & 1.0.
- **saturation** (numbers.Real) – a normalized double between 0.0 & 1.0.
- **lightness** (numbers.Real) – a normalized double between 0.0 & 1.0.

Return type *Color*

New in version 0.5.1.

property green
(numbers.Real) Green, from 0.0 to 1.0.

property green_int8
(numbers.Integral) Green as 8bit integer which is a common style. From 0 to 255.

New in version 0.3.0.

property green_quantum
(numbers.Integral) Green. Scale depends on *QUANTUM_DEPTH*.

New in version 0.3.0.

hsl()
Calculate the HSL color values from the RGB color.

Returns Tuple containing three normalized doubles, between 0.0 & 1.0, representing `hue`, `saturation`, and `lightness`.

Return type `collections.Sequence`

New in version 0.5.1.

property magenta
(`numbers.Real`) Magenta color channel in CMYK colorspace. Unused by RGB colorspace.

New in version 0.5.1.

property magenta_int8
(`numbers.Integral`) Magenta value as 8bit integer which is a common style. From 0 to 255.

New in version 0.5.1.

property normalized_string
(`basestring`) The normalized string representation of the color. The same color is always represented to the same string.

New in version 0.3.0.

property red
(`numbers.Real`) Red, from 0.0 to 1.0.

property red_int8
(`numbers.Integral`) Red as 8bit integer which is a common style. From 0 to 255.

New in version 0.3.0.

property red_quantum
(`numbers.Integral`) Red. Scale depends on *QUANTUM_DEPTH*.

New in version 0.3.0.

property string
(`basestring`) The string representation of the color.

property yellow
(`numbers.Real`) Yellow color channel in CMYK colorspace. Unused by RGB colorspace.

New in version 0.5.1.

property yellow_int8
(`numbers.Integral`) Yellow as 8bit integer which is a common style. From 0 to 255.

New in version 0.5.1.

property yellow_quantum
(`numbers.Integral`) Yellow. Scale depends on *QUANTUM_DEPTH*.

New in version 0.5.1.

wand.color.**scale_quantum_to_int8**(*quantum*)
Straightforward port of `ScaleQuantumToChar()` inline function.

Deprecated since version 0.6.6.

Parameters **quantum** (`numbers.Integral`) – quantum value

Returns 8bit integer of the given `quantum` value

Return type `numbers.Integral`

New in version 0.3.0.

Changed in version 0.5.0: Added HDRI support

4.1.3 wand.font — Fonts

New in version 0.3.0.

Font is an object which takes the *path* of font file, *size*, *color*, and whether to use *antialias*ing. If you want to use font by its name rather than the file path, use TTFQuery package. The font path resolution by its name is a very complicated problem to achieve.

See also:

TTFQuery — Find and Extract Information from TTF Files TTFQuery builds on the FontTools-TTX package to allow the Python programmer to accomplish a number of tasks:

- query the system to find installed fonts
- retrieve metadata about any TTF font file
  - this includes the glyph outlines (shape) of individual code-points, which allows for rendering the glyphs in 3D (such as is done in OpenGLContext)
- lookup/find fonts by:
  - abstract family type
  - proper font name
- build simple metadata registries for run-time font matching

class wand.font.**Font**(*path*, *size=0*, *color=None*, *antialias=True*, *stroke_color=None*, *stroke_width=None*)

Font struct which is a subtype of `tuple`.

Parameters

- **path** (`str`, `basestring`) – the path of the font file
- **size** (`numbers.Real`) – the size of typeface. 0 by default which means *autosized*
- **color** (*Color*) – the color of typeface. black by default
- **antialias** (`bool`) – whether to use antialiasing. `True` by default
- **stroke_color** (*Color*) – optional color to outline typeface.
- **stroke_width** (`numbers.Real`) – optional thickness of typeface outline.

Changed in version 0.3.9: The `size` parameter becomes optional. Its default value is 0, which means *autosized*.

Changed in version 0.5.0: Added `stroke_color` & `stoke_width` parameters.

property antialias

(`bool`) Whether to apply antialiasing (`True`) or not (`False`).

property color

(*wand.color.Color*) The font color.

property path

(`basestring`) The path of font file.

property size

(`numbers.Real`) The font size in pixels.

property stroke_color
(*wand.color.Color*) The stroke color.

property stroke_width
(numbers.Real) The width of the stroke line.

4.1.4 wand.drawing — Drawings

The module provides some vector drawing functions.

New in version 0.3.0.

wand.drawing.**CLIP_PATH_UNITS = ('undefined_path_units', 'user_space', 'user_space_on_use', 'object_bounding_box')**
(collections.abc.Sequence) The list of clip path units

- 'undefined_path_units'
- 'user_space'
- 'user_space_on_use'
- 'object_bounding_box'

class wand.drawing.**Drawing**(*drawing=None*)
Drawing object. It maintains several vector drawing instructions and can get drawn into zero or more *Image* objects by calling it.

For example, the following code draws a diagonal line to the image:

```
with Drawing() as draw:
    draw.line((0, 0), image.size)
    draw(image)
```

Parameters **drawing** (*Drawing*) – an optional drawing object to clone. use *clone()* method rather than this parameter

New in version 0.3.0.

affine(*matrix*)
Adjusts the current affine transformation matrix with the specified affine transformation matrix. Note that the current affine transform is adjusted rather than replaced.

```
                                  | sx  rx  0 |
| x', y', 1 |  =  | x, y, 1 |  *  | ry  sy  0 |
                                  | tx  ty  1 |
```

Parameters **matrix** (collections.abc.Sequence) – a list of Real to define affine matrix [sx, rx, ry, sy, tx, ty]

New in version 0.4.0.

alpha(*x=None*, *y=None*, *paint_method='undefined'*)
Paints on the image's opacity channel in order to set effected pixels to transparent.

To influence the opacity of pixels. The available methods are:

- 'undefined'

- 'point'
- 'replace'
- 'floodfill'
- 'filltoborder'
- 'reset'

Note: This method replaces *matte()* in ImageMagick version 7. An AttributeError will be raised if attempting to call on a library without DrawAlpha support.

New in version 0.5.0.

arc(*start*, *end*, *degree*)
Draws a arc using the current *stroke_color*, *stroke_width*, and *fill_color*.

Parameters

- **start** (Sequence) – (Real, numbers.Real) pair which represents starting x and y of the arc
- **end** (Sequence) – (Real, numbers.Real) pair which represents ending x and y of the arc
- **degree** (Sequence) – (Real, numbers.Real) pair which represents starting degree, and ending degree

New in version 0.4.0.

bezier(*points=None*)
Draws a bezier curve through a set of points on the image, using the specified array of coordinates.

At least four points should be given to complete a bezier path. The first & forth point being the start & end point, and the second & third point controlling the direction & curve.

Example bezier on image

```
with Drawing() as draw:
    points = [(40,10), # Start point
              (20,50), # First control
              (90,10), # Second control
              (70,40)] # End point
    draw.stroke_color = Color('#000')
    draw.fill_color = Color('#fff')
    draw.bezier(points)
    draw.draw(image)
```

Parameters **points** (list) – list of x,y tuples

New in version 0.4.0.

property border_color
(*Color*) the current border color. It also can be set. This attribute controls the behavior of *color()* during 'filltoborder' operation.

New in version 0.4.0.

circle(*origin*, *perimeter*)
Draws a circle from origin to perimeter

Parameters

- **origin** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents origin x and y of circle
- **perimeter** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents perimeter x and y of circle

New in version 0.4.0.

property clip_path
(basestring) The current clip path. It also can be set.

New in version 0.4.0.

property clip_rule
(basestring) The current clip rule. It also can be set. It's a string value from *FILL_RULE_TYPES* list.

New in version 0.4.0.

property clip_units
(basestring) The current clip units. It also can be set. It's a string value from *CLIP_PATH_UNITS* list.

New in version 0.4.0.

clone()
Copies a drawing object.

Returns a duplication

Return type *Drawing*

color(*x=0.0*, *y=0.0*, *paint_method='undefined'*)
Draws a color on the image using current fill color, starting at specified position & method.

Available methods in *wand.drawing.PAINT_METHOD_TYPES*:

- 'undefined'
- 'point'
- 'replace'
- 'floodfill'
- 'filltoborder'
- 'reset'

New in version 0.4.0.

comment(*message=None*)
Adds a comment to the vector stream.

Parameters **message** (basestring) – the comment to set.

New in version 0.4.0.

composite(*operator*, *left*, *top*, *width*, *height*, *image*)
Composites an image onto the current image, using the specified composition operator, specified position, and at the specified size.

Parameters

- **operator** – the operator that affects how the composite is applied to the image. available values can be found in the COMPOSITE_OPERATORS list
- **type** – COMPOSITE_OPERATORS

- **left** (numbers.Real) – the column offset of the composited drawing source
- **top** (numbers.Real) – the row offset of the composited drawing source
- **width** (numbers.Real) – the total columns to include in the composited source
- **height** (numbers.Real) – the total rows to include in the composited source

New in version 0.4.0.

draw(*image*)

Renders the current drawing into the `image`. You can simply call *Drawing* instance rather than calling this method. That means the following code which calls *Drawing* object itself:

```
drawing(image)
```

is equivalent to the following code which calls *draw()* method:

```
drawing.draw(image)
```

Parameters **image** (*BaseImage*) – the image to be drawn

ellipse(*origin*, *radius*, *rotation=None*)

Draws a ellipse at `origin` with independent x & y `radius`. Ellipse can be partial by setting start & end `rotation`.

Parameters

- **origin** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents origin x and y of circle
- **radius** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents radius x and radius y of circle
- **rotation** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents start and end of ellipse. Default (0,360)

New in version 0.4.0.

property fill_color

(*Color*) The current color to fill. It also can be set.

property fill_opacity

(Real) The current fill opacity. It also can be set.

New in version 0.4.0.

property fill_rule

(basestring) The current fill rule. It can also be set. It's a string value from *FILL_RULE_TYPES* list.

New in version 0.4.0.

property font

(basestring) The current font name. It also can be set.

property font_family

(basestring) The current font family. It also can be set.

New in version 0.4.0.

property font_resolution

(Sequence) The current font resolution. It also can be set.

New in version 0.4.0.

property font_size
(numbers.Real) The font size. It also can be set.

property font_stretch
(basestring) The current font stretch variation. It also can be set, but will only apply if the font-family or encoder supports the stretch type.

New in version 0.4.0.

property font_style
(basestring) The current font style. It also can be set, but will only apply if the font-family supports the style.

New in version 0.4.0.

property font_weight
(Integral) The current font weight. It also can be set.

New in version 0.4.0.

get_font_metrics(*image*, *text*, *multiline=False*)
Queries font metrics by cloning the image, and rendering the text with the font properties defined on *Drawing*.

This is useful for determining the size of the rendered text. Set multiline to True if the given text contains line-breaks.

The return value is an instance of *FontMetrics* which has the attributes of the text's rendered size, max horizontal advance, and distance of ascent & descent from the baseline.

Parameters

- **image** (*BaseImage*) – the image to be drawn
- **text** (basestring) – the text string for get font metrics.
- **multiline** (*boolean*) – text is multiline or not

Returns *FontMetrics*

New in version 0.3.0.

property gravity
(basestring) The text placement gravity used when annotating with text. It's a string from *GRAVITY_TYPES* list. It also can be set.

line(*start*, *end*)
Draws a line start to end.

Parameters

- **start** (collections.abc.Sequence) – (Integral, numbers.Integral) pair which represents starting x and y of the line
- **end** (collections.abc.Sequence) – (Integral, numbers.Integral) pair which represents ending x and y of the line

matte(*x=0.0*, *y=0.0*, *paint_method='undefined'*)
Paints on the image's opacity channel in order to set effected pixels to transparent.

To influence the opacity of pixels. The available methods are:

- 'undefined'
- 'point'

- 'replace'
- 'floodfill'
- 'filltoborder'
- 'reset'

Note: This method has been replace by *alpha()* in ImageMagick version 7. An AttributeError will be raised if attempting to call on a library without DrawMatte support.

New in version 0.4.0.

property opacity
(Real) returns the opacity used when drawing with the fill or stroke color or texture. Fully opaque is 1.0. This method only affects vector graphics, and is experimental. To set the opacity of a drawing, use *Drawing.fill_opacity* & *Drawing.stroke_opacity*

New in version 0.4.0.

path_close()
Adds a path element to the current path which closes the current subpath by drawing a straight line from the current point to the current subpath's most recent starting point.

New in version 0.4.0.

path_curve(*to=None*, *controls=None*, *smooth=False*, *relative=False*)
Draws a cubic Bezier curve from the current point to given to (x,y) coordinate using controls points at the beginning and the end of the curve. If smooth is set to True, only one controls is expected and the previous control is used, else two pair of coordinates are expected to define the control points. The to coordinate then becomes the new current point.

Parameters

- **to** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents coordinates to draw to
- **controls** (collections.abc.Sequence) – (Real, numbers.Real) coordinate to used to influence curve
- **smooth** (bool) – bool assume last defined control coordinate
- **relative** (bool) – treat given coordinates as relative to current point

New in version 0.4.0.

path_curve_to_quadratic_bezier(*to=None*, *control=None*, *smooth=False*, *relative=False*)
Draws a quadratic Bezier curve from the current point to given to coordinate. The control point is assumed to be the reflection of the control point on the previous command if smooth is True, else a pair of control coordinates must be given. Each coordinates can be relative, or absolute, to the current point by setting the relative flag. The to coordinate then becomes the new current point, and the control coordinate will be assumed when called again when smooth is set to true.

Parameters

- **to** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents coordinates to draw to
- **control** (collections.abc.Sequence) – (Real, numbers.Real) coordinate to used to influence curve
- **smooth** (bool) – assume last defined control coordinate

- **relative** (bool) – treat given coordinates as relative to current point

New in version 0.4.0.

path_elliptic_arc(*to=None*, *radius=None*, *rotation=0.0*, *large_arc=False*, *clockwise=False*, *relative=False*)

Draws an elliptical arc from the current point to given `to` coordinates. The `to` coordinates can be relative, or absolute, to the current point by setting the `relative` flag. The size and orientation of the ellipse are defined by two radii (rx, ry) in `radius` and an `rotation` parameters, which indicates how the ellipse as a whole is rotated relative to the current coordinate system. The center of the ellipse is calculated automagically to satisfy the constraints imposed by the other parameters. `large_arc` and `clockwise` contribute to the automatic calculations and help determine how the arc is drawn. If `large_arc` is True then draw the larger of the available arcs. If `clockwise` is true, then draw the arc matching a clock-wise rotation.

Parameters

- **to** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents coordinates to draw to
- **radius** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents the radii of the ellipse to draw
- **rotate** (Real) – degree to rotate ellipse on x-axis
- **large_arc** (bool) – draw largest available arc
- **clockwise** (bool) – draw arc path clockwise from start to target
- **relative** (bool) – treat given coordinates as relative to current point

New in version 0.4.0.

path_finish()

Terminates the current path.

New in version 0.4.0.

path_horizontal_line(*x=None*, *relative=False*)

Draws a horizontal line path from the current point to the target point. Given `x` parameter can be relative, or absolute, to the current point by setting the `relative` flag. The target point then becomes the new current point.

Parameters

- **x** (Real) – Real x-axis point to draw to.
- **relative** (bool) – bool treat given point as relative to current point

New in version 0.4.0.

path_line(*to=None*, *relative=False*)

Draws a line path from the current point to the given `to` coordinate. The `to` coordinates can be relative, or absolute, to the current point by setting the `relative` flag. The coordinate then becomes the new current point.

Parameters

- **to** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents coordinates to draw to.
- **relative** (bool) – bool treat given coordinates as relative to current point

New in version 0.4.0.

path_move(*to=None*, *relative=False*)
Starts a new sub-path at the given coordinates. Given `to` parameter can be relative, or absolute, by setting the `relative` flag.

Parameters

- **to** (collections.abc.Sequence) – (Real, numbers.Real) pair which represents coordinates to draw to.
- **relative** (bool) – bool treat given coordinates as relative to current point

New in version 0.4.0.

path_start()
Declares the start of a path drawing list which is terminated by a matching *path_finish()* command. All other *path_** commands must be enclosed between a *path_start()* and a *path_finish()* command. This is because path drawing commands are subordinate commands and they do not function by themselves.

New in version 0.4.0.

path_vertical_line(*y=None*, *relative=False*)
Draws a vertical line path from the current point to the target point. Given `y` parameter can be relative, or absolute, to the current point by setting the `relative` flag. The target point then becomes the new current point.

Parameters

- **y** (Real) – Real y-axis point to draw to.
- **relative** (bool) – bool treat given point as relative to current point

New in version 0.4.0.

point(*x*, *y*)
Draws a point at given `x` and `y`

Parameters

- **x** (Real) – Real x of point
- **y** (Real) – Real y of point

New in version 0.4.0.

polygon(*points=None*)
Draws a polygon using the current *stroke_color*, *stroke_width*, and *fill_color*, using the specified array of coordinates.

Example polygon on `image`

```
with Drawing() as draw:
    points = [(40,10), (20,50), (90,10), (70,40)]
    draw.polygon(points)
    draw.draw(image)
```

Parameters **points** (list) – list of x,y tuples

New in version 0.4.0.

polyline(*points=None*)
Draws a polyline using the current *stroke_color*, *stroke_width*, and *fill_color*, using the specified array of coordinates.

Identical to *polygon*, but without closed stroke line.

Parameters **points** (list) – list of x,y tuples

New in version 0.4.0.

pop()
Pop destroys the current tip of the drawing context stack, and restores the parent style context. See *push()* method for an example.

Note: Popping the graphical context stack will not erase, or alter, any previously executed drawing commands.

Returns success of pop operation.

Return type bool

New in version 0.4.0.

pop_clip_path()
Terminates a clip path definition.

New in version 0.4.0.

pop_defs()
Terminates a definition list.

New in version 0.4.0.

pop_pattern()
Terminates a pattern definition.

New in version 0.4.0.

push()
Grows the current drawing context stack by one, and inherits the previous style attributes. Use *Drawing.pop* to return to restore previous style attributes.

This is useful for drawing shapes with diffrent styles without repeatedly setting the similar *fill_color* & *stroke_color* properties.

For example:

```
with Drawing() as ctx:
    ctx.fill_color = Color('GREEN')
    ctx.stroke_color = Color('ORANGE')
    ctx.push()
    ctx.fill_color = Color('RED')
    ctx.text(x1, y1, 'this is RED with ORANGE outline')
    ctx.push()
    ctx.stroke_color = Color('BLACK')
    ctx.text(x2, y2, 'this is RED with BLACK outline')
    ctx.pop()
    ctx.pop()
    ctx.text(x3, y3, 'this is GREEN with ORANGE outline')
```

Which translate to the following MVG:

```
push graphic-context
    fill "GREEN"
```

(continues on next page)

(continued from previous page)

```
    stroke "ORANGE"
    push graphic-context
        fill "RED"
        text x1,y1 "this is RED with ORANGE outline"
        push graphic-context
            stroke "BLACK"
            text x2,y2 "this is RED with BLACK outline"
        pop graphic-context
    pop graphic-context
    text x3,y3 "this is GREEN with ORANGE outline"
pop graphic-context
```

Note: Pushing graphical context does not reset any previously drawn artifacts.

Returns success of push operation.

Return type bool

New in version 0.4.0.

push_clip_path(*clip_mask_id*)

Starts a clip path definition which is comprised of any number of drawing commands and terminated by a *Drawing.pop_clip_path* command.

Parameters **clip_mask_id** (basestring) – string identifier to associate with the clip path.

New in version 0.4.0.

push_defs()

Indicates that commands up to a terminating *Drawing.pop_defs* command create named elements (e.g. clip-paths, textures, etc.) which may safely be processed earlier for the sake of efficiency.

New in version 0.4.0.

push_pattern(*pattern_id*, *left*, *top*, *width*, *height*)

Indicates that subsequent commands up to a *Drawing.pop_pattern* command comprise the definition of a named pattern. The pattern space is assigned top left corner coordinates, a width and height, and becomes its own drawing space. Anything which can be drawn may be used in a pattern definition. Named patterns may be used as stroke or brush definitions.

Parameters

- **pattern_id** (basestring) – a unique identifier for the pattern.
- **left** (numbers.Real) – x ordinate of top left corner.
- **top** (numbers.Real) – y ordinate of top left corner.
- **width** (numbers.Real) – width of pattern space.
- **height** (numbers.Real) – height of pattern space.

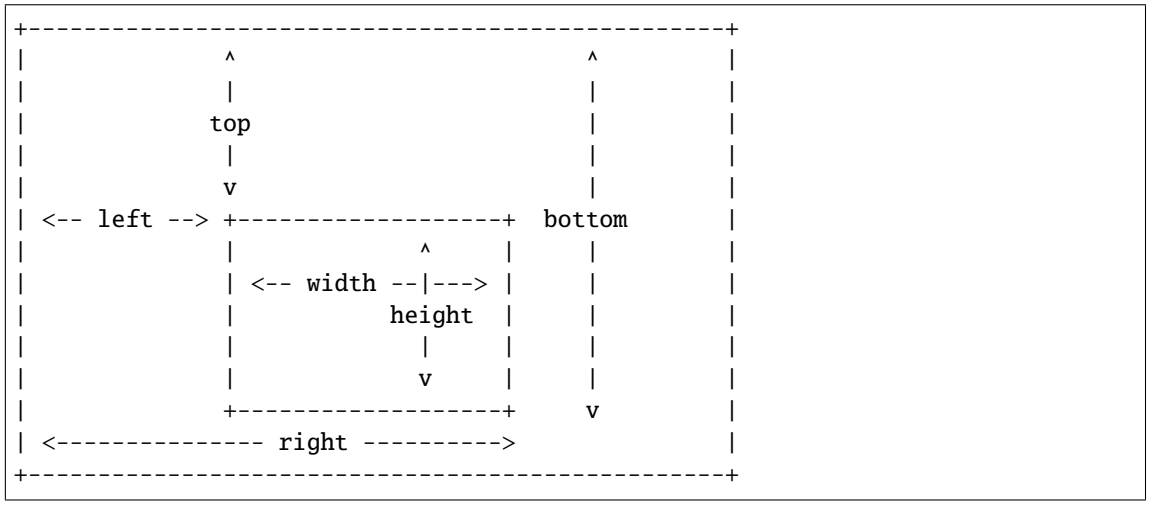
Returns success of push operation

Return type bool

New in version 0.4.0.

rectangle(*left=None*, *top=None*, *right=None*, *bottom=None*, *width=None*, *height=None*, *radius=None*, *xradius=None*, *yradius=None*)

Draws a rectangle using the current *stroke_color*, *stroke_width*, and *fill_color*.



Parameters

- **left** (numbers.Real) – x-offset of the rectangle to draw
- **top** (numbers.Real) – y-offset of the rectangle to draw
- **right** (numbers.Real) – second x-offset of the rectangle to draw. this parameter and `width` parameter are exclusive each other
- **bottom** (numbers.Real) – second y-offset of the rectangle to draw. this parameter and `height` parameter are exclusive each other
- **width** (numbers.Real) – the `width` of the rectangle to draw. this parameter and `right` parameter are exclusive each other
- **height** (numbers.Real) – the `height` of the rectangle to draw. this parameter and `bottom` parameter are exclusive each other
- **radius** (numbers.Real) – the corner rounding. this is a short-cut for setting both `xradius`, and `yradius`
- **xradius** (numbers.Real) – the `xradius` corner in horizontal direction.
- **yradius** (numbers.Real) – the `yradius` corner in vertical direction.

New in version 0.3.6.

Changed in version 0.4.0: Radius keywords added to create rounded rectangle.

rotate(*degree=0.0*)

Applies the specified rotation to the current coordinate space.

Parameters **degree** (Real) – degree to rotate

New in version 0.4.0.

scale(*x=1.0*, *y=1.0*)

Adjusts the scaling factor to apply in the horizontal and vertical directions to the current coordinate space.

Parameters

- **x** (Real) – Horizontal scale factor. Default *1.0*

- **y** (Real) – Vertical scale factor. Default *1.0*

New in version 0.4.0.

set_fill_pattern_url(*url*)
Sets the URL to use as a fill pattern for filling objects. Only local URLs ("#identifier") are supported at this time. These local URLs are normally created by defining a named fill pattern with Drawing.push_pattern & Drawing.pop_pattern.

Parameters **url** (basestring) – URL to use to obtain fill pattern.

New in version 0.4.0.

set_stroke_pattern_url(*url*)
Sets the pattern used for stroking object outlines. Only local URLs ("#identifier") are supported at this time. These local URLs are normally created by defining a named stroke pattern with Drawing.push_pattern & Drawing.pop_pattern.

Parameters **url** (basestring) – URL to use to obtain stroke pattern.

New in version 0.4.0.

skew(*x=None*, *y=None*)
Skews the current coordinate system in the horizontal direction if **x** is given, and vertical direction if **y** is given.

Parameters

- **x** (Real) – Skew horizontal direction
- **y** (Real) – Skew vertical direction

New in version 0.4.0.

property stroke_antialias
(bool) Controls whether stroked outlines are antialiased. Stroked outlines are antialiased by default. When antialiasing is disabled stroked pixels are thresholded to determine if the stroke color or underlying canvas color should be used.

It also can be set.

New in version 0.4.0.

property stroke_color
(*Color*) The current color of stroke. It also can be set.

New in version 0.3.3.

property stroke_dash_array
(Sequence) - (numbers.Real) An array representing the pattern of dashes & gaps used to stroke paths. It also can be set.

New in version 0.4.0.

property stroke_dash_offset
(numbers.Real) The stroke dash offset. It also can be set.

New in version 0.4.0.

property stroke_line_cap
(basestring) The stroke line cap. It also can be set.

New in version 0.4.0.

property stroke_line_join
(basestring) The stroke line join. It also can be set.

New in version 0.4.0.

property stroke_miter_limit
(Integral) The current miter limit. It also can be set.

New in version 0.4.0.

property stroke_opacity
(Real) The current stroke opacity. It also can be set.

New in version 0.4.0.

property stroke_width
(numbers.Real) The stroke width. It also can be set.

New in version 0.3.3.

text(*x*, *y*, *body*)
Writes a text body into (x, y).

Parameters

- **x** (numbers.Integral) – the left offset where to start writing a text
- **y** (numbers.Integral) – the baseline where to start writing text
- **body** (basestring) – the body string to write

property text_alignment
(basestring) The current text alignment setting. It's a string value from *TEXT_ALIGN_TYPES* list. It also can be set.

property text_antialias
(bool) The boolean value which represents whether antialiasing is used for text rendering. It also can be set to True or False to switch the setting.

property text_decoration
(basestring) The text decoration setting, a string from *TEXT_DECORATION_TYPES* list. It also can be set.

property text_direction
(basestring) The text direction setting. a string from *TEXT_DIRECTION_TYPES* list. It also can be set.

property text_encoding
(basestring) The internally used text encoding setting. Although it also can be set, but it's not encouraged.

property text_interline_spacing
(numbers.Real) The setting of the text line spacing. It also can be set.

property text_interword_spacing
(numbers.Real) The setting of the word spacing. It also can be set.

property text_kerning
(numbers.Real) The setting of the text kerning. It also can be set.

property text_under_color
(*Color*) The color of a background rectangle to place under text annotations. It also can be set.

translate(*x=0.0*, *y=0.0*)
Applies a translation to the current coordinate system which moves the coordinate system origin to the specified coordinate.

Parameters

- **x** (Real) – Skew horizontal direction
- **y** (Real) – Skew vertical direction

New in version 0.4.0.

property vector_graphics
(basestring) The XML text of the Vector Graphics. It also can be set. The drawing-wand XML is experimental, and subject to change.

Setting this property to None will reset all vector graphic properties to the default state.

New in version 0.4.0.

viewbox(*left*, *top*, *right*, *bottom*)
Viewbox sets the overall canvas size to be recorded with the drawing vector data. Usually this will be specified using the same size as the canvas image. When the vector data is saved to SVG or MVG formats, the viewbox is use to specify the size of the canvas image that a viewer will render the vector data on.

Parameters

- **left** (Integral) – the left most point of the viewbox.
- **top** (Integral) – the top most point of the viewbox.
- **right** (Integral) – the right most point of the viewbox.
- **bottom** (Integral) – the bottom most point of the viewbox.

New in version 0.4.0.

wand.drawing.**FILL_RULE_TYPES = ('undefined', 'evenodd', 'nonzero')**
(collections.abc.Sequence) The list of fill-rule types.

- 'undefined'
- 'evenodd'
- 'nonzero'

wand.drawing.**FONT_METRICS_ATTRIBUTES = ('character_width', 'character_height', 'ascender', 'descender', 'text_width', 'text_height', 'maximum_horizontal_advance', 'x1', 'y1', 'x2', 'y2', 'x', 'y')**
(collections.abc.Sequence) The attribute names of font metrics.

class wand.drawing.**FontMetrics**(*character_width*, *character_height*, *ascender*, *descender*, *text_width*, *text_height*, *maximum_horizontal_advance*, *x1*, *y1*, *x2*, *y2*, *x*, *y*)
The tuple subtype which consists of font metrics data.

Pixel values can be converted to points by the following:

$$points = \frac{pixels * 72}{resolution}$$

character_width
(numbers.Real) The horizontal value of the **x** in pixels.

character_height
(numbers.Real) The vertical value of the **x** in pixels.

ascender
(numbers.Real) The largest distance from the text baseline to the highest point above the x-height. This value should always be positive.

descender
(numbers.Real) The farthest distance from the text baseline to the lowest point below the x-height. This value should always be negative.

text_width
(numbers.Real) The full width of the text.

text_height
(numbers.Real) The full height of the text.

maximum_horizontal_advance
(numbers.Real) The largest distance from the start of one character to the start of the following character.

x1
(numbers.Real) The horizontal value of the bounding box's starting corner.

y1
(numbers.Real) The vertical value of the bounding box's starting corner.

x2
(numbers.Real) The horizontal value of the bounding box's ending corner.

y2
(numbers.Real) The vertical value of the bounding box's ending corner.

x
(numbers.Real) The horizontal value of the origin.

y
(numbers.Real) The vertical value of the origin.

Note: When using *get_font_metrics()*, the returned attributes x1, y1, x2, y2, x and y can be ignored as they are used for internal FreeType rendering, and have no meaningful context.

New in version 0.3.0.

Changed in version 0.6.8: Created sub-class of the namedtuple to improve auto-documentation.

size()
Short-cut method for the width & height of the rendered text.

Returns (width, height)

New in version 0.6.8.

wand.drawing.**GRAVITY_TYPES = ('forget', 'north_west', 'north', 'north_east', 'west', 'center', 'east', 'south_west', 'south', 'south_east', 'static')**
(collections.abc.Sequence) The list of text gravity types.

- 'forget'
- 'north_west'
- 'north'
- 'north_east'
- 'west'
- 'center'
- 'east'
- 'south_west'

- 'south'
- 'south_east'
- 'static'

wand.drawing.**LINE_CAP_TYPES = ('undefined', 'butt', 'round', 'square')**
(collections.abc.Sequence) The list of LineCap types

- 'undefined;
- 'butt'
- 'round'
- 'square'

wand.drawing.**LINE_JOIN_TYPES = ('undefined', 'miter', 'round', 'bevel')**
(collections.abc.Sequence) The list of LineJoin types

- 'undefined'
- 'miter'
- 'round'
- 'bevel'

wand.drawing.**PAINT_METHOD_TYPES = ('undefined', 'point', 'replace', 'floodfill', 'filltoborder', 'reset')**
(collections.abc.Sequence) The list of paint method types.

- 'undefined'
- 'point'
- 'replace'
- 'floodfill'
- 'filltoborder'
- 'reset'

wand.drawing.**STRETCH_TYPES = ('undefined', 'normal', 'ultra_condensed', 'extra_condensed', 'condensed', 'semi_condensed', 'semi_expanded', 'expanded', 'extra_expanded', 'ultra_expanded', 'any')**
(collections.abc.Sequence) The list of stretch types for fonts

- 'undefined;
- 'normal'
- 'ultra_condensed'
- 'extra_condensed'
- 'condensed'
- 'semi_condensed'
- 'semi_expanded'
- 'expanded'
- 'extra_expanded'
- 'ultra_expanded'

- 'any'

wand.drawing.**STYLE_TYPES = ('undefined', 'normal', 'italic', 'oblique', 'any')**
(collections.abc.Sequence) The list of style types for fonts

- 'undefined;
- 'normal'
- 'italic'
- 'oblique'
- 'any'

wand.drawing.**TEXT_ALIGN_TYPES = ('undefined', 'left', 'center', 'right')**
(collections.abc.Sequence) The list of text align types.

- 'undefined'
- 'left'
- 'center'
- 'right'

wand.drawing.**TEXT_DECORATION_TYPES = ('undefined', 'no', 'underline', 'overline', 'line_through')**
(collections.abc.Sequence) The list of text decoration types.

- 'undefined'
- 'no'
- 'underline'
- 'overline'
- 'line_through'

wand.drawing.**TEXT_DIRECTION_TYPES = ('undefined', 'right_to_left', 'left_to_right')**
(collections.abc.Sequence) The list of text direction types.

- 'undefined'
- 'right_to_left'
- 'left_to_right'

4.1.5 wand.sequence — Sequences

New in version 0.3.0.

class wand.sequence.**Sequence**(*image*)
The list-like object that contains every *SingleImage* in the *Image* container. It implements collections.abc.Sequence protocol.

New in version 0.3.0.

append(*image*)
S.append(value) – append value to the end of the sequence

property current_index
(numbers.Integral) The current index of its internal iterator.

Note: It's only for internal use.

extend(*images*, *offset=None*)
S.extend(iterable) – extend sequence by appending elements from the iterable

index_context(*index*)
Scoped setter of *current_index*. Should be used for `with` statement e.g.:

```
with image.sequence.index_context(3):
    print(image.size)
```

Note: It's only for internal use.

insert(*index*, *image*)
S.insert(index, value) – insert value before index

class wand.sequence.**SingleImage**(*wand*, *container*, *c_original_resource*)
Each single image in *Image* container. For example, it can be a frame of GIF animation.

Note that all changes on single images are invisible to their containers unless they are altered a `with ...` context manager.

with Image(filename='animation.gif') as container:

with container.sequence[0] as frame: frame.negate()

New in version 0.3.0.

Changed in version 0.5.1: Only sync changes of a *SingleImage* when exiting a `with ...` context. Not when parent *Image* closes.

container = None
(*wand.image.Image*) The container image.

property delay
(numbers.Integral) The delay to pause before display the next image (in the *sequence* of its *container*). It's hundredths of a second.

property index
(numbers.Integral) The index of the single image in the *container* image.

4.1.6 wand.resource — Global resource management

There is the global resource to manage in MagickWand API. This module implements automatic global resource management through reference counting.

exception wand.resource.**DestroyedResourceError**
An error that rises when some code tries access to an already destroyed resource.

Changed in version 0.3.0: It becomes a subtype of *wand.exceptions.WandException*.

class wand.resource.**Resource**
Abstract base class for MagickWand object that requires resource management. Its all subclasses manage the resource semiautomatically and support `with` statement as well:

```
with Resource() as resource:
    # use the resource...
    pass
```

It doesn't implement constructor by itself, so subclasses should implement it. Every constructor should assign the pointer of its resource data into *resource* attribute inside of `with` *allocate()* context. For example:

```
class Pizza(Resource):
    '''My pizza yummy.'''

    def __init__(self):
        with self.allocate():
            self.resource = library.NewPizza()
```

New in version 0.1.2.

allocate()
Allocates the memory for the resource explicitly. Its subclasses should assign the created resource into *resource* attribute inside of this context. For example:

```
with resource.allocate():
    resource.resource = library.NewResource()
```

c_clear_exception = NotImplemented
(`ctypes.CFUNCTYPE`) The `ctypes` function that clears an exception of the *resource*.

Note: It is an abstract attribute that has to be implemented in the subclass.

c_destroy_resource = NotImplemented
(`ctypes.CFUNCTYPE`) The `ctypes` function that destroys the *resource*.

Note: It is an abstract attribute that has to be implemented in the subclass.

c_get_exception = NotImplemented
(`ctypes.CFUNCTYPE`) The `ctypes` function that gets an exception from the *resource*.

Note: It is an abstract attribute that has to be implemented in the subclass.

c_is_resource = NotImplemented
(`ctypes.CFUNCTYPE`) The `ctypes` predicate function that returns whether the given pointer (that contains a resource data usually) is a valid resource.

Note: It is an abstract attribute that has to be implemented in the subclass.

destroy()
Cleans up the resource explicitly. If you use the resource in `with` statement, it was called implicitly so have not to call it.

get_exception()
Gets a current exception instance.

Returns a current exception. it can be `None` as well if any errors aren't occurred

Return type *wand.exceptions.WandException*

raise_exception(*stacklevel=1*)
Raises an exception or warning if it has occurred.

property resource
Internal pointer to the resource instance. It may raise *DestroyedResourceError* when the resource has destroyed already.

class wand.resource.**ResourceLimits**
Wrapper for MagickCore resource limits. Useful for dynamically reducing system resources before attempting risky, or slow running, *Image* operations.

For example:

```
from wand.image import Image
from wand.resource import limits

# Use 100MB of ram before writing temp data to disk.
limits['memory'] = 1024 * 1024 * 100
# Reject images larger than 1000x1000.
limits['width'] = 1000
limits['height'] = 1000

# Debug resources used.
with Image(filename='user.jpg') as img:
    print('Using {0} of {1} memory'.format(limits.resource('memory'),
                                           limits['memory']))

# Dump list of all limits.
for label in limits:
    print('{0} => {1}'.format(label, limits[label]))
```

Available resource keys:

- `'area'` - Maximum *width* * *height* of a pixel cache before writing to disk.
- `'disk'` - Maximum bytes used by pixel cache on disk before exception is thrown.
- `'file'` - Maximum cache files opened at any given time.
- `'height'` - Maximum height of image before exception is thrown.
- `'list_length'` - Maximum images in sequence. Only available with recent version of ImageMagick.
- `'map'` - Maximum memory map in bytes to allocated for pixel cache before using disk.
- `'memory'` - Maximum bytes to allocated for pixel cache before using disk.
- `'thread'` - Maximum parallel task sub-routines can spawn - if using OpenMP.
- `'throttle'` - Total milliseconds to yield to CPU - if possible.
- `'time'` - Maximum seconds before exception is thrown.
- `'width'` - Maximum width of image before exception is thrown.

New in version 0.5.1.

get_resource_limit(*resource*)
Get the current limit for the resource type.

Parameters **resource** (basestring) – Resource type.

Return type numeric.Integral

New in version 0.5.1.

resource(*resource*)
Get the current value for the resource type.

Parameters **resource** (basestring) – Resource type.

Return type numeric.Integral

New in version 0.5.1.

set_resource_limit(*resource*, *limit*)
Sets a new limit for resource type.

Note: The new limit value must be equal to or less than the maximum limit defined by the policy.xml. Any values set outside normal bounds will be ignored silently.

Parameters

- **resource** (basestring) – Resource type.
- **limit** (numeric.Integral) – New limit value.

New in version 0.5.1.

wand.resource.**genesis**()
Instantiates the MagickWand API.

Warning: Don't call this function directly. Use increment_refcount() and decrement_refcount() functions instead.

wand.resource.**limits = <wand.resource.ResourceLimits object>**
(*ResourceLimits*) Helper to get & set Magick Resource Limits.

New in version 0.5.1.

wand.resource.**terminus**()
Cleans up the MagickWand API.

Warning: Don't call this function directly. Use increment_refcount() and decrement_refcount() functions instead.

4.1.7 wand.exceptions — Errors and warnings

This module maps MagickWand API's errors and warnings to Python's native exceptions and warnings. You can catch all MagickWand errors using Python's natural way to catch errors.

See also:

ImageMagick Exceptions

New in version 0.1.1.

Changed in version 0.5.8: Warning & Error Exceptions are now explicitly defined. Previously ImageMagick domain-based errors were dynamically generated at runtime.

exception wand.exceptions.**BaseError**
Bases: *wand.exceptions.WandException*

Base class for Wand-related errors.

New in version 0.4.4.

exception wand.exceptions.**BaseFatalError**
Bases: *wand.exceptions.WandException*

Base class for Wand-related fatal errors.

New in version 0.4.4.

exception wand.exceptions.**BaseWarning**
Bases: *wand.exceptions.WandException*, Warning

Base class for Wand-related warnings.

New in version 0.4.4.

exception wand.exceptions.**BlobError**
Bases: *wand.exceptions.BaseError*, OSError

A binary large object could not be allocated, read, or written.

exception wand.exceptions.**BlobFatalError**
Bases: *wand.exceptions.BaseFatalError*, OSError

A binary large object could not be allocated, read, or written.

exception wand.exceptions.**BlobWarning**
Bases: *wand.exceptions.BaseWarning*, OSError

A binary large object could not be allocated, read, or written.

exception wand.exceptions.**CacheError**
Bases: *wand.exceptions.BaseError*

Pixels could not be read or written to the pixel cache.

exception wand.exceptions.**CacheFatalError**
Bases: *wand.exceptions.BaseFatalError*

Pixels could not be read or written to the pixel cache.

exception wand.exceptions.**CacheWarning**
Bases: *wand.exceptions.BaseWarning*

Pixels could not be read or written to the pixel cache.

exception wand.exceptions.**CoderError**
Bases: *wand.exceptions.BaseError*

There was a problem with an image coder.

exception wand.exceptions.**CoderFatalError**
Bases: *wand.exceptions.BaseFatalError*

There was a problem with an image coder.

exception wand.exceptions.**CoderWarning**
Bases: *wand.exceptions.BaseWarning*

There was a problem with an image coder.

exception wand.exceptions.**ConfigureError**
Bases: *wand.exceptions.BaseError*

There was a problem getting a configuration file.

exception wand.exceptions.**ConfigureFatalError**
Bases: *wand.exceptions.BaseFatalError*

There was a problem getting a configuration file.

exception wand.exceptions.**ConfigureWarning**
Bases: *wand.exceptions.BaseWarning*

There was a problem getting a configuration file.

exception wand.exceptions.**CorruptImageError**
Bases: *wand.exceptions.BaseError*, ValueError

The image file may be corrupt.

exception wand.exceptions.**CorruptImageFatalError**
Bases: *wand.exceptions.BaseFatalError*, ValueError

The image file may be corrupt.

exception wand.exceptions.**CorruptImageWarning**
Bases: *wand.exceptions.BaseWarning*, ValueError

The image file may be corrupt.

exception wand.exceptions.**DelegateError**
Bases: *wand.exceptions.BaseError*

An ImageMagick delegate failed to complete.

exception wand.exceptions.**DelegateFatalError**
Bases: *wand.exceptions.BaseFatalError*

An ImageMagick delegate failed to complete.

exception wand.exceptions.**DelegateWarning**
Bases: *wand.exceptions.BaseWarning*

An ImageMagick delegate failed to complete.

exception wand.exceptions.**DrawError**
Bases: *wand.exceptions.BaseError*

A drawing operation failed.

exception wand.exceptions.**DrawFatalError**
Bases: *wand.exceptions.BaseFatalError*

A drawing operation failed.

exception wand.exceptions.**DrawWarning**
Bases: *wand.exceptions.BaseWarning*

A drawing operation failed.

exception wand.exceptions.**FileOpenError**
Bases: *wand.exceptions.BaseError*, OSError

The image file could not be opened for reading or writing.

exception wand.exceptions.**FileOpenFatalError**
Bases: *wand.exceptions.BaseFatalError*, OSError

The image file could not be opened for reading or writing.

exception wand.exceptions.**FileOpenWarning**
Bases: *wand.exceptions.BaseWarning*, OSError

The image file could not be opened for reading or writing.

exception wand.exceptions.**ImageError**
Bases: *wand.exceptions.BaseError*

The operation could not complete due to an incompatible image.

exception wand.exceptions.**ImageFatalError**
Bases: *wand.exceptions.BaseFatalError*

The operation could not complete due to an incompatible image.

exception wand.exceptions.**ImageWarning**
Bases: *wand.exceptions.BaseWarning*

The operation could not complete due to an incompatible image.

exception wand.exceptions.**MissingDelegateError**
Bases: *wand.exceptions.BaseError*, ImportError

The image type can not be read or written because the appropriate; delegate is missing.

exception wand.exceptions.**MissingDelegateFatalError**
Bases: *wand.exceptions.BaseFatalError*, ImportError

The image type can not be read or written because the appropriate; delegate is missing.

exception wand.exceptions.**MissingDelegateWarning**
Bases: *wand.exceptions.BaseWarning*, ImportError

The image type can not be read or written because the appropriate; delegate is missing.

exception wand.exceptions.**ModuleError**
Bases: *wand.exceptions.BaseError*

There was a problem with an image module.

exception wand.exceptions.**ModuleFatalError**
Bases: *wand.exceptions.BaseFatalError*

There was a problem with an image module.

exception wand.exceptions.**ModuleWarning**
Bases: *wand.exceptions.BaseWarning*

There was a problem with an image module.

exception wand.exceptions.**MonitorError**
Bases: *wand.exceptions.BaseError*

There was a problem activating the progress monitor.

exception wand.exceptions.**MonitorFatalError**
Bases: *wand.exceptions.BaseFatalError*

There was a problem activating the progress monitor.

exception wand.exceptions.**MonitorWarning**
Bases: *wand.exceptions.BaseWarning*

There was a problem activating the progress monitor.

exception wand.exceptions.**OptionError**
Bases: *wand.exceptions.BaseError*

A command-line option was malformed.

exception wand.exceptions.**OptionFatalError**
Bases: *wand.exceptions.BaseFatalError*

A command-line option was malformed.

exception wand.exceptions.**OptionWarning**
Bases: *wand.exceptions.BaseWarning*

A command-line option was malformed.

exception wand.exceptions.**PolicyError**
Bases: *wand.exceptions.BaseError*

A policy denies access to a delegate, coder, filter, path, or resource.

exception wand.exceptions.**PolicyFatalError**
Bases: *wand.exceptions.BaseFatalError*

A policy denies access to a delegate, coder, filter, path, or resource.

exception wand.exceptions.**PolicyWarning**
Bases: *wand.exceptions.BaseWarning*

A policy denies access to a delegate, coder, filter, path, or resource.

exception wand.exceptions.**RandomError**
Bases: *wand.exceptions.BaseError*

There is a problem generating a true or pseudo-random number.

exception wand.exceptions.**RandomFatalError**
Bases: *wand.exceptions.BaseFatalError*

There is a problem generating a true or pseudo-random number.

exception wand.exceptions.**RandomWarning**
Bases: *wand.exceptions.BaseWarning*

There is a problem generating a true or pseudo-random number.

exception wand.exceptions.**RegistryError**
Bases: *wand.exceptions.BaseError*

There was a problem getting or setting the registry.

exception wand.exceptions.**RegistryFatalError**
Bases: *wand.exceptions.BaseFatalError*

There was a problem getting or setting the registry.

exception wand.exceptions.**RegistryWarning**
Bases: *wand.exceptions.BaseWarning*

There was a problem getting or setting the registry.

exception wand.exceptions.**ResourceLimitError**
Bases: *wand.exceptions.BaseError*, MemoryError

A program resource is exhausted e.g. not enough memory.

exception wand.exceptions.**ResourceLimitFatalError**
Bases: *wand.exceptions.BaseFatalError*, MemoryError

A program resource is exhausted e.g. not enough memory.

exception wand.exceptions.**ResourceLimitWarning**
Bases: *wand.exceptions.BaseWarning*, MemoryError

A program resource is exhausted e.g. not enough memory.

exception wand.exceptions.**StreamError**
Bases: *wand.exceptions.BaseError*, OSError

There was a problem reading or writing from a stream.

exception wand.exceptions.**StreamFatalError**
Bases: *wand.exceptions.BaseFatalError*, OSError

There was a problem reading or writing from a stream.

exception wand.exceptions.**StreamWarning**
Bases: *wand.exceptions.BaseWarning*, OSError

There was a problem reading or writing from a stream.

exception wand.exceptions.**TypeError**
Bases: *wand.exceptions.BaseError*

A font is unavailable; a substitution may have occurred.

exception wand.exceptions.**TypeFatalError**
Bases: *wand.exceptions.BaseFatalError*

A font is unavailable; a substitution may have occurred.

exception wand.exceptions.**TypeWarning**
Bases: *wand.exceptions.BaseWarning*

A font is unavailable; a substitution may have occurred.

exception wand.exceptions.**WandError**
Bases: *wand.exceptions.BaseError*

There was a problem specific to the MagickWand API.

exception wand.exceptions.**WandException**
Bases: Exception

All Wand-related exceptions are derived from this class.

exception wand.exceptions.**WandFatalError**
Bases: *wand.exceptions.BaseFatalError*

There was a problem specific to the MagickWand API.

exception wand.exceptions.**WandLibraryVersionError**
Bases: *wand.exceptions.WandException*

Base class for Wand-related ImageMagick version errors.

New in version 0.3.2.

exception wand.exceptions.**WandRuntimeError**
Bases: *wand.exceptions.WandException*, RuntimeError

Generic class for Wand-related runtime errors.

New in version 0.5.2.

exception wand.exceptions.**WandWarning**
Bases: *wand.exceptions.BaseWarning*

There was a problem specific to the MagickWand API.

exception wand.exceptions.**XServerError**
Bases: *wand.exceptions.BaseError*

An X resource is unavailable.

exception wand.exceptions.**XServerFatalError**
Bases: *wand.exceptions.BaseFatalError*

An X resource is unavailable.

exception wand.exceptions.**XServerWarning**
Bases: *wand.exceptions.BaseWarning*

An X resource is unavailable.

4.1.8 wand.api — Low-level interfaces

Changed in version 0.1.10: Changed to throw `ImportError` instead of `AttributeError` when the shared library fails to load.

wand.api.**load_library**()
Loads the MagickWand library.

Returns the MagickWand library and the ImageMagick library

Return type ctypes.CDLL

wand.api.**library**
ctypes.CDLL) The MagickWand library.

wand.api.**libmagick**
(ctypes.CDLL) The MagickCore library.

New in version 0.1.10.

4.1.9 wand.compat — Compatibility layer

This module provides several subtle things to support multiple Python versions (2.7, 3.3+) and VM implementations (CPython, PyPy).

wand.compat.**PY3 = True**
(bool) Whether it is Python 3.x or not.

wand.compat.**abc = <module 'collections.abc' from '/home/docs/checkouts/readthedocs.org/user_builds/wand/envs/0.6.10/lib/python3.8/collections/abc.py'>**
(module) Module containing abstract base classes. collections in Python 2 and collections.abc in Python 3.

wand.compat.**binary**(*string*, *var=None*)
Makes string to str in Python 2. Makes string to bytes in Python 3.

Parameters

- **string** (bytes, str, unicode) – a string to cast it to *binary_type*
- **var** (str) – an optional variable name to be used for error message

wand.compat.**binary_type**
alias of bytes

wand.compat.**encode_filename**(*filename*)
If filename is a *text_type*, encode it to *binary_type* according to filesystem's default encoding.

Changed in version 0.5.3: Added support for PEP-519 https://github.com/emcconville/wand/pull/339

wand.compat.**file_types**
alias of io.RawIOBase

wand.compat.**string_type**
alias of str

wand.compat.**text_type**
alias of str

wand.compat.**to_bytes**(*value*, *string_pattern='{0}'*)
Short-cut method to allow mixed value types to be converted to bytes.

Parameters

- **value** (basestring, int, float) – Value to be cast to bytes
- **string_pattern** (basestring) – String format to allow printf style control of bytes output.

New in version 0.6.4.

wand.compat.**xrange**
alias of range

4.1.10 wand.display — Displaying images

The *display()* functions shows you the image. It is useful for debugging.

If you are in Mac, the image will be opened by your default image application (**Preview.app** usually).

If you are in Windows, the image will be opened by **imdisplay.exe**, or your default image application (**Windows Photo Viewer** usually) if **imdisplay.exe** is unavailable.

You can use it from CLI also. Execute *wand.display* module through `python -m` option:

```
$ python -m wand.display wandtests/assets/mona-lisa.jpg
```

New in version 0.1.9.

wand.display.**display**(*image*, *server_name=':0'*)
Displays the passed `image`.

Parameters

- **image** (*Image*) – an image to display
- **server_name** (`str`) – X11 server name to use. it is ignored and not used for Mac. default is `':0'`

4.1.11 wand.version — Version data

You can find the current version in the command line interface:

```
$ python -m wand.version
0.6.10
$ python -m wand.version --verbose
Wand 0.6.10
ImageMagick 6.7.7-6 2012-06-03 Q16 http://www.imagemagick.org
$ python -m wand.version --config | grep CC | cut -d : -f 2
gcc -std=gnu99 -std=gnu99
$ python -m wand.version --fonts | grep Helvetica
Helvetica
Helvetica-Bold
Helvetica-Light
Helvetica-Narrow
Helvetica-Oblique
$ python -m wand.version --formats | grep CMYK
CMYK
CMYKA
```

New in version 0.2.0: The command line interface.

New in version 0.2.2: The `--verbose`/`-v` option which also prints ImageMagick library version for CLI.

New in version 0.4.1: The `--fonts`, `--formats`, & `--config` option allows printing additional information about ImageMagick library.

wand.version.**MAGICK_HDRI = None**
(`bool`) True if ImageMagick is compiled for High Dynamic Range Image.

wand.version.**MAGICK_RELEASE_DATE = None**
(`datetime.date`) The release date of the linked ImageMagick library. Equivalent to the result of `GetMagickReleaseDate()` function.

New in version 0.2.1.

wand.version.**MAGICK_RELEASE_DATE_STRING = None**
(basestring) The date string e.g. '2012-06-03' of *MAGICK_RELEASE_DATE_STRING*. This value is the exactly same string to the result of GetMagickReleaseDate() function.

New in version 0.2.1.

wand.version.**MAGICK_VERSION = None**
(basestring) The version string of the linked ImageMagick library. The exactly same string to the result of GetMagickVersion() function.

Example:

```
'ImageMagick 6.7.7-6 2012-06-03 Q16 http://www.imagemagick.org'
```

New in version 0.2.1.

wand.version.**MAGICK_VERSION_FEATURES = 'Cipher DPC Modules OpenMP '**
(basestring) A string of all features enabled. This value is identical to what is returned by GetMagickFeatures()

New in version 0.5.0.

wand.version.**MAGICK_VERSION_INFO = None**
(tuple) The version tuple e.g. (6, 7, 7, 6) of *MAGICK_VERSION*.

New in version 0.2.1.

wand.version.**MAGICK_VERSION_NUMBER = 1792**
(numbers.Integral) The version number of the linked ImageMagick library.

New in version 0.2.1.

wand.version.**QUANTUM_DEPTH = None**
(numbers.Integral) The quantum depth configuration of the linked ImageMagick library. One of 8, 16, 32, or 64.

New in version 0.3.0.

wand.version.**QUANTUM_RANGE = None**
(numbers.Integral) The quantum range configuration of the linked ImageMagick library.

New in version 0.5.0.

wand.version.**QUANTUM_SCALE = None**
(numbers.Real) The quantum scale of the linked ImageMagick library. This is calculated as *1.0 / QUANTUM_RANGE*.

New in version 0.6.8.

wand.version.**VERSION = '0.6.10'**
(basestring) The version string e.g. '0.1.2'.

Changed in version 0.1.9: Becomes string. (It was tuple before.)

wand.version.**VERSION_INFO = (0, 6, 10)**
(tuple) The version tuple e.g. (0, 1, 2).

Changed in version 0.1.9: Becomes tuple. (It was string before.)

wand.version.**configure_options**(*pattern='*'*)
Queries ImageMagick library for configurations options given at compile-time.

Example: Find where the ImageMagick documents are installed:

```
>>> from wand.version import configure_options
>>> configure_options('DOC*')
{'DOCUMENTATION_PATH': '/usr/local/share/doc/ImageMagick-6'}
```

Parameters **pattern** (basestring) – A term to filter queries against. Supports wildcard '*' characters. Default patterns '*' for all options.

Returns Directory of configuration options matching given pattern

Return type collections.defaultdict

wand.version.**fonts**(*pattern='*'*)

Queries ImageMagick library for available fonts.

Available fonts can be configured by defining *types.xml*, *type-ghostscript.xml*, or *type-windows.xml*. Use *wand.version.configure_options()* to locate system search path, and resources article for defining xml file.

Example: List all bold Helvetica fonts:

```
>>> from wand.version import fonts
>>> fonts('*Helvetica*Bold*')
['Helvetica-Bold', 'Helvetica-Bold-Oblique', 'Helvetica-BoldOblique',
 'Helvetica-Narrow-Bold', 'Helvetica-Narrow-BoldOblique']
```

Parameters **pattern** (basestring) – A term to filter queries against. Supports wildcard '*' characters. Default patterns '*' for all options.

Returns Sequence of matching fonts

Return type collections.Sequence

wand.version.**formats**(*pattern='*'*)

Queries ImageMagick library for supported formats.

Example: List supported PNG formats:

```
>>> from wand.version import formats
>>> formats('PNG*')
['PNG', 'PNG00', 'PNG8', 'PNG24', 'PNG32', 'PNG48', 'PNG64']
```

Parameters **pattern** (basestring) – A term to filter formats against. Supports wildcards '*' characters. Default pattern '*' for all formats.

Returns Sequence of matching formats

Return type collections.Sequence

CHAPTER

FIVE

TROUBLESHOOTING

5.1 Stack Overflow

There's a Stack Overflow tag for Wand:

http://stackoverflow.com/questions/tagged/wand

Freely ask questions about Wand including troubleshooting.

Thanks to everyone in the Stack Overflow community for contributions.

5.2 Documentation

The documentation for Wand is hosted by ReadTheDocs.org. The nightly development docs can be found under the latest version, and the most recent release under stable. Previous & maintenance releases are also available.

CHAPTER

SIX

OPEN SOURCE

Wand is an open source software initially written by Hong Minhee (for StyleShare), and is currently maintained by E. McConville. See also the complete list of contributors as well. The source code is distributed under MIT license and you can find it at GitHub repository. Check out now:

```
$ git clone git://github.com/emcconville/wand.git
```

If you find a bug, please notify to our issue tracker. Pull requests are always welcome!

Check out *Wand Changelog* also.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

W

wand, 149
wand.api, 266
wand.color, 235
wand.compat, 266
wand.display, 267
wand.drawing, 240
wand.exceptions, 260
wand.font, 239
wand.image, 149
wand.resource, 257
wand.sequence, 256
wand.version, 268

INDEX

A

abc (*in module wand.compat*), 267
adaptive_blur() (*wand.image.BaseImage method*), 150
adaptive_resize() (*wand.image.BaseImage method*), 150
adaptive_sharpen() (*wand.image.BaseImage method*), 151
adaptive_threshold() (*wand.image.BaseImage method*), 151
affine() (*wand.drawing.Drawing method*), 240
allocate() (*wand.resource.Resource method*), 258
alpha (*wand.color.Color property*), 236
alpha() (*wand.drawing.Drawing method*), 240
alpha_channel (*wand.image.BaseImage property*), 151
ALPHA_CHANNEL_TYPES (*in module wand.image*), 149
alpha_int8 (*wand.color.Color property*), 236
alpha_quantum (*wand.color.Color property*), 236
animation (*wand.image.BaseImage property*), 152
animation (*wand.image.Image property*), 222
annotate() (*wand.image.BaseImage method*), 152
antialias (*wand.font.Font property*), 239
antialias (*wand.image.BaseImage property*), 152
append() (*wand.sequence.Sequence method*), 256
arc() (*wand.drawing.Drawing method*), 241
area (*wand.image.ConnectedComponentObject attribute*), 214
artifacts (*wand.image.Image attribute*), 222
ArtifactTree (*class in wand.image*), 150
ascender (*wand.drawing.FontMetrics attribute*), 253
auto_gamma() (*wand.image.BaseImage method*), 153
auto_level() (*wand.image.BaseImage method*), 153
auto_orient() (*wand.image.BaseImage method*), 153
auto_threshold() (*wand.image.BaseImage method*), 153
AUTO_THRESHOLD_METHODS (*in module wand.image*), 150

B

background_color (*wand.image.BaseImage property*), 153
BaseError, 261
BaseFatalError, 261
BaseImage (*class in wand.image*), 150
BaseWarning, 261
bezier() (*wand.drawing.Drawing method*), 241
binary() (*in module wand.compat*), 267
binary_type (*in module wand.compat*), 267
black (*wand.color.Color property*), 236
black_int8 (*wand.color.Color property*), 236
black_quantum (*wand.color.Color property*), 236
black_threshold() (*wand.image.BaseImage method*), 153
blank() (*wand.image.Image method*), 222
BlobError, 261
BlobFatalError, 261
BlobWarning, 261
blue (*wand.color.Color property*), 236
blue_int8 (*wand.color.Color property*), 236
blue_primary (*wand.image.BaseImage property*), 153
blue_quantum (*wand.color.Color property*), 236
blue_shift() (*wand.image.BaseImage method*), 153
blur() (*wand.image.BaseImage method*), 153
border() (*wand.image.BaseImage method*), 154
border_color (*wand.drawing.Drawing property*), 241
border_color (*wand.image.BaseImage property*), 154
brightness_contrast() (*wand.image.BaseImage method*), 154

C

c_clear_exception (*wand.resource.Resource attribute*), 258
c_destroy_resource (*wand.resource.Resource attribute*), 258
c_equals() (*wand.color.Color static method*), 236
c_get_exception (*wand.resource.Resource attribute*), 258
c_is_resource (*wand.resource.Resource attribute*), 258
CacheError, 261
CacheFatalError, 261
CacheWarning, 261
canny() (*wand.image.BaseImage method*), 154
caption() (*wand.image.BaseImage method*), 155
cdl() (*wand.image.BaseImage method*), 155

center_x (*wand.image.ConnectedComponentObject attribute*), 214
center_y (*wand.image.ConnectedComponentObject attribute*), 215
centroid (*wand.image.ConnectedComponentObject property*), 215
channel_depths (*wand.image.Image attribute*), 223
channel_images (*wand.image.Image attribute*), 223
ChannelDepthDict (*class in wand.image*), 214
ChannelImageDict (*class in wand.image*), 214
CHANNELS (*in module wand.image*), 208
character_height (*wand.drawing.FontMetrics attribute*), 253
character_width (*wand.drawing.FontMetrics attribute*), 253
charcoal() (*wand.image.BaseImage method*), 155
chop() (*wand.image.BaseImage method*), 155
circle() (*wand.drawing.Drawing method*), 241
clahe() (*wand.image.BaseImage method*), 156
clamp() (*wand.image.BaseImage method*), 156
clear() (*wand.image.Image method*), 223
clip_path (*wand.drawing.Drawing property*), 242
CLIP_PATH_UNITS (*in module wand.drawing*), 240
clip_rule (*wand.drawing.Drawing property*), 242
clip_units (*wand.drawing.Drawing property*), 242
clone() (*wand.drawing.Drawing method*), 242
clone() (*wand.image.BaseImage method*), 156
clone() (*wand.image.Iterator method*), 228
clone_from_cc_object_info() (*wand.image.ConnectedComponentObject method*), 215
clone_from_extra_70A_info() (*wand.image.ConnectedComponentObject method*), 215
clone_from_extra_710_info() (*wand.image.ConnectedComponentObject method*), 215
close() (*wand.image.Image method*), 223
ClosedImageError, 214
clut() (*wand.image.BaseImage method*), 156
coalesce() (*wand.image.BaseImage method*), 157
CoderError, 261
CoderFatalError, 262
CoderWarning, 262
Color (*class in wand.color*), 235
color (*wand.font.Font property*), 239
color() (*wand.drawing.Drawing method*), 242
color_decision_list() (*wand.image.BaseImage method*), 157
color_map() (*wand.image.BaseImage method*), 157
color_matrix() (*wand.image.BaseImage method*), 158
color_threshold() (*wand.image.BaseImage method*), 158
colorize() (*wand.image.BaseImage method*), 159
colors (*wand.image.BaseImage property*), 159
colorspace (*wand.image.BaseImage property*), 159
COLORSPACE_TYPES (*in module wand.image*), 209
combine() (*wand.image.BaseImage method*), 159
comment() (*wand.drawing.Drawing method*), 242
compare() (*wand.image.BaseImage method*), 159
compare_layers() (*wand.image.Image method*), 223
COMPARE_METRICS (*in module wand.image*), 210
complex() (*wand.image.BaseImage method*), 160
compose (*wand.image.BaseImage property*), 161
composite() (*wand.drawing.Drawing method*), 242
composite() (*wand.image.BaseImage method*), 161
composite_channel() (*wand.image.BaseImage method*), 161
COMPOSITE_OPERATORS (*in module wand.image*), 211
compression (*wand.image.BaseImage property*), 162
compression_quality (*wand.image.BaseImage property*), 162
COMPRESSION_TYPES (*in module wand.image*), 213
concat() (*wand.image.BaseImage method*), 162
configure_options() (*in module wand.version*), 269
ConfigureError, 262
ConfigureFatalError, 262
ConfigureWarning, 262
connected_components() (*wand.image.BaseImage method*), 162
ConnectedComponentObject (*class in wand.image*), 214
container (*wand.sequence.SingleImage attribute*), 257
contrast() (*wand.image.BaseImage method*), 164
contrast_stretch() (*wand.image.BaseImage method*), 164
convert() (*wand.image.Image method*), 223
convex_hull() (*wand.image.BaseImage method*), 164
CorruptImageError, 262
CorruptImageFatalError, 262
CorruptImageWarning, 262
crop() (*wand.image.BaseImage method*), 165
current_index (*wand.sequence.Sequence property*), 256
cyan (*wand.color.Color property*), 237
cyan_int8 (*wand.color.Color property*), 237
cyan_quantum (*wand.color.Color property*), 237
cycle_color_map() (*wand.image.BaseImage method*), 166

D

data_url() (*wand.image.Image method*), 224
decipher() (*wand.image.BaseImage method*), 166
deconstruct() (*wand.image.BaseImage method*), 166
delay (*wand.image.BaseImage property*), 166
delay (*wand.sequence.SingleImage property*), 257
DelegateError, 262
DelegateFatalError, 262

DelegateWarning, 262
depth (*wand.image.BaseImage property*), 166
descender (*wand.drawing.FontMetrics attribute*), 253
deskew() (*wand.image.BaseImage method*), 166
despeckle() (*wand.image.BaseImage method*), 167
destroy() (*wand.resource.Resource method*), 258
DestroyedResourceError, 257
dirty (*wand.color.Color attribute*), 237
dirty (*wand.image.BaseImage attribute*), 167
display() (*in module wand.display*), 268
dispose (*wand.image.BaseImage property*), 167
DISPOSE_TYPES (*in module wand.image*), 215
distort() (*wand.image.BaseImage method*), 167
DISTORTION_METHODS (*in module wand.image*), 215
DITHER_METHODS (*in module wand.image*), 216
draw() (*wand.drawing.Drawing method*), 243
DrawError, 262
DrawFatalError, 262
Drawing (*class in wand.drawing*), 240
DrawWarning, 263

E

edge() (*wand.image.BaseImage method*), 168
ellipse() (*wand.drawing.Drawing method*), 243
emboss() (*wand.image.BaseImage method*), 168
encipher() (*wand.image.BaseImage method*), 168
encode_filename() (*in module wand.compat*), 267
enhance() (*wand.image.BaseImage method*), 168
environment variable
    MAGICK_HOME, 10–13
    WAND_MAGICK_LIBRARY_SUFFIX, 12, 130
equalize() (*wand.image.BaseImage method*), 168
evaluate() (*wand.image.BaseImage method*), 169
EVALUATE_OPS (*in module wand.image*), 216
export_pixels() (*wand.image.BaseImage method*), 169
extend() (*wand.sequence.Sequence method*), 257
extent() (*wand.image.BaseImage method*), 170

F

features() (*wand.image.BaseImage method*), 170
fft() (*wand.image.BaseImage method*), 171
file_types (*in module wand.compat*), 267
FileOpenError, 263
FileOpenFatalError, 263
FileOpenWarning, 263
fill_color (*wand.drawing.Drawing property*), 243
fill_opacity (*wand.drawing.Drawing property*), 243
fill_rule (*wand.drawing.Drawing property*), 243
FILL_RULE_TYPES (*in module wand.drawing*), 253
FILTER_TYPES (*in module wand.image*), 217
flip() (*wand.image.BaseImage method*), 171
flop() (*wand.image.BaseImage method*), 171
Font (*class in wand.font*), 239
font (*wand.drawing.Drawing property*), 243
font (*wand.image.BaseImage property*), 171
font_antialias (*wand.image.BaseImage property*), 171
font_family (*wand.drawing.Drawing property*), 243
FONT_METRICS_ATTRIBUTES (*in module wand.drawing*), 253
font_path (*wand.image.BaseImage property*), 171
font_resolution (*wand.drawing.Drawing property*), 243
font_size (*wand.drawing.Drawing property*), 243
font_size (*wand.image.BaseImage property*), 172
font_stretch (*wand.drawing.Drawing property*), 244
font_style (*wand.drawing.Drawing property*), 244
font_weight (*wand.drawing.Drawing property*), 244
FontMetrics (*class in wand.drawing*), 253
fonts() (*in module wand.version*), 270
format (*wand.image.BaseImage property*), 172
formats() (*in module wand.version*), 270
forward_fourier_transform() (*wand.image.BaseImage method*), 172
frame() (*wand.image.BaseImage method*), 172
from_array() (*wand.image.Image class method*), 224
from_hsl() (*wand.color.Color class method*), 237
function() (*wand.image.BaseImage method*), 173
FUNCTION_TYPES (*in module wand.image*), 219
fuzz (*wand.image.BaseImage property*), 173
fx() (*wand.image.BaseImage method*), 173

G

gamma() (*wand.image.BaseImage method*), 174
gaussian_blur() (*wand.image.BaseImage method*), 174
genesis() (*in module wand.resource*), 260
get_exception() (*wand.resource.Resource method*), 258
get_font_metrics() (*wand.drawing.Drawing method*), 244
get_image_distortion() (*wand.image.BaseImage method*), 174
get_resource_limit() (*wand.resource.ResourceLimits method*), 259
gravity (*wand.drawing.Drawing property*), 244
gravity (*wand.image.BaseImage property*), 175
GRAVITY_TYPES (*in module wand.drawing*), 254
GRAVITY_TYPES (*in module wand.image*), 219
green (*wand.color.Color property*), 237
green_int8 (*wand.color.Color property*), 237
green_primary (*wand.image.BaseImage property*), 175
green_quantum (*wand.color.Color property*), 237

H

hald_clut() (*wand.image.BaseImage method*), 175

height (*wand.image.BaseImage property*), 175
height (*wand.image.ConnectedComponentObject attribute*), 215
histogram (*wand.image.BaseImage property*), 175
HistogramDict (*class in wand.image*), 219
hough_lines() (*wand.image.BaseImage method*), 175
hsl() (*wand.color.Color method*), 237

I

ift() (*wand.image.BaseImage method*), 176
Image (*class in wand.image*), 221
image (*wand.image.ImageProperty property*), 228
image_add() (*wand.image.Image method*), 225
image_get() (*wand.image.Image method*), 225
IMAGE_LAYER_METHOD (*in module wand.image*), 219
image_remove() (*wand.image.Image method*), 225
image_set() (*wand.image.Image method*), 225
image_swap() (*wand.image.Image method*), 225
IMAGE_TYPES (*in module wand.image*), 220
ImageError, 263
ImageFatalError, 263
ImageProperty (*class in wand.image*), 228
ImageWarning, 263
implode() (*wand.image.BaseImage method*), 176
import_pixels() (*wand.image.BaseImage method*), 176
index (*wand.sequence.SingleImage property*), 257
index_context() (*wand.sequence.Sequence method*), 257
insert() (*wand.sequence.Sequence method*), 257
interlace_scheme (*wand.image.BaseImage property*), 177
INTERLACE_TYPES (*in module wand.image*), 220
interpolate_method (*wand.image.BaseImage property*), 177
inverse_fourier_transform() (*wand.image.BaseImage method*), 177
Iterator (*class in wand.image*), 228
iterator_first() (*wand.image.BaseImage method*), 177
iterator_get() (*wand.image.BaseImage method*), 177
iterator_last() (*wand.image.BaseImage method*), 178
iterator_length() (*wand.image.BaseImage method*), 178
iterator_next() (*wand.image.BaseImage method*), 178
iterator_previous() (*wand.image.BaseImage method*), 178
iterator_reset() (*wand.image.BaseImage method*), 178
iterator_set() (*wand.image.BaseImage method*), 178

K

KERNEL_INFO_TYPES (*in module wand.image*), 229
kmeans() (*wand.image.BaseImage method*), 178
kurtosis (*wand.image.BaseImage property*), 178
kurtosis_channel() (*wand.image.BaseImage method*), 179
kuwahara() (*wand.image.BaseImage method*), 179

L

label() (*wand.image.BaseImage method*), 179
left (*wand.image.ConnectedComponentObject attribute*), 215
length_of_bytes (*wand.image.BaseImage property*), 180
level() (*wand.image.BaseImage method*), 180
level_colors() (*wand.image.BaseImage method*), 180
levelize() (*wand.image.BaseImage method*), 180
levelize_colors() (*wand.image.BaseImage method*), 181
libmagick (*in module wand.api*), 266
library (*in module wand.api*), 266
limits (*in module wand.resource*), 260
line() (*wand.drawing.Drawing method*), 244
LINE_CAP_TYPES (*in module wand.drawing*), 255
LINE_JOIN_TYPES (*in module wand.drawing*), 255
linear_stretch() (*wand.image.BaseImage method*), 181
liquid_rescale() (*wand.image.BaseImage method*), 181
load_library() (*in module wand.api*), 266
local_contrast() (*wand.image.BaseImage method*), 182
loop (*wand.image.BaseImage property*), 182

M

magenta (*wand.color.Color property*), 238
magenta_int8 (*wand.color.Color property*), 238
MAGICK_HDRI (*in module wand.version*), 268
MAGICK_HOME, 10–13
MAGICK_RELEASE_DATE (*in module wand.version*), 268
MAGICK_RELEASE_DATE_STRING (*in module wand.version*), 269
MAGICK_VERSION (*in module wand.version*), 269
MAGICK_VERSION_FEATURES (*in module wand.version*), 269
MAGICK_VERSION_INFO (*in module wand.version*), 269
MAGICK_VERSION_NUMBER (*in module wand.version*), 269
magnify() (*wand.image.BaseImage method*), 182
make_blob() (*wand.image.Image method*), 225
manipulative() (*in module wand.image*), 235
matte() (*wand.drawing.Drawing method*), 244
matte_color (*wand.image.BaseImage property*), 182

maxima (*wand.image.BaseImage property*), 182
maximum_horizontal_advance (*wand.drawing.FontMetrics attribute*), 254
mean (*wand.image.BaseImage property*), 182
mean_channel() (*wand.image.BaseImage method*), 183
mean_color (*wand.image.ConnectedComponentObject attribute*), 215
mean_shift() (*wand.image.BaseImage method*), 183
merge (*wand.image.ConnectedComponentObject attribute*), 215
merge_layers() (*wand.image.BaseImage method*), 183
Metadata (*class in wand.image*), 230
metadata (*wand.image.Image attribute*), 225
metric (*wand.image.ConnectedComponentObject attribute*), 215
mimetype (*wand.image.Image property*), 225
minima (*wand.image.BaseImage property*), 183
minimum_bounding_box() (*wand.image.BaseImage method*), 184
MissingDelegateError, 263
MissingDelegateFatalError, 263
MissingDelegateWarning, 263
mode() (*wand.image.BaseImage method*), 185
modulate() (*wand.image.BaseImage method*), 185
module
  wand, 149
  wand.api, 266
  wand.color, 235
  wand.compat, 266
  wand.display, 267
  wand.drawing, 240
  wand.exceptions, 260
  wand.font, 239
  wand.image, 149
  wand.resource, 257
  wand.sequence, 256
  wand.version, 268
ModuleError, 263
ModuleFatalError, 263
ModuleWarning, 263
MonitorError, 264
MonitorFatalError, 264
MonitorWarning, 264
montage() (*wand.image.Image method*), 226
morphology() (*wand.image.BaseImage method*), 185
MORPHOLOGY_METHODS (*in module wand.image*), 230
motion_blur() (*wand.image.BaseImage method*), 186

N

negate() (*wand.image.BaseImage method*), 186
next() (*wand.image.Iterator method*), 228
noise() (*wand.image.BaseImage method*), 186
NOISE_TYPES (*in module wand.image*), 231
normalize() (*wand.image.BaseImage method*), 186
normalized_string (*wand.color.Color property*), 238

O

offset (*wand.image.ConnectedComponentObject property*), 215
oil_paint() (*wand.image.BaseImage method*), 187
opacity (*wand.drawing.Drawing property*), 245
opaque_paint() (*wand.image.BaseImage method*), 187
optimize_layers() (*wand.image.BaseImage method*), 187
optimize_transparency() (*wand.image.BaseImage method*), 187
OptionDict (*class in wand.image*), 231
OptionError, 264
OptionFatalError, 264
options (*wand.image.BaseImage attribute*), 187
OptionWarning, 264
ordered_dither() (*wand.image.BaseImage method*), 187
orientation (*wand.image.BaseImage property*), 188
ORIENTATION_TYPES (*in module wand.image*), 231

P

page (*wand.image.BaseImage property*), 188
page_height (*wand.image.BaseImage property*), 188
page_width (*wand.image.BaseImage property*), 189
page_x (*wand.image.BaseImage property*), 189
page_y (*wand.image.BaseImage property*), 189
PAINT_METHOD_TYPES (*in module wand.drawing*), 255
PAPERSIZE_MAP (*in module wand.image*), 231
parse_meta_geometry() (*wand.image.BaseImage method*), 189
path (*wand.font.Font property*), 239
path_close() (*wand.drawing.Drawing method*), 245
path_curve() (*wand.drawing.Drawing method*), 245
path_curve_to_quadratic_bezier() (*wand.drawing.Drawing method*), 245
path_elliptic_arc() (*wand.drawing.Drawing method*), 246
path_finish() (*wand.drawing.Drawing method*), 246
path_horizontal_line() (*wand.drawing.Drawing method*), 246
path_line() (*wand.drawing.Drawing method*), 246
path_move() (*wand.drawing.Drawing method*), 246
path_start() (*wand.drawing.Drawing method*), 247
path_vertical_line() (*wand.drawing.Drawing method*), 247
percent_escape() (*wand.image.BaseImage method*), 189
ping() (*wand.image.Image class method*), 226
PIXEL_INTERPOLATE_METHODS (*in module wand.image*), 232
point() (*wand.drawing.Drawing method*), 247
polaroid() (*wand.image.BaseImage method*), 189

PolicyError, 264
PolicyFatalError, 264
PolicyWarning, 264
polygon() (*wand.drawing.Drawing method*), 247
polyline() (*wand.drawing.Drawing method*), 247
polynomial() (*wand.image.BaseImage method*), 190
pop() (*wand.drawing.Drawing method*), 248
pop_clip_path() (*wand.drawing.Drawing method*), 248
pop_defs() (*wand.drawing.Drawing method*), 248
pop_pattern() (*wand.drawing.Drawing method*), 248
posterize() (*wand.image.BaseImage method*), 190
ProfileDict (*class in wand.image*), 232
profiles (*wand.image.Image attribute*), 226
pseudo() (*wand.image.Image method*), 226
push() (*wand.drawing.Drawing method*), 248
push_clip_path() (*wand.drawing.Drawing method*), 249
push_defs() (*wand.drawing.Drawing method*), 249
push_pattern() (*wand.drawing.Drawing method*), 249
PY3 (*in module wand.compat*), 267

Q

quantize() (*wand.image.BaseImage method*), 190
QUANTUM_DEPTH (*in module wand.version*), 269
QUANTUM_RANGE (*in module wand.version*), 269
quantum_range (*wand.image.BaseImage property*), 191
QUANTUM_SCALE (*in module wand.version*), 269

R

raise_exception() (*wand.resource.Resource method*), 259
random_threshold() (*wand.image.BaseImage method*), 191
RandomError, 264
RandomFatalError, 264
RandomWarning, 264
range_channel() (*wand.image.BaseImage method*), 191
range_threshold() (*wand.image.BaseImage method*), 191
read() (*wand.image.Image method*), 227
read_mask() (*wand.image.BaseImage method*), 192
rectangle() (*wand.drawing.Drawing method*), 249
red (*wand.color.Color property*), 238
red_int8 (*wand.color.Color property*), 238
red_primary (*wand.image.BaseImage property*), 192
red_quantum (*wand.color.Color property*), 238
region() (*wand.image.BaseImage method*), 192
RegistryError, 264
RegistryFatalError, 265
RegistryWarning, 265
remap() (*wand.image.BaseImage method*), 193
rendering_intent (*wand.image.BaseImage property*), 193
RENDERING_INTENT_TYPES (*in module wand.image*), 233
resample() (*wand.image.BaseImage method*), 193
reset_coords() (*wand.image.BaseImage method*), 194
reset_sequence() (*wand.image.BaseImage method*), 194
reset_sequence() (*wand.image.Image method*), 227
resize() (*wand.image.BaseImage method*), 194
resolution (*wand.image.BaseImage property*), 194
Resource (*class in wand.resource*), 257
resource (*wand.resource.Resource property*), 259
resource() (*wand.resource.ResourceLimits method*), 260
ResourceLimitError, 265
ResourceLimitFatalError, 265
ResourceLimits (*class in wand.resource*), 259
ResourceLimitWarning, 265
roll() (*wand.image.BaseImage method*), 194
rotate() (*wand.drawing.Drawing method*), 250
rotate() (*wand.image.BaseImage method*), 194
rotational_blur() (*wand.image.BaseImage method*), 195

S

sample() (*wand.image.BaseImage method*), 195
sampling_factors (*wand.image.BaseImage property*), 195
save() (*wand.image.Image method*), 227
scale() (*wand.drawing.Drawing method*), 250
scale() (*wand.image.BaseImage method*), 195
scale_quantum_to_int8() (*in module wand.color*), 238
scene (*wand.image.BaseImage property*), 196
seed (*wand.image.BaseImage property*), 196
selective_blur() (*wand.image.BaseImage method*), 196
sepia_tone() (*wand.image.BaseImage method*), 196
Sequence (*class in wand.sequence*), 256
sequence (*wand.image.BaseImage attribute*), 196
set_fill_pattern_url() (*wand.drawing.Drawing method*), 251
set_resource_limit() (*wand.resource.ResourceLimits method*), 260
set_stroke_pattern_url() (*wand.drawing.Drawing method*), 251
shade() (*wand.image.BaseImage method*), 197
shadow() (*wand.image.BaseImage method*), 197
sharpen() (*wand.image.BaseImage method*), 197
shave() (*wand.image.BaseImage method*), 197
shear() (*wand.image.BaseImage method*), 197

sigmoidal_contrast() (*wand.image.BaseImage method*), 198
signature (*wand.image.BaseImage property*), 198
similarity() (*wand.image.BaseImage method*), 198
SingleImage (*class in wand.sequence*), 257
size (*wand.font.Font property*), 239
size (*wand.image.BaseImage property*), 199
size (*wand.image.ConnectedComponentObject property*), 215
size() (*wand.drawing.FontMetrics method*), 254
sketch() (*wand.image.BaseImage method*), 199
skew() (*wand.drawing.Drawing method*), 251
skewness (*wand.image.BaseImage property*), 199
smush() (*wand.image.BaseImage method*), 199
solarize() (*wand.image.BaseImage method*), 200
sparse_color() (*wand.image.BaseImage method*), 200
SPARSE_COLOR_METHODS (*in module wand.image*), 233
splice() (*wand.image.BaseImage method*), 201
spread() (*wand.image.BaseImage method*), 201
standard_deviation (*wand.image.BaseImage property*), 201
statistic() (*wand.image.BaseImage method*), 201
STATISTIC_TYPES (*in module wand.image*), 233
stegano() (*wand.image.BaseImage method*), 202
stereogram() (*wand.image.Image class method*), 228
STORAGE_TYPES (*in module wand.image*), 234
StreamError, 265
StreamFatalError, 265
StreamWarning, 265
STRETCH_TYPES (*in module wand.drawing*), 255
string (*wand.color.Color property*), 238
string_type (*in module wand.compat*), 267
strip() (*wand.image.BaseImage method*), 202
stroke_antialias (*wand.drawing.Drawing property*), 251
stroke_color (*wand.drawing.Drawing property*), 251
stroke_color (*wand.font.Font property*), 239
stroke_dash_array (*wand.drawing.Drawing property*), 251
stroke_dash_offset (*wand.drawing.Drawing property*), 251
stroke_line_cap (*wand.drawing.Drawing property*), 251
stroke_line_join (*wand.drawing.Drawing property*), 251
stroke_miter_limit (*wand.drawing.Drawing property*), 252
stroke_opacity (*wand.drawing.Drawing property*), 252
stroke_width (*wand.drawing.Drawing property*), 252
stroke_width (*wand.font.Font property*), 240
STYLE_TYPES (*in module wand.drawing*), 256
swirl() (*wand.image.BaseImage method*), 202

T

terminus() (*in module wand.resource*), 260
text() (*wand.drawing.Drawing method*), 252
TEXT_ALIGN_TYPES (*in module wand.drawing*), 256
text_alignment (*wand.drawing.Drawing property*), 252
text_antialias (*wand.drawing.Drawing property*), 252
text_decoration (*wand.drawing.Drawing property*), 252
TEXT_DECORATION_TYPES (*in module wand.drawing*), 256
text_direction (*wand.drawing.Drawing property*), 252
TEXT_DIRECTION_TYPES (*in module wand.drawing*), 256
text_encoding (*wand.drawing.Drawing property*), 252
text_height (*wand.drawing.FontMetrics attribute*), 254
text_interline_spacing (*wand.drawing.Drawing property*), 252
text_interword_spacing (*wand.drawing.Drawing property*), 252
text_kerning (*wand.drawing.Drawing property*), 252
text_type (*in module wand.compat*), 267
text_under_color (*wand.drawing.Drawing property*), 252
text_width (*wand.drawing.FontMetrics attribute*), 254
texture() (*wand.image.BaseImage method*), 202
threshold() (*wand.image.BaseImage method*), 203
thumbnail() (*wand.image.BaseImage method*), 203
ticks_per_second (*wand.image.BaseImage property*), 203
tint() (*wand.image.BaseImage method*), 203
to_bytes() (*in module wand.compat*), 267
top (*wand.image.ConnectedComponentObject attribute*), 215
transform() (*wand.image.BaseImage method*), 203
transform_colorspace() (*wand.image.BaseImage method*), 204
translate() (*wand.drawing.Drawing method*), 252
transparent_color() (*wand.image.BaseImage method*), 205
transparentize() (*wand.image.BaseImage method*), 205
transpose() (*wand.image.BaseImage method*), 205
transverse() (*wand.image.BaseImage method*), 205
trim() (*wand.image.BaseImage method*), 205
type (*wand.image.BaseImage property*), 206
TypeError, 265
TypeFatalError, 265
TypeWarning, 265

U

unique_colors() (*wand.image.BaseImage method*),

206
UNIT_TYPES (*in module wand.image*), 234
units (*wand.image.BaseImage property*), 206
unsharp_mask() (*wand.image.BaseImage method*), 206

V

vector_graphics (*wand.drawing.Drawing property*), 253
VERSION (*in module wand.version*), 269
VERSION_INFO (*in module wand.version*), 269
viewbox() (*wand.drawing.Drawing method*), 253
vignette() (*wand.image.BaseImage method*), 206
virtual_pixel (*wand.image.BaseImage property*), 207
VIRTUAL_PIXEL_METHOD (*in module wand.image*), 234

W

wand
    module, 149
wand (*wand.image.BaseImage property*), 207
wand.api
    module, 266
wand.color
    module, 235
wand.compat
    module, 266
wand.display
    module, 267
wand.drawing
    module, 240
wand.exceptions
    module, 260
wand.font
    module, 239
wand.image
    module, 149
wand.resource
    module, 257
wand.sequence
    module, 256
wand.version
    module, 268
WAND_MAGICK_LIBRARY_SUFFIX, 12, 130
WandError, 265
WandException, 265
WandFatalError, 266
WandLibraryVersionError, 266
WandRuntimeError, 266
WandWarning, 266
watermark() (*wand.image.BaseImage method*), 207
wave() (*wand.image.BaseImage method*), 207
wavelet_denoise() (*wand.image.BaseImage method*), 207
white_balance() (*wand.image.BaseImage method*), 208
white_point (*wand.image.BaseImage property*), 208
white_threshold() (*wand.image.BaseImage method*), 208
width (*wand.image.BaseImage property*), 208
width (*wand.image.ConnectedComponentObject attribute*), 215
write_mask() (*wand.image.BaseImage method*), 208

X

x (*wand.drawing.FontMetrics attribute*), 254
x1 (*wand.drawing.FontMetrics attribute*), 254
x2 (*wand.drawing.FontMetrics attribute*), 254
xrange (*in module wand.compat*), 267
XServerError, 266
XServerFatalError, 266
XServerWarning, 266

Y

y (*wand.drawing.FontMetrics attribute*), 254
y1 (*wand.drawing.FontMetrics attribute*), 254
y2 (*wand.drawing.FontMetrics attribute*), 254
yellow (*wand.color.Color property*), 238
yellow_int8 (*wand.color.Color property*), 238
yellow_quantum (*wand.color.Color property*), 238