Contents

1 Why just another binding? 3

2 Requirements 5

3 User's guide 7
  3.1 What's new in Wand 0.5? 7
  3.2 Installation 10
  3.3 Reading images 14
  3.4 Writing images 17
  3.5 Resizing and cropping 19
  3.6 Image Effects 23
  3.7 Special Effects (FX) 33
  3.8 Transformation 43
  3.9 Colorspace 49
  3.10 Color Enhancement 50
  3.11 Distortion 56
  3.12 Drawing 63
  3.13 Reading EXIF 74
  3.14 Layers 75
  3.15 Sequence 81
  3.16 Resource management 83
  3.17 CLI Reference 83
  3.18 Running tests 87
  3.19 Roadmap 89
  3.20 Wand Changelog 89
  3.21 Talks and Presentations 107

4 References 109
  4.1 wand — Simple MagickWand API binding for Python 109

5 Troubleshooting 217
  5.1 Mailing list 217
  5.2 Stack Overflow 217
  5.3 Documentation 217

6 Open source 219
7 Indices and tables 221
Python Module Index 223
Index 225
Wand is a ctypes-based simple ImageMagick binding for Python.

```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 1

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 2

Requirements

• Python 2.6 or higher
  – CPython 2.6 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
3.1 What’s new in Wand 0.5?

This guide doesn’t cover all changes in 0.5. See also the full list of changes in 0.5 series.

3.1.1 Numpy I/O

New in version 0.5.3.

Instances of Wand’s Image can be created directly from a numpy.array object, or other objects that implements array interface protocol.

```python
import numpy as nprom wand.image import Image

array = np.zeros([100, 100, 3], dtype=np.uint8)
array[:, :, :] = [0xff, 0x00, 0x00]

with Image.from_array(array) as img:
    print(img[0, 0]) #=> srgb(255, 0, 0)
```

You can also convert an instance of Image into an array.

```python
import numpy as np
from wand.image import Image

with Image(filename='rose:') as img:
    array = np.array(img)
    print(array.shape) #=> (70, 46, 3)
```
3.1.2 Resource Limits

New in version 0.5.1.

The `limits` dictionary helper allows you to define run-time policies. Doing this allows your application to process images without consuming too much system resources.

```python
from wand.image import Image
from wand.resource import limits

limits['thread'] = 1  # Only allow one CPU thread for raster.

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

See `ResourceLimits` for more details.

3.1.3 Import & Extract Profiles

New in version 0.5.1.

Embedded profiles, like ICC, can be accessed via `Image.profiles` dictionary.

```python
with Image(filename='photo.jpg') as photo:
    with open('color_profile.icc', 'rb') as profile:
        photo.profiles['icc'] = profile.read()
```

**Hint:** Each profile payload will be a raw binary blob. ImageMagick & Wand will not edit payloads, but only get, set, and delete them from an image.

See `ProfileDict` for more details.

3.1.4 Pseudo Images

New in version 0.5.0.

The `Image` constructor now accepts the `pseudo` parameter. This allows you to quickly read Pseudo-image Formats, and Built-in Patterns.

Checkout `Open a Pseudo Image` for some examples.

3.1.5 ImageMagick-7 Support

New in version 0.5.0.

The release of Wand 0.5 now supports both versions of ImageMagick-6 & ImageMagick-7. ImageMagick-7 introduces some key behavior changes, and some care should go into any application that was previously written for ImageMagick-6 before upgrading system libraries.

To understand the fundamental changes, please review `Porting to ImageMagick Version 7` for a more definitive overview.

**Notes on Porting 6 to 7**

A few key changes worth reviewing.
HDRI by Default

Vanilla installs of ImageMagick-7 include HDRI enabled by default. Users may experience increase depth of color, but with reduced performances during certain color manipulation routines. Max color-values should never be hard-coded, but rely on `Image.quantum_range` to ensure consistent results. It is also possible to experience color-value underflow / overflow during arithmetic operations when upgrading.

An example of an underflow between versions:

```plaintext
# ImageMagick-6
with Image(width=1, height=1, background=Color("gray5")) as canvas:
    canvas.evaluate("subtract", canvas.quantum_range * 0.07)
    print(canvas[0, 0])  #=> srgb(0,0,0)

# ImageMagick-7
with Image(width=1, height=1, background=Color("gray5")) as canvas:
    canvas.evaluate("subtract", canvas.quantum_range * 0.07)
    print(canvas[0, 0])  #=> srgb(-1.90207%,-1.90207%,-1.90207%)
```

The majority of the image-manipulation methods are guarded against overflows by internal clamping routines, but folks working directly with `Image.evaluate()`, `Image.function()`, and `Image.composite_channel()` should take caution. Method `Image.clamp()` as been provided for this task:

```plaintext
with Image(width=1, height=1, background=Color("gray5")) as canvas:
    canvas.evaluate("subtract", canvas.quantum_range * 0.07)
    canvas.clamp()
    print(canvas[0, 0])  #=> srgb(0,0,0)
```

Image Color-Channel Awareness

With ImageMagick-7, colors have descriptive traits, and are managed through channel-masks. An elegant solution to manage active color channels, and simplify core library functions.

Users implementing `Image.composite_channel()` should review previous solutions of composite "copy..." operators as the behavior may have changed.

You can play around with the effects of channel masks with `MagickSetImageChannelMask()` function. For example:

```python
from wand.image import Image, CHANNELS
from wand.api import library

with Image(filename="rose:") as img:
    # Isolate only Red & Green channels
    active_mask = CHANNELS["red"] | CHANNELS["green"]
    previous_mask = library.MagickSetImageChannelMask(img.wand, active_mask)
    img.evaluate("rightshift", 1)
    # Restore previous channels
    library.MagickSetImageChannelMask(img.wand, previous_mask)
    img.save(filename="blue_rose.png")
```

Alpha Replaces Opacity & Matte

Opacity methods & enumerated value have been renamed to alpha with ImageMagick-7. Although the majority of the functionalities are the same, user are responsible for checking the library version before calling an opacity method /
For example:

```python
from wand.version import MAGICK_VERSION_NUMBER
from wand.image import Image

with Image(filename="wizard:") as img:
    image_type = "truecoloralpha"  # IM7 enum
    if MAGICK_VERSION_NUMBER < 0x700:  # Backwards support for IM6
        image_type = "truecolormatte"
    img.type = image_type
```

The reference documentation have been updated to note specific values available per ImageMagick versions.

**Note:** For “What’s New in Wand 0.4”, 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**

#### 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
```

If you need SVG, WMF, OpenEXR, DjVu, and Graphviz support you have to install `libmagickcore5-extra` as well:

```
$ sudo apt-get install libmagickcore5-extra
```
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:

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

3.2.3 Install ImageMagick on Mac

You need one of Homebrew or MacPorts to install ImageMagick.

**Homebrew**

```bash
$ brew install imagemagick
```

If `seam carving` (Image.liquid_rescale()) is needed you have install libcq as well:

```bash
$ brew install imagemagick --with-libqcr
```

**MacPorts**

```bash
$ sudo port install imagemagick
```

If your Python is 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.

```bash
$ 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:

http://legacy.imagemagick.org/script/binary-releases.php#windows

Choose a binary for your architecture:

- **Windows 32-bit**  ImageMagick-6.9.x-x-Q16-x86-dll.exe
- **Windows 64-bit**  ImageMagick-6.9.x-x-Q16-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:

<table>
<thead>
<tr>
<th>~~~~~~~~~ Library Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>libMagickWand-7.Q16HDRI.so</td>
</tr>
<tr>
<td>~ Major version number. Can be blank, 6, or 7.</td>
</tr>
<tr>
<td>~~~ Magick Quantum. Can be blank, Q8, or Q16.</td>
</tr>
<tr>
<td>~~~~ Optional HDRI-Support. Can be blank, or HDRI</td>
</tr>
</tbody>
</table>

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

```bash
$ 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:

```bash
$ 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:

```bash
$ 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:

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

3.2. Installation
3.3 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.

3.3.1 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`:

```python
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).

3.3.2 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:

```python
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.
3.3.3 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:

```python
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.

3.3.4 Clone an image

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

```python
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:

```python
from wand.image import Image

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

3.3.5 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:

```python
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.

3.3.6 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.

### 3.3.7 Open a Pseudo Image

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

from wand.image import Image

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

Common 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.

### 3.3.8 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:
You can also use \[W \times H\] format to resize the input image during read:

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

Cropping an image can be achieved by following the \[W \times H + x+y\] modifier:

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

### 3.4 Writing images

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

#### 3.4.1 Convert images to JPEG

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

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

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

```python
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:

```python
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:
3.4.2 Save to file

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

```python
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:

```python
from wand.image import Image

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

3.4.3 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`:

```python
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()
```

3.4.4 Get binary string

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

```python
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')

3.5 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 \textit{width} and \textit{height} properties:

\begin{verbatim}
>>> from urllib.request import urlopen
>>> 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
\end{verbatim}

If you want the pair of (width, height), check \textit{size} property also.

Note: These three properties are all readonly.

3.5.1 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.

\texttt{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.

\begin{verbatim}
>>> img.size
(500, 600)
>>> img.resize(50, 60)
>>> img.size
(50, 60)
\end{verbatim}
3.5.2 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:

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

3.5.3 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:

```python
>>> 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`:

```python
>>> 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:

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

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

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

3.5.4 Transform images

Use this function to crop and resize and 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:

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

Other example calls:

```python
# 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.

### 3.5.5 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:
You can easily rescale images with seam carving using Wand: use `Image.liquid_rescale()` method:

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

(continues on next page)
(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.

3.6 Image Effects

3.6.1 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.

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

![Original](original-image.png) ![Blur](blur-image.png)

3.6. Image Effects 23
**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).

```python
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")
```

---

**Gaussian Blur**

New in version 0.3.3.

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

```python
from wand.image import Image

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

```python
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")
```

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.

```python
from wand.image import Image

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

### 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.

```python
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")
```
3.6.2 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*.

```python
from wand.image import Image

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

3.6.3 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.
```python
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")
```

### 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*.

```python
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")
```
3.6.5 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.

```python
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")
```

---

3.6.6 Shade

New in version 0.5.0.

Creates a 3D effect by simulating light from source where azimuth 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.

```python
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")
```
3.6.7 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.

```python
from wand.image import Image

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

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.

```python
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")
```

<table>
<thead>
<tr>
<th>Original</th>
<th>Adaptive Sharpen</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Original Image" /></td>
<td><img src="image2.png" alt="Adaptive Sharpen" /></td>
</tr>
</tbody>
</table>

**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 sharpen.

```python
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")
```
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.

```python
from wand.image import Image

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

3.7.1 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.

```python
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 vs. Noise comparison](image)

3.7.2 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.

```python
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    # (code continues here)
```

(continues on next page)
3.7.3 Charcoal

New in version 0.5.3.

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

```python
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")
```

3.7.4 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.

\[
\begin{align*}
\text{red}' &= 0.0 \times \text{red} + 0.0 \times \text{green} + 1.0 \times \text{blue} \\
\text{green}' &= 0.0 \times \text{red} + 1.0 \times \text{green} + 0.0 \times \text{blue} \\
\text{blue}' &= 1.0 \times \text{red} + 0.0 \times \text{green} + 0.0 \times \text{blue}
\end{align*}
\]

```python
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")
```

3.7.5 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.

```python
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")
```
**3.7.6 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°.

```python
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 vs. FX](image-url)
3.7.7 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.

```python
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

![Original Image](image1.png)

### Implode

![Imploded Image](image2.png)

3.7.8 Polaroid

New in version 0.5.4.

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

```python
from wand.image import Image

with Image(filename="inca_tern.jpg") as img:
    img.polaroid()
    img.save(filename="fx-polaroid.jpg")
```
New in version 0.5.7.

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

```python
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')
```

3.7.10 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")

3.7.11 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")
3.7.12 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.

```python
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")
```

3.7.13 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.

```python
with Image(filename='inca_tern.jpg') as img:
    img.swirl(degree=-90)
    img.save(filename='fx-swirl.jpg')
```
3.7.14 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.

```python
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")
```

3.7.15 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")

3.7.16 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")
3.7.17 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.

```python
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")
```

3.8 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.

3.8.1 Enhance

New in version 0.5.0.

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

```python
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.enhance()
        right.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-enhance.jpg")
```
3.8.2 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:

```python
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:

```python
from wand.image import Image

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

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

### 3.8.3 Remap

New in version 0.5.3.

Remap replaces all pixels with the closest matching pixel found in the *affinity* reference image.
```python
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        with Image(width=100, height=1, pseudo="plasma") as affinity:
            right.remap(affinity)
            left.extent(width=left.width*2)
            left.composite(right, top=0, left=right.width)
            left.save(filename="hummingbird-remap.jpg")
```

### 3.8.4 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):

```python
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:

3.8. Transformation
3.8.5 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.

```python
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")
```

### 3.9 Colorspace

#### 3.9.1 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.

```python
>>> 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:

```python
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.
3.9.2 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.

```python
>>> 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:

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

3.10 Color Enhancement

3.10.1 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')

3.10.2 Function Expression

New in version 0.4.1.

Similar to \texttt{evaluate()}, \texttt{function()} applies a multi-argument function to pixel channels. See \texttt{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:

```python
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])
```
3.10.3 Gamma

New in version 0.4.1.

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

The \texttt{level()} method can also adjust \texttt{gamma} value.

For example, when given image \texttt{enhancement.jpg}:

![Image](image.png)

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

```python
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)
```

3.10. Color Enhancement
3.10.4 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:

```python
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')
```
3.11 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.

3.11.1 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.

```python
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.
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, 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%'

3.11.2 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.

<table>
<thead>
<tr>
<th>Total Arguments</th>
<th>Argument Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Angle</td>
</tr>
<tr>
<td>2</td>
<td>Scale, Angle</td>
</tr>
<tr>
<td>3</td>
<td>X, Y, Angle</td>
</tr>
<tr>
<td>4</td>
<td>X, Y, Scale, Angle</td>
</tr>
<tr>
<td>5</td>
<td>X, Y, ScaleX, ScaleY, Angle</td>
</tr>
<tr>
<td>6</td>
<td>X, Y, Scale, Angle, NewX, NewY</td>
</tr>
<tr>
<td>7</td>
<td>X, Y, ScaleX, ScaleY, Angle, NewX, NewY</td>
</tr>
</tbody>
</table>

For example...

A single argument would be treated as an angle:
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.
3.11.3 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.

\[
\begin{align*}
\text{src1}_x & \text{, src1}_y, \text{ dst1}_x & \text{, dst1}_y, \\
\text{src2}_x & \text{, src2}_y, \text{ dst2}_x & \text{, dst2}_y, \\
\text{src3}_x & \text{, src3}_y, \text{ dst3}_x & \text{, dst3}_y, \\
\text{src4}_x & \text{, src4}_y, \text{ dst4}_x & \text{, dst4}_y
\end{align*}
\]

For example:

```python
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'
    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)
```

3.11.4 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.

\[
\begin{align*}
\text{src1}_x & \text{, src1}_y, \text{ dst1}_x & \text{, dst1}_y, \\
\text{src2}_x & \text{, src2}_y, \text{ dst2}_x & \text{, dst2}_y, \\
\text{src3}_x & \text{, src3}_y, \text{ dst3}_x & \text{, dst3}_y
\end{align*}
\]

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)

---

### 3.11.5 Affine Projection

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

\[
\text{Scale}_x, \text{Rotate}_x, \text{Rotate}_y, \text{Scale}_y, \text{Translate}_x, \text{Translate}_y
\]

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 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:

```python
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)
```

### 3.12.1 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:

```python
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,
               height=100,
               background=Color('lightblue')) as img:
        draw.draw(img)
    img.save(filename='draw-arc.gif')
```
3.12.2 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:

```python
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`

3.12.3 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:

```python
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.circle((50, 50), # Center point
                 (25, 25)) # Perimeter point
with Image(width=100, height=100, background=Color('lightblue')) as image:
    draw(image)
```
3.12.4 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 borded:

```python
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.

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

3.12.5 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.

```python
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)
```
3.12.6 Ellipse

New in version 0.4.0.

Ellipse can be drawn by using the `ellipse()` method. Like drawing circles, the ellipse requires an 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:

```python
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:

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

3.12.7 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:

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

Or you can turn this diagonal line upside down:

```python
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:

```python
from wand.color import Color

with Color('red') as color:
    draw.fill_color = color
    draw.line((0, 0), image.size)
    draw(image)
```
3.12.8 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:

```python
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 across top-left to center
    draw.path_curve(to=(40, 0),
                    controls=[(10, -40), (30, -40)],
                    relative=True)
    # Continue curve across 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)
```
3.12.9 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:

```python
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`

3.12.10 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:

```python
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

3.12.11 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:

```python
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
### 3.12.12 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.

```python
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)
```

### 3.12.13 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`:

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

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

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

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

Note that the stroke 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`

### 3.12.14 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:

```python
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

3.12.15 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.

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

def draw_roi(ctx, roi_width, roi_height):
    """Let’s draw a blue box so we can identify what our region of interest 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 = ' '.join(wrap(mutable_message, columns))
            wrapped_width, _ = eval_metrics(mutable_message)
        if wrapped_width <= roi_width:

(continues on next page)
break
    if columns < 1:
        ctx.font_size -= 0.75  # Reduce fontsize
        mutable_message = text  # Restore original text
    else:
        break
    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')
3.13 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:

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

**Note:** You can’t write into `Image.metadata`.

3.13.1 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:

```python
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.

### 3.14 Layers

#### 3.14.1 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.

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

#### 3.14.2 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.

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

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

(continues on next page)
Another quick helper method to allow us to view/debug each frame.

```python
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:

```python
>>> 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.

```python
>>> with Image(filename='layers.gif') as img:
...     img.optimize_layers()
...     debug_layers(img, 'layers-optimized-layers.png')
Debugging to file layers-optimized-layers.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)
```

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 on the website, and has not special meaning outside of “no-data here.”*

### 3.14.3 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.

```python
>>> with Image(filename='layers.gif') as img:
...     img.optimize_transparency()
...     debug_layers(img, 'layers-optimized-transparent.png')
Debugging to file layers-optimized-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.

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

Debugging to file layers-optimized-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.

### 3.15 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](http://example.com))

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):

### 3.15.1 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:

```python
>>> 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:

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

Or slice it:

```python
>>> 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)>]
```

### 3.15.2 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.

### 3.15.3 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.
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 invisible to 'image' container.
...    image.sequence[2].negate()
...    image.save(filename='output.gif')  # Changes ignored.

>>> 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.
```

### 3.16 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.

### 3.17 CLI Reference

For users migrating old CLI scripts to python.

#### 3.17.1 CLI Operators to Wand Methods

This table maps ImageMagick’s CLI operators to Wand’s `Image` methods.
<table>
<thead>
<tr>
<th>CLI Operators</th>
<th>Wand Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>-adaptive-blur</td>
<td>wand.image.BaseImage.adaptive_blur()</td>
</tr>
<tr>
<td>-adaptive-resize</td>
<td>wand.image.BaseImage.adaptive_resize()</td>
</tr>
<tr>
<td>-adaptive-sharpen</td>
<td>wand.image.BaseImage.adaptive_sharpen()</td>
</tr>
<tr>
<td>-annotate</td>
<td>wand.image.BaseImage.annotate()</td>
</tr>
<tr>
<td>-append</td>
<td>wand.image.BaseImage.concat()</td>
</tr>
<tr>
<td>-auto-gamma</td>
<td>wand.image.BaseImage.auto_gamma()</td>
</tr>
<tr>
<td>-auto-level</td>
<td>wand.image.BaseImage.auto_level()</td>
</tr>
<tr>
<td>-auto-orient</td>
<td>wand.image.BaseImage.auto_orient()</td>
</tr>
<tr>
<td>-auto-threshold</td>
<td>wand.image.BaseImage.auto_threshold()</td>
</tr>
<tr>
<td>-black-threshold</td>
<td>wand.image.BaseImage.black_threshold()</td>
</tr>
<tr>
<td>-blue-shift</td>
<td>wand.image.BaseImage.blue_shift()</td>
</tr>
<tr>
<td>-blur</td>
<td>wand.image.BaseImage.blur()</td>
</tr>
<tr>
<td>-border</td>
<td>wand.image.BaseImage.border()</td>
</tr>
<tr>
<td>-brightness-contrast</td>
<td>wand.image.BaseImage.brightness_contrast()</td>
</tr>
<tr>
<td>-canny</td>
<td>wand.image.BaseImage.canny()</td>
</tr>
<tr>
<td>-caption</td>
<td>wand.image.BaseImage.caption()</td>
</tr>
<tr>
<td>-cdl</td>
<td>wand.image.BaseImage.color_decision_list()</td>
</tr>
<tr>
<td>-charcoal</td>
<td>wand.image.BaseImage.charcoal()</td>
</tr>
<tr>
<td>-chop</td>
<td>wand.image.BaseImage.chop()</td>
</tr>
<tr>
<td>-clahe</td>
<td>wand.image.BaseImage.clahe()</td>
</tr>
<tr>
<td>-clamp</td>
<td>wand.image.BaseImage.clamp()</td>
</tr>
<tr>
<td>-clut</td>
<td>wand.image.BaseImage.clut()</td>
</tr>
<tr>
<td>-coalesce</td>
<td>wand.image.BaseImage.coalesce()</td>
</tr>
<tr>
<td>-colorize</td>
<td>wand.image.BaseImage.colorize()</td>
</tr>
<tr>
<td>-colormap</td>
<td>wand.image.BaseImage.color_map()</td>
</tr>
<tr>
<td>-color-matrix</td>
<td>wand.image.BaseImage.color_matrix()</td>
</tr>
<tr>
<td>-colorspace</td>
<td>wand.image.BaseImage.transform_colorspace()</td>
</tr>
<tr>
<td>-compare</td>
<td>wand.image.BaseImage.compare()</td>
</tr>
<tr>
<td>-complex</td>
<td>wand.image.BaseImage.complex()</td>
</tr>
<tr>
<td>-composite</td>
<td>wand.image.BaseImage.composite()</td>
</tr>
<tr>
<td>-connected-components</td>
<td>wand.image.BaseImage.connected_components()</td>
</tr>
<tr>
<td>-contrast</td>
<td>wand.image.BaseImage.contrast()</td>
</tr>
<tr>
<td>-contrast-stretch</td>
<td>wand.image.BaseImage.contrast_stretch()</td>
</tr>
<tr>
<td>-crop</td>
<td>wand.image.BaseImage.crop()</td>
</tr>
<tr>
<td>-cycle</td>
<td>wand.image.BaseImage.cycle_color_map()</td>
</tr>
<tr>
<td>-deconstruct</td>
<td>wand.image.BaseImage.deconstruct()</td>
</tr>
<tr>
<td>-deskew</td>
<td>wand.image.BaseImage.deskew()</td>
</tr>
<tr>
<td>-despeckle</td>
<td>wand.image.BaseImage.despeckle()</td>
</tr>
<tr>
<td>-distort</td>
<td>wand.image.BaseImage.distort()</td>
</tr>
<tr>
<td>-edge</td>
<td>wand.image.BaseImage.edge()</td>
</tr>
<tr>
<td>-emboss</td>
<td>wand.image.BaseImage.emboss()</td>
</tr>
<tr>
<td>-enhance</td>
<td>wand.image.BaseImage.enhance()</td>
</tr>
<tr>
<td>-equalize</td>
<td>wand.image.BaseImage.equalize()</td>
</tr>
<tr>
<td>-evaluate</td>
<td>wand.image.BaseImage.evaluate()</td>
</tr>
<tr>
<td>-extent</td>
<td>wand.image.BaseImage.extent()</td>
</tr>
<tr>
<td>-features</td>
<td>wand.image.BaseImage.features()</td>
</tr>
<tr>
<td>-fft</td>
<td>wand.image.BaseImage.forward_fourier_transform()</td>
</tr>
<tr>
<td>-flip</td>
<td>wand.image.BaseImage.flip()</td>
</tr>
<tr>
<td>-flop</td>
<td>wand.image.BaseImage.flop()</td>
</tr>
<tr>
<td>-frame</td>
<td>wand.image.BaseImage.frame()</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>CLI Operators</th>
<th>Wand Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>-function</td>
<td><code>wand.image.BaseImage.function()</code></td>
</tr>
<tr>
<td>-fx</td>
<td><code>wand.image.BaseImage.fx()</code></td>
</tr>
<tr>
<td>-gamma</td>
<td><code>wand.image.BaseImage.gamma()</code></td>
</tr>
<tr>
<td>-gaussian-blur</td>
<td><code>wand.image.BaseImage.gaussian_blur()</code></td>
</tr>
<tr>
<td>-hald-clut</td>
<td><code>wand.image.BaseImage.hald_clut()</code></td>
</tr>
<tr>
<td>-hough-lines</td>
<td><code>wand.image.BaseImage.hough_lines()</code></td>
</tr>
<tr>
<td>-ift</td>
<td><code>wand.image.BaseImage.inverse_fourier_transform()</code></td>
</tr>
<tr>
<td>-implode</td>
<td><code>wand.image.BaseImage.implode()</code></td>
</tr>
<tr>
<td>-kuwahara</td>
<td><code>wand.image.BaseImage.kuwahara()</code></td>
</tr>
<tr>
<td>-lat</td>
<td><code>wand.image.BaseImage.adaptive_threshold()</code></td>
</tr>
<tr>
<td>-level</td>
<td><code>wand.image.BaseImage.level()</code></td>
</tr>
<tr>
<td>+level</td>
<td><code>wand.image.BaseImage.levelize()</code></td>
</tr>
<tr>
<td>-level-colors</td>
<td><code>wand.image.BaseImage.level_colors()</code></td>
</tr>
<tr>
<td>+level-colors</td>
<td><code>wand.image.BaseImage.levelize_colors()</code></td>
</tr>
<tr>
<td>-linear-stretch</td>
<td><code>wand.image.BaseImage.linear_stretch()</code></td>
</tr>
<tr>
<td>-liquid-rescale</td>
<td><code>wand.image.BaseImage.liquid_rescale()</code></td>
</tr>
<tr>
<td>-magnify</td>
<td><code>wand.image.BaseImage.magnify()</code></td>
</tr>
<tr>
<td>-mean-shift</td>
<td><code>wand.image.BaseImage.mean_shift()</code></td>
</tr>
<tr>
<td>-layers</td>
<td><code>wand.image.BaseImage.merge_layers()</code></td>
</tr>
<tr>
<td>-layers</td>
<td><code>wand.image.Image.compare_layers()</code></td>
</tr>
<tr>
<td>-layers</td>
<td><code>wand.image.BaseImage.optimize_layers()</code></td>
</tr>
<tr>
<td>-layers</td>
<td><code>wand.image.BaseImage.optimize_transparency()</code></td>
</tr>
<tr>
<td>-mode</td>
<td><code>wand.image.BaseImage.mode()</code></td>
</tr>
<tr>
<td>-modulate</td>
<td><code>wand.image.BaseImage.modulate()</code></td>
</tr>
<tr>
<td>-morphology</td>
<td><code>wand.image.BaseImage.morphology()</code></td>
</tr>
<tr>
<td>-motion-blur</td>
<td><code>wand.image.BaseImage.motion_blur()</code></td>
</tr>
<tr>
<td>-negate</td>
<td><code>wand.image.BaseImage.negate()</code></td>
</tr>
<tr>
<td>-noise</td>
<td><code>wand.image.BaseImage.noise()</code></td>
</tr>
<tr>
<td>-normalize</td>
<td><code>wand.image.BaseImage.normalize()</code></td>
</tr>
<tr>
<td>-paint</td>
<td><code>wand.image.BaseImage.oil_paint()</code></td>
</tr>
<tr>
<td>-opaque</td>
<td><code>wand.image.BaseImage.opaque_paint()</code></td>
</tr>
<tr>
<td>-ordered-dither</td>
<td><code>wand.image.BaseImage.ordered_dither()</code></td>
</tr>
<tr>
<td>-polaroid</td>
<td><code>wand.image.BaseImage.polaroid()</code></td>
</tr>
<tr>
<td>-polynomial</td>
<td><code>wand.image.BaseImage.polynomial()</code></td>
</tr>
<tr>
<td>-posterize</td>
<td><code>wand.image.BaseImage.posterize()</code></td>
</tr>
<tr>
<td>-quantize</td>
<td><code>wand.image.BaseImage.quantize()</code></td>
</tr>
<tr>
<td>-random-threshold</td>
<td><code>wand.image.BaseImage.random_threshold()</code></td>
</tr>
<tr>
<td>-range-threshold</td>
<td><code>wand.image.BaseImage.range_threshold()</code></td>
</tr>
<tr>
<td>-read-mask</td>
<td><code>wand.image.BaseImage.read_mask()</code></td>
</tr>
<tr>
<td>-remap</td>
<td><code>wand.image.BaseImage.remap()</code></td>
</tr>
<tr>
<td>-resample</td>
<td><code>wand.image.BaseImage.resample()</code></td>
</tr>
<tr>
<td>+repage</td>
<td><code>wand.image.BaseImage.reset_coords()</code></td>
</tr>
<tr>
<td>-resize</td>
<td><code>wand.image.BaseImage.resize()</code></td>
</tr>
<tr>
<td>-rotate</td>
<td><code>wand.image.BaseImage.rotate()</code></td>
</tr>
<tr>
<td>-radial_blur</td>
<td><code>wand.image.BaseImage.rotational_blur()</code></td>
</tr>
<tr>
<td>-sample</td>
<td><code>wand.image.BaseImage.sample()</code></td>
</tr>
<tr>
<td>-scale</td>
<td><code>wand.image.BaseImage.scale()</code></td>
</tr>
<tr>
<td>-selective-blur</td>
<td><code>wand.image.BaseImage.selective_blur()</code></td>
</tr>
<tr>
<td>-sepia-tone</td>
<td><code>wand.image.BaseImage.sepia_tone()</code></td>
</tr>
</tbody>
</table>
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>CLI Operators</th>
<th>Wand Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>-shade</td>
<td>wand.image.BaseImage.shade()</td>
</tr>
<tr>
<td>-shadow</td>
<td>wand.image.BaseImage.shadow()</td>
</tr>
<tr>
<td>-sharpen</td>
<td>wand.image.BaseImage.sharpen()</td>
</tr>
<tr>
<td>-shave</td>
<td>wand.image.BaseImage.shave()</td>
</tr>
<tr>
<td>-shear</td>
<td>wand.image.BaseImage.shear()</td>
</tr>
<tr>
<td>-sigmoidal-contrast</td>
<td>wand.image.BaseImage.sigmoidal_contrast()</td>
</tr>
<tr>
<td>-similarity-threshold</td>
<td>See wand.image.BaseImage.similarity()</td>
</tr>
<tr>
<td>-subimage-search</td>
<td>See wand.image.BaseImage.similarity()</td>
</tr>
<tr>
<td>-sketch</td>
<td>wand.image.BaseImage.sketch()</td>
</tr>
<tr>
<td>-smush</td>
<td>wand.image.BaseImage.smush()</td>
</tr>
<tr>
<td>-solarize</td>
<td>wand.image.BaseImage.solarize()</td>
</tr>
<tr>
<td>-sparse-color</td>
<td>wand.image.BaseImage.sparse_color()</td>
</tr>
<tr>
<td>-splice</td>
<td>wand.image.BaseImage.splice()</td>
</tr>
<tr>
<td>-spread</td>
<td>wand.image.BaseImage.spread()</td>
</tr>
<tr>
<td>-stegano</td>
<td>wand.image.BaseImage.stegano()</td>
</tr>
<tr>
<td>-stereo</td>
<td>wand.image.Image.stereogram()</td>
</tr>
<tr>
<td>-statistic</td>
<td>wand.image.BaseImage.statistic()</td>
</tr>
<tr>
<td>-strip</td>
<td>wand.image.BaseImage.strip()</td>
</tr>
<tr>
<td>-swirl</td>
<td>wand.image.BaseImage.swirl()</td>
</tr>
<tr>
<td>-texture</td>
<td>wand.image.BaseImage.texture()</td>
</tr>
<tr>
<td>-threshold</td>
<td>wand.image.BaseImage.threshold()</td>
</tr>
<tr>
<td>-thumbnail</td>
<td>wand.image.BaseImage.thumbnail()</td>
</tr>
<tr>
<td>-tint</td>
<td>wand.image.BaseImage.tint()</td>
</tr>
<tr>
<td>-transform</td>
<td>wand.image.BaseImage.transform()</td>
</tr>
<tr>
<td>-transparent-color</td>
<td>wand.image.BaseImage.transparent_color()</td>
</tr>
<tr>
<td>-transpose</td>
<td>wand.image.BaseImage.transpose()</td>
</tr>
<tr>
<td>-transverse</td>
<td>wand.image.BaseImage.transverse()</td>
</tr>
<tr>
<td>-treedepth</td>
<td>See wand.image.BaseImage.quantize()</td>
</tr>
<tr>
<td>-trim</td>
<td>wand.image.BaseImage.trim()</td>
</tr>
<tr>
<td>-unique-colors</td>
<td>wand.image.BaseImage.unique_colors()</td>
</tr>
<tr>
<td>-unsharp-mask</td>
<td>wand.image.BaseImage.unsharp_mask()</td>
</tr>
<tr>
<td>-vignette</td>
<td>wand.image.BaseImage.vignette()</td>
</tr>
<tr>
<td>-watermark</td>
<td>wand.image.BaseImage.watermark()</td>
</tr>
<tr>
<td>-wave</td>
<td>wand.image.BaseImage.wave()</td>
</tr>
<tr>
<td>-wavelet-denoise</td>
<td>wand.image.BaseImage.wavelet_denoise()</td>
</tr>
<tr>
<td>-white-threshold</td>
<td>wand.image.BaseImage.white_threshold()</td>
</tr>
<tr>
<td>-write-mask</td>
<td>wand.image.BaseImage.write_mask()</td>
</tr>
</tbody>
</table>

### 3.17.2 CLI Options to Wand Properties

This table list ImageMagick’s options, and maps them to Wand’s `Image` properties.

<table>
<thead>
<tr>
<th>CLI Options</th>
<th>Wand Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>-alpha</td>
<td>wand.image.BaseImage.alpha_channel</td>
</tr>
<tr>
<td>-antialias</td>
<td>wand.image.BaseImage.font_antialias</td>
</tr>
<tr>
<td>-antialias</td>
<td>wand.image.BaseImage.antialias</td>
</tr>
<tr>
<td>-background</td>
<td>wand.image.BaseImage.background_color</td>
</tr>
<tr>
<td>-blue-primary</td>
<td>wand.image.BaseImage.blue_primary</td>
</tr>
</tbody>
</table>
## Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>CLI Options</th>
<th>Wand Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>-bordercolor</td>
<td><code>wand.image.BaseImage.border_color</code></td>
</tr>
<tr>
<td>-colorspace</td>
<td><code>wand.image.BaseImage.colorspace</code></td>
</tr>
<tr>
<td>-compose</td>
<td><code>wand.image.BaseImage.compose</code></td>
</tr>
<tr>
<td>-compression</td>
<td><code>wand.image.BaseImage.compression</code></td>
</tr>
<tr>
<td>-delay</td>
<td><code>wand.image.BaseImage.ticks_per_second</code></td>
</tr>
<tr>
<td>-density</td>
<td><code>wand.image.BaseImage.resolution</code></td>
</tr>
<tr>
<td>-depth</td>
<td><code>wand.image.BaseImage.depth</code></td>
</tr>
<tr>
<td>-dispose</td>
<td><code>wand.image.BaseImage.dispose</code></td>
</tr>
<tr>
<td>-fill</td>
<td><code>wand.image.BaseImage.font_color</code></td>
</tr>
<tr>
<td>-font</td>
<td><code>wand.image.BaseImage.font</code></td>
</tr>
<tr>
<td>-format</td>
<td><code>wand.image.BaseImage.format</code></td>
</tr>
<tr>
<td>-fuzz</td>
<td><code>wand.image.BaseImage.fuzz</code></td>
</tr>
<tr>
<td>-gravity</td>
<td><code>wand.image.BaseImage.gravity</code></td>
</tr>
<tr>
<td>-green-primary</td>
<td><code>wand.image.BaseImage.green_primary</code></td>
</tr>
<tr>
<td>-intent</td>
<td><code>wand.image.BaseImage.rendering_intent</code></td>
</tr>
<tr>
<td>-interlace</td>
<td><code>wand.image.BaseImage.interlace_scheme</code></td>
</tr>
<tr>
<td>-interpolate</td>
<td><code>wand.image.BaseImage.interpolate_method</code></td>
</tr>
<tr>
<td>-loop</td>
<td><code>wand.image.BaseImage.loop</code></td>
</tr>
<tr>
<td>-mattecolor</td>
<td><code>wand.image.BaseImage.matte_color</code></td>
</tr>
<tr>
<td>-orientation</td>
<td><code>wand.image.BaseImage.orientation</code></td>
</tr>
<tr>
<td>-page</td>
<td><code>wand.image.BaseImage.page</code></td>
</tr>
<tr>
<td>-page</td>
<td><code>wand.image.BaseImage.page_height</code></td>
</tr>
<tr>
<td>-page</td>
<td><code>wand.image.BaseImage.page_width</code></td>
</tr>
<tr>
<td>-page</td>
<td><code>wand.image.BaseImage.page_x</code></td>
</tr>
<tr>
<td>-page</td>
<td><code>wand.image.BaseImage.page_y</code></td>
</tr>
<tr>
<td>-pointsize</td>
<td><code>wand.image.BaseImage.font_size</code></td>
</tr>
<tr>
<td>-quality</td>
<td><code>wand.image.BaseImage.compression_quality</code></td>
</tr>
<tr>
<td>-red-primary</td>
<td><code>wand.image.BaseImage.red_primary</code></td>
</tr>
<tr>
<td>-scene</td>
<td><code>wand.image.BaseImage.scene</code></td>
</tr>
<tr>
<td>-seed</td>
<td><code>wand.image.BaseImage.seed</code></td>
</tr>
<tr>
<td>-size</td>
<td><code>wand.image.BaseImage.height</code></td>
</tr>
<tr>
<td>-size</td>
<td><code>wand.image.BaseImage.width</code></td>
</tr>
<tr>
<td>-size</td>
<td><code>wand.image.BaseImage.size</code></td>
</tr>
<tr>
<td>-stroke</td>
<td><code>wand.image.BaseImage.stroke_color</code></td>
</tr>
<tr>
<td>-strokewidth</td>
<td><code>wand.image.BaseImage.stroke_width</code></td>
</tr>
<tr>
<td>-type</td>
<td><code>wand.image.BaseImage.type</code></td>
</tr>
<tr>
<td>-units</td>
<td><code>wand.image.BaseImage.units</code></td>
</tr>
<tr>
<td>-virtual-pixel</td>
<td><code>wand.image.BaseImage.virtual_pixel</code></td>
</tr>
<tr>
<td>-white-point</td>
<td><code>wand.image.BaseImage.white_point</code></td>
</tr>
</tbody>
</table>

### 3.18 Running tests

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

```bash
$ 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 `py.test` command. It provides more options:
$ pip install pytest
$ py.test

### 3.18.1 Skipping tests

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

$ py.test --skip-slow

By 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:

$ py.test --skip-pdf

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

$ py.test -k image

### 3.18.2 Using tox

Wand should be compatible with various Python implementations including CPython 2.6, 2.7, 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:

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

### 3.18.3 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
3.18.4 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

3.19 Roadmap

3.19.1 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:

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

- Global monkeypatcher which changes sys.modules:

```python
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.

3.20 Wand Changelog

3.20.1 0.5 series

Version 0.5.8

Unreleased

- Check WAND_MAGICK_LIBRARY_SUFFIX for additional library suffixes. [#436]
- Fixed MagickCompareImagesLayers() loading for ImageMagick-6 [#439]

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 behaviour 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_fontMetrics`. [#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]

### 3.20.2 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 YunWon]
- Added `Image.transform_colorspace()` quantize (MagickTransformImageColorsSpace()). [#152 by Adrian Jung, #262 by Jeong YunWon]
- Now ImageMagick DLL can be loaded on Windows even if its location is stored in the resitry. [#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](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`).
• **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.

- **Push & Pop**
  - 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.**
3.20.3 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.
• Renamed 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. [8857468a38015669dae903185fb328abdd717c0 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]

### 3.20.4 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**


• 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.compose()` 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`.

**3.20.5 0.1 series**

**Version 0.1.10**


• 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

• 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

• 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 \texttt{PixelWand} instead of the string representation for \texttt{Color} type.

**Version 0.1.5**


- Now \texttt{Image} can read Python file objects by \texttt{file} keyword argument.
- Now \texttt{Image.save()} method can write into Python file objects by \texttt{file} keyword argument.
- \texttt{Image.make_blob()}'s \texttt{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 \texttt{Image}.
- Row getter for \texttt{Image}.
- Mac compatibility.
- Windows compatibility.
- 64-bit processor compatibility.

**Version 0.1.2**


- \texttt{Image} implements iterable interface.
- Added \texttt{wand.color} module.
- Added the abstract base class of all Wand resource objects: \texttt{wand.resource.Resource}.
- \texttt{Image} implements slicing.
- Cropping \texttt{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 \texttt{Image.make_blob()} method.
- Added \texttt{blob} parameter into \texttt{Image} constructor.
- Added \texttt{Image.resize()} method.
- Added \texttt{Image.save()} method.
- Added \texttt{Image.clone()} method.
- Drewed the pretty logo picture (thanks to Hyojin Choi).
Version 0.1.0

Released on October 1, 2011. Very alpha version.

3.21 Talks and Presentations

3.21.1 Talks in 2012

• Lightning talk at Python Korea November 2012
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):

```python
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', 'extract', 'off', 'on', 'opaque', 'reset', 'set')

(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.

wand.image.CHANNELS = {'all_channels': 134217727, 'alpha': 16, 'black': 8, 'blue': 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.

The list of colorspace.

- '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.COMPRESS_METRICS = ('undefined', 'absolute', 'fuzz', 'mean_absolute', 'mean_error_per_pixel', 'mean_squared', ...
'normalized_cross_correlation', ...
'perceptual_hash', 'peak_absolute', 'peak_signal_to_noise_ratio', 'structural_similarity', 'structural_dissimilarity', ...
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', ...
'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', ...
See also:

Chapter 4. References
Changed in version 0.5.0: Support for ImageMagick-6 & ImageMagick-7

```
 wand.image.DISPOSE_TYPES = ('undefined', 'none', 'background', 'previous')
 (tuple) The list of BaselineImage.dispose types.
 New in version 0.5.0.
 wand.image.DISTORTION_METHODS = ('undefined', 'affine', 'affine_projection', 'scale_rotate_translate', 'perspective', ....
 (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'
 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', ....
 (tuple) The list of evaluation operators. Used by Image.evaluate() method.
   • 'undefined'
   • 'abs'
```
• 'add'
• 'addmodulus'
• 'and'
• 'cosine'
• 'divide'
• 'exponential'
• 'gaussiannoise'
• 'impulsenoise'
• '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', 'hann', 'cosine', 'spline', 'sentinel') The list of filter types. Used by Image.resample() and Image.resize() methods.
• 'undefined'
See also:

**ImageMagick Resize Filters**  Demonstrates the results of resampling images using the various resize filters and blur settings available in ImageMagick.

```
3.6. wand — Simple MagickWand API binding for Python
```
• 'arcsin'
• 'arctan'
• 'polynomial'
• 'sinusoid'

```
(wand.image.GRAVITY_TYPES = ('forget', 'north_west', 'north', 'north_east', 'west', 'center',
'gravity',
'forget',
'west',
'center',
'east',
'south_west',
'south',
'south_east',

New in version 0.3.0.
```

```
(wand.image.IMAGE_LAYER_METHOD = ('undefined', 'coalesce', 'compareany', 'compareclear', 'compareoverlay',
(methods),
'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')

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')

The list of interlace schemes.

- 'undefined'
- 'no'
- 'line'
- 'plane'
- 'partition'
- 'gif'
- 'jpeg'
- 'png'

New in version 0.5.2.

wand.image KERNEL_INFO_TYPES = ('undefined', 'unity', 'gaussian', 'dog', 'log', 'blur', 'comet', 'binomial', 'laplacian', 'sobel', 'ridges', 'convex_hull', 'thin_se', 'skeleton', 'chebyshev', 'manhattan', 'octagonal', 'euclidean', 'user_defined')

The list of builtin kernels.
• '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', 'distance', 'open', 'close', 'open_intensity', 'close_intensity', 'smooth', 'edgein', 'edgeout', 'edge', 'tophat', 'bottom_hat', 'hit_and_miss', 'thinning', 'thicken', 'voronoi', 'iterative_distance')

wand.image.NOISE_TYPES = ('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.

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.

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}
(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', '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.

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.
• 'horizontal_tile'
• 'vertical_tile'
• 'horizontal_tile_edge'
• 'vertical_tile_edge'
• 'checker_tile'

New in version 0.4.1.

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

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.

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) – hight 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.

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.

**alpha_channel** *(bool)*

Get state of image alpha channel. It can also be used to enable/disable alpha channel, but with different behavior 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.

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.

**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` consists 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`.

4.1. **wand** — Simple MagickWand API binding for Python

123
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.

```python
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, top=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.

`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.

**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.

**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.

**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.

**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.

**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 parameter, and ImageMagick 7 support.

**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.

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.

c chop (*width, height, x=0, y=0*)
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.

New in version 0.5.5.

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.

```python
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:

```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>
```

(continues on next page)
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:

```python
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:

```python
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:

\[
\begin{align*}
red' &= 1.0 \times red + 0.0 \times green + 0.0 \times blue \\
green' &= 0.0 \times red + 1.0 \times green + 0.0 \times blue \\
blue' &= 0.0 \times red + 0.0 \times green + 1.0 \times blue
\end{align*}
\]

For RGB colorspace images, the rows & columns are laid out as:
Or for a CMYK colorspace image:

<table>
<thead>
<tr>
<th>Cyan'</th>
<th>Yellow'</th>
<th>Magenta'</th>
<th>Black'</th>
<th>Alpha'</th>
<th>Offset'</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

See color-matrix for examples.

**Parameters**

- `matrix (collections.abc.Sequence) – 2D List of doubles.

New in version 0.5.3.

**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.

**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.

**colors (numbers.Integral) Count of unique colors used within the image. This is READ ONLY property.**

New in version 0.5.3.

**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.

**compare (image, metric='undefined', highlight=None, lowlight=None)**

Compares an image to a reconstructed image.

Set `fuzz` property to adjust pixel-compare thresholds.

For example:

```python
from wand.image import Image

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

(continues on next page)
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.Integral*)

Return type *tuple*

New in version 0.4.3.

Changed in version 0.5.3: Added support for `highlight` & `lowlight`.

```python
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()`

```python
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.
**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.

**compression**
(base_string) 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.

**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** *(connectivity=4, area_threshold=None, mean_color=False, keep=None, remove=None)*
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.

```python
from wand.image import Image
with Image(filename='objects.gif') as img:
    objects = img.connected_components()
for cc_obj in objects:
    print("{}: {}.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:

```python
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

- **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.

- **area_threshold** (*basestring*) – Optional argument to exclude objects under an area size.

- **mean_color** (*bool*) – Optional argument. Replace object color with mean color of the source image.

- **keep** (*basestring*) – Comma separated list of object IDs to isolate, the reset are converted to transparent.

- **remove** (*basestring*) – Comma separated list of object IDs to ignore, and convert to transparent.

**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.

**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.

**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.
**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.

**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.

New in version 0.5.0.

**despeckle**

Applies filter to reduce noise in image.

New in version 0.5.0.

**dirty** = None

(bool) Whether the image is changed or not.

**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.

```python
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')
```

![Image example](image_example.png)
Use \texttt{virtual\_pixel}, \texttt{background\_color}, and \texttt{matte\_color} properties to control the behavior of pixels rendered outside of the image boundaries.

Use \texttt{interpolate\_method} to control how images scale-up.

Distortion viewport, and scale, can be defined by using \texttt{Image.artifacts} dictionary. For example:

\begin{verbatim}
img.artifacts['distort:viewport'] = '44x44+15+0'
img.artifacts['distort:scale'] = '10'
\end{verbatim}

**Parameters**

- \texttt{method} (\texttt{basestring}) – Distortion method name from \texttt{DISTORTION\_METHODS}

- \texttt{arguments} (\texttt{collections.abc.Sequence}) – List of distorting float arguments unique to distortion method

- \texttt{best\_fit} (\texttt{bool}) – Attempt to resize resulting image fit distortion. Defaults False

New in version 0.4.1.

\texttt{edge} (\texttt{radius}=0.0)

Applies convolution filter to detect edges.

**Parameters** \texttt{radius} (\texttt{numbers.Real}) – aperture of detection filter.

New in version 0.5.0.

\texttt{emboss} (\texttt{radius}=0.0, \texttt{sigma}=0.0)

Applies convolution filter against Gaussians filter.

**Note:** The \texttt{radius} value should be larger than \texttt{sigma} for best results.

**Parameters**

- \texttt{radius} (\texttt{numbers.Real}) – filter aperture size.

- \texttt{sigma} (\texttt{numbers.Real}) – standard deviation.

New in version 0.5.0.

\texttt{enhance()}

Applies digital filter to reduce noise.

New in version 0.5.0.

\texttt{equalize(channel=None)}

Equalizes the image histogram

**Parameters** \texttt{channel} (\texttt{basestring}) – Optional channel. See \texttt{CHANNELS}.
New in version 0.3.10.

Changed in version 0.5.5: Added optional \texttt{channel} argument.

\begin{verbatim}
\textbf{evaluate (operator=None, value=0.0, channel=None)}
\end{verbatim}

Apply arithmetic, relational, or logical expression to an image.

Percent values must be calculated against the quantum range of the image:

\begin{verbatim}
fifty\_percent = img.quantum\_range * 0.5
img.evaluate(operator='set', value=fifty\_percent)
\end{verbatim}

\textbf{Parameters}

- \texttt{operator (EVALUATE\_OPS)} – Type of operation to calculate
- \texttt{value (numbers.Real)} – Number to calculate with \texttt{operator}
- \texttt{channel (CHANNELS)} – Optional channel to apply operation on.

\textbf{Raises}

- \texttt{TypeError} – When \texttt{value} is not numeric.
- \texttt{ValueError} – When \texttt{operator}, or \texttt{channel} are not defined in constants.

New in version 0.4.1.

\begin{verbatim}
\textbf{export\_pixels (x=0, y=0, width=None, height=None, channel\_map='RGBA', storage='char')}
\end{verbatim}

Export pixel data from a raster image to a list of values.

The \texttt{channel\_map} tells ImageMagick which color channels to export, and what order they should be written as – per pixel. Valid entries for \texttt{channel\_map} are:

- \texttt{'R'} - Red channel
- \texttt{'G'} - Green channel
- \texttt{'B'} - Blue channel
- \texttt{'A'} - Alpha channel (0 is transparent)
- \texttt{'O'} - Alpha channel (0 is opaque)
- \texttt{'C'} - Cyan channel
- \texttt{'Y'} - Yellow channel
- \texttt{'M'} - Magenta channel
- \texttt{'K'} - Black channel
- \texttt{'I'} - Intensity channel (only for grayscale)
- \texttt{'P'} - Padding

See \texttt{STORAGE\_TYPES} for a list of valid \texttt{storage} options. This tells ImageMagick what type of data it should calculate & write to. For example; a storage type of \texttt{'char'} will write a 8-bit value between 0 ~ 255, a storage type of \texttt{'short'} will write a 16-bit value between 0 ~ 65535, and a \texttt{'integer'} will write a 32-bit value between 0 ~ 4294967295.

\textbf{Note}: By default, the entire image will be exported as \texttt{'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 \((width=None, height=None, x=0, y=0)\)
extends the image as defined by the geometry, gravity, and wand background color. Set the \((x,y)\) offset of the geometry to move the original wand relative to the extended wand.

Parameters

- \( width \) (numbers.Integral) – the \( width \) of the extended image. default is the \( width \) of the image.
- \( height \) (numbers.Integral) – the \( height \) of the extended image. default is the \( height \) of the image.
- \( x \) (numbers.Integral) – the \( x \) offset of the extended image. default is 0
- \( y \) (numbers.Integral) – the \( y \) offset of the extended image. default is 0

New in version 0.4.5.

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.

```python
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.

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.

New in version 0.3.0.

**font** *(wand.font.Font)*

- The current font options.

**font_antialias**

- Deprecated since version 0.5.0: Use `antialias` instead.

**font_path** *(basestring)*

- The path of the current font. It also can be set.

**font_size** *(numbers.Real)*

- The font size. It also can be set.

**format** *(basestring)*

- The image format.

If you want to convert the image format, just reset this property:

```python
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).

```python
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

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.

**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](http://www.imagemagick.org/script/fx.php)

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.

**gamma** (**adjustment_value**, **channel=None**) Gamma correct image.

Specific color channels can be correct individual. Typical values range between 0.8 and 2.3.

Parameters

- **adjustment_value** (numbers.Real) – value to adjust gamma level
- **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.

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.

```
 gravity  (basestring)
```

The text placement gravity used when annotating with text. It’s a string from `GRAVITY_TYPES` list. It also can be set.

```
 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.

```
 height  (numbers.Integral)
```

The height of this image.

```
 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`.

```python
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.

**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.

**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.

interlace_scheme
(basestring) The interlace used by the image. See INTERLACE_TYPES.

New in version 0.5.2.

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.

**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.

```python
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.

**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.

**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.

**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, or less then, 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.
**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
• \texttt{delta\_x(numbers.Real)} – maximum seam transversal step. 0 means straight seams. default is 0

• \texttt{rigidity(numbers.Real)} – introduce a bias for non-straight seams. default is 0

\textbf{Raises} \texttt{wand.exceptions.MissingDelegateError} – when ImageMagick isn’t configured \texttt{--with-lqr} option.

\textbf{Note:} This feature requires ImageMagick to be configured \texttt{--with-lqr} option. Or it will raise \texttt{MissingDelegateError}:

\textbf{See also:}

\texttt{Seam carving — Wikipedia} The article which explains what seam carving is on Wikipedia.

\texttt{local\_contrast(radius=10, strength=12.5)}

Increase light-dark transitions within image.

\textbf{Warning:} This class method is only available with ImageMagick 6.9.3, or greater.

\textbf{Parameters}

• \texttt{radius(numbers.Real)} – The size of the Gaussian operator. Default value is 10.0.

• \texttt{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.

\texttt{loop(numbers.Integral)} Number of frame iterations. A value of 0 will loop forever.

\texttt{magnify()}

Quickly double an image in size. This is a convenience method. Use \texttt{resize()}, \texttt{resample()}, or \texttt{sample()} for more control.

New in version 0.5.5.

\texttt{matte\_color(wand.color.Color)} The color value of the matte channel. This can also be set.

New in version 0.4.1.

\texttt{maxima(numbers.Real)} The maximum quantum value within the image. Value between 0.0 and \texttt{quantum\_range}

\textbf{Tip:} If you want both \texttt{maxima} \& \texttt{minima}, it would be faster to call \texttt{range\_channel()} directly.

New in version 0.5.3.

\texttt{mean(numbers.Real)} The mean of the image, and have a value between 0.0 and \texttt{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.

```python
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.
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.

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:
```python
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.

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.

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.

    Note: This will only affect GIF image formats.

    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 formats.

    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.

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

**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.

**orientation** *(basestring)* The image orientation. It’s a string from `ORIENTATION_TYPES` list. It also can be set.

New in version 0.3.0.

**page**

The dimensions and offset of this Wand’s page as a 4-tuple: *(width, height, x, y)*.

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.

New in version 0.4.3.

**page_height** *(numbers.Integral)* The height of the page for this wand.

New in version 0.4.3.

**page_width** *(numbers.Integral)* The width of the page for this wand.

New in version 0.4.3.

**page_x** *(numbers.Integral)* The X-offset of the page for this wand.
New in version 0.4.3.

**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.

```python
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 percent 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.

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:

```python
with Image(filename='rose:') as img:
    img.polynomial(arguments=[0.5, 1.0])
```

The output image will be calculated as:

\[
\text{output} = 0.5 \times \text{image}^{1.0}
\]

This can work on multiple images in a sequence by calculating across each frame in the image stack.

```python
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:

\[
\text{output} = 0.5 \times \text{frame}^{1.0} + 0.25 \times \text{frame}^{2.25}
\]

**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, treedepth, dither, measure_error`)  
`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 number of colors.
- `colorspace_type` ([`basestring`]) – colorspace_type. available value can be found in the `COLORSPACE_TYPES`
- `treedepth` ([`numbers.Integral`]) – normally, this integer value is zero or one. a zero or one tells `quantize()` to choose a optimal tree depth of $\log_4(\text{number_colors})$. a tree of this depth generally allows the best representation of the reference image with the least amount of memory and the fastest computational speed. in some cases, such as an image with low color dispersion (a few number of colors), a value other than $\log_4(\text{number_colors})$ is required. to expand the color tree completely, use a value of 8
• **dither** (bool) – a value other than zero distributes the difference between an original image and the corresponding color reduced algorithm to neighboring pixels along a Hilbert curve

• **measure_error** (bool) – a value other than zero measures the difference between the original and quantized images. This difference is the total quantization error. The error is computed by summing over all pixels in an image the distance squared in RGB space between each reference pixel value and its quantized value

New in version 0.4.2.

**quantum_range**

(int) 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 operates 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.

```python
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:

```python
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:

```python
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.

**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.

**remap** *(affinity=None, method='no')*

Rebuild image palette with closest color from given affinity image.

**Parameters**

- **affinity** *(BaseImage)* – reference image.
- **method** *(basestring)* – dither method. See *DITHER_METHODS*. Default is *'no'* dither.

New in version 0.5.3.

**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.

**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.

**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.

**resolution**
(tuple) Resolution of this image.

New in version 0.3.0.

**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.

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.

**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.

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.

scene (numbers.Integral) The scene number of the current frame within an animated image.

New in version 0.5.4.

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)

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.

```
sephia_tone (threshold=0.8)
```

Creates a Sepia Tone special effect similar to a darkroom chemical toning.

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 `SingleImages` 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.

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 \texttt{radius} is larger than \texttt{sigma}.

Defaults values of zero will have ImageMagick attempt to auto-select suitable values.

Parameters

- \texttt{radius} (\texttt{numbers.Real}) – size of gaussian aperture.
- \texttt{sigma} (\texttt{numbers.Real}) – Standard deviation of the gaussian filter.
- \texttt{channel} (\texttt{basestring}) – Optional color channel to target. See \texttt{CHANNELS}.

New in version 0.5.0.

Changed in version 0.5.5: Added \texttt{channel} argument.

\texttt{shave (columns=0, rows=0)}

Remove pixels from the edges.

Parameters

- \texttt{columns} (\texttt{numbers.Integral}) – amount to shave off both sides of the x-axis.
- \texttt{rows} (\texttt{numbers.Integral}) – amount to shave off both sides of the y-axis.

New in version 0.5.0.

\texttt{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

- \texttt{background} (\texttt{wand.color.Color}) – Color to fill the void created by shearing the image.
- \texttt{x} (\texttt{numbers.Real}) – Slide the image along the X-axis.
- \texttt{y} (\texttt{numbers.Real}) – Slide the image along the Y-axis.

New in version 0.5.4.

\texttt{sigmoidal_contrast (sharpen=True, strength=0.0, midpoint=0.0, channel=None)}

Modifies the contrast of the image by applying non-linear sigmoidal algorithm.

```python
with Image(filename='photo.jpg') as img:
    img.sigmoidal_contrast(sharpen=True, strength=3, midpoint=0.65 * img.quantum_range)
```

Parameters

- \texttt{sharpen} (\texttt{bool}) – Increase the contrast when True (default), else reduces contrast.
- \texttt{strength} (\texttt{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.
- \texttt{midpoint} (\texttt{numbers.Real}) – Normalized value between 0.0 & \texttt{quantum_range}
- \texttt{channel} (\texttt{basestring}) – Optional color channel to target. See \texttt{CHANNELS}.

New in version 0.5.4.

Changed in version 0.5.5: Added \texttt{channel} argument.
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.

```python
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')
        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.

**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.
**Wand Documentation, Release 0.5.8**

**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`.

**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.

**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.

**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:

```python
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_colors('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:

```python
from wand.image import Image, CHANNELS

with Image(filename='input.png') as img:
    colors = {
        img[50, 50]: (50, 50),
        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_colors('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)

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.

#### 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`.

**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.

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.

```python
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.

**Parameters**
- **degree** *(numbers.Real)* – Defines the amount of pixels to be affected. Value between -360.0 and 360.0.
- **method** *(basestring)* – Controls interpolation of the affected 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.

```python
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

**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.

**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.

**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%**  **scale-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.

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:

```python
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.

**transform colorspace**(colorspace_type)

Transform image’s colorspace.

Parameters **colorspace_type** (basestring) – colorspace_type. available value can be found in the COLORSSPACE_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.Integral) – 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.

**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)

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.Integral)* – Defines how much tolerance is acceptable to consider two colors as the same. Value can be between 0, and `quantum_range`.

  Changed in version 0.5.2: The `color` parameter may except color-compliant strings.

  Changed in version 0.3.0: Optional `color` and `fuzz` parameters.

  New in version 0.2.1.

**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.

**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.

  **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.

  **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.
virtual_pixel
(basestring) The virtual pixel of image. This can also be set with a value from VIRTUAL_PIXEL_METHOD ... versionadded:: 0.4.1

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.

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.

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_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.

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.

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.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.

class wand.image.Image(image=None, blob=None, file=None, filename=None, format=None, width=None, height=None, depth=None, background=None, resolution=None, pseudo=None, colorspace=None, units=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 COLORSSPACE_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* parameter to specify canvas color when reading in image.

Changed in version 0.5.7: Added the *colorspace & units* parameter.

**[left:right, top:bottom]**

Crops the image by its left, right, top and bottom, and then returns the cropped one.

```python
with img[100:200, 150:300] as cropped:
    # manipulated the cropped image
pass
```

Like other subscriptable objects, default is 0 or its width/height:

```python
img[:, :]  #--> just clone
img[:, 100, 200:]  #--> equivalent to img[0:100, 200:img.height]
```

Negative integers count from the end (width/height):

**Returns** the cropped image

**Return type** Image

New in version 0.1.2.

**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 consists 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.

```python
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.

```python
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.

```python
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.

destroy()
Manually remove `SingleImage`'s in the `Sequence`, allowing it to be properly garbage collected after using a `with` `Image()` context manager.

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.

```python
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", "I", 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:

```python
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.

**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.

**mimetype** *(basestring)* The MIME type of the image e.g. 'image/jpeg', 'image/png'.

New in version 0.1.7.

**classmethod ping** *(file=None, filename=None, blob=None, resolution=None, format=None)*

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.

```python
profiles = None
```

*(ProfileDict)* The mapping of image profiles.

  New in version 0.5.1.

```python
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.

```python
read(file=None, filename=None, blob=None, resolution=None, units=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.

• **resolution** *(collections.abc.Sequence, numbers.Integral)* – set a resolution value (DPI), useful for vectorial formats (like PDF)

• **units** *(basestring)* – used with resolution, can either be 'pixelperinch', or 'pixelpercentimeter'.

  New in version 0.3.0.

  Changed in version 0.5.7: Added units parameter.

```python
save(file=None, filename=None)
```

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

  New in version 0.1.5: The file parameter.

  New in version 0.1.1.

```python
classmethod stereogram(left, right)
```

Create a new stereogram image from two existing images.

**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.

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.

close ()
Clones the same iterator.

next (x=None)
Return the next item from the iterator. When exhausted, raise StopIteration

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.

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.manipulative(function)
Mark the operation manipulating itself instead of returning new one.

```python
class wand.image.ArtifactTree(image)
    Splay tree to map image artifacts. Values defined here are intended to be used elsewhere, 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.

```python
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:

```python
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 img.profiles['xmp']
```

See also:

- Embedded Image Profiles for a list of supported profiles.

New in version 0.5.1.

```python
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.

- **area** *(numbers.Real)* Quantity of pixels that make-up the objects shape.
- **center_x** *(numbers.Real)* X offset of objects centroid.
- **center_y** *(numbers.Real)* Y offset of objects centroid.
- **centroid** *(tuple(center_x, center_y))* Center of object.
clone_from_cc_object_info(cc_object)
    Copy data from CCObjectInfo.

definitions:

    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.
    offset
        (tuple (left, top)) Position of objects minimum bounding rectangle.
    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.

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.

```python
== (other)
Equality operator.

    Param other a color another one
    Type color Color
    Returns True only if two images equal.
    Rtype bool
```

### alpha
(numbers.Real) Alpha value, from 0.0 to 1.0.

### alpha_int8
(numbers.Integral) Alpha value as 8bit integer which is a common style. From 0 to 255.
  New in version 0.3.0.

### alpha_quantum
(numbers.Integral) Alpha value. Scale depends on QUANTUM_DEPTH.
  New in version 0.3.0.

### black
(numbers.Real) Black, or 'K', color channel in CMYK colorspace. Unused by RGB colorspace.
  New in version 0.5.1.

### black_int8
(numbers.Integral) Black value as 8bit integer which is a common style. From 0 to 255.
  New in version 0.5.1.

### black_quantum
(numbers.Integral) Black. Scale depends on QUANTUM_DEPTH.
  New in version 0.5.1.

### blue
(numbers.Real) Blue, from 0.0 to 1.0.

### blue_int8
(numbers.Integral) Blue as 8bit integer which is a common style. From 0 to 255.
  New in version 0.3.0.

### blue_quantum
(numbers.Integral) Blue. Scale depends on QUANTUM_DEPTH.
  New in version 0.3.0.

```python
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.

cyan
(numbers.Real) Cyan color channel in CMYK colorspace. Unused by RGB colorspace.
New in version 0.5.1.

cyan_int8
(numbers.Integral) Cyan value as 8bit integer which is a common style. From 0 to 255.
New in version 0.5.1.

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.

green
(numbers.Real) Green, from 0.0 to 1.0.

green_int8
(numbers.Integral) Green as 8bit integer which is a common style. From 0 to 255.
New in version 0.3.0.

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.

magenta
(numbers.Real) Magenta color channel in CMYK colorspace. Unused by RGB colorspace.

New in version 0.5.1.

magenta_int8
(numbers.Integral) Magenta value as 8bit integer which is a common style. From 0 to 255.

New in version 0.5.1.

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.

red
(numbers.Real) Red, from 0.0 to 1.0.

red_int8
(numbers.Integral) Red as 8bit integer which is a common style. From 0 to 255.

New in version 0.3.0.

red_quantum
(numbers.Integral) Red. Scale depends on QUANTUM_DEPTH.

New in version 0.3.0.

string
(basestring) The string representation of the color.

yellow
(numbers.Real) Yellow color channel in CMYK colorspace. Unused by RGB colorspace.

New in version 0.5.1.

yellow_int8
(numbers.Integral) Yellow as 8bit integer which is a common style. From 0 to 255.

New in version 0.5.1.

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.

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 antialiasing. 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
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 & stroke_width parameters.

antialias
(bool) Whether to apply antialiasing (True) or not (False).

color
(wand.color.Color) The font color.

path
(basestring) The path of font file.

size
(numbers.Real) The font size in pixels.

stroke_color
(wand.color.Color) The stroke color.

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.

```python
wand.drawing.CLIP_PATH_UNITS = ('undefined_path_units', 'user_space', 'user_space_on_use',
(collections.abc.Sequence) The list of clip path units
    • 'undefined_path_units'
    • 'user_space'
    • 'user_space_on_use'
    • 'object_bounding_box'

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',
(collections.abc.Sequence) The attribute names of font metrics.

wand.drawing.GRAVITY_TYPES = ('forget', 'north_west', 'north', 'north_east', 'west', 'center',
(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'
```
wand.drawing.PAINT_METHOD_TYPES = ('undefined', 'point', 'replace', 'floodfill', 'filltoborder', 'reset')
(collections.abc.Sequence) The list of paint method types.

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

wand.drawing.STYLE_TYPES = ('undefined', 'normal', 'italic', 'oblique', 'any')
(collections.abc.Sequence) The list of style types for fonts

wand.drawing.TEXT_ALIGN_TYPES = ('undefined', 'left', 'center', 'right')
(collections.abc.Sequence) The list of text align types.
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'

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:

```python
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.

```
border_color
```

(What) 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.

**clip_path**
(basestring) The current clip path. It also can be set.

  New in version 0.4.0.

**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.

**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=None, y=None, 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 \((\text{numbers.Real})\) – the row offset of the composited drawing source
• width \((\text{numbers.Real})\) – the total columns to include in the composited source
• height \((\text{numbers.Real})\) – the total rows to include in the composited source

New in version 0.4.0.

draw \((\text{image})\)
Renders the current drawing into the \text{image}. You can simply call \text{Drawing} instance rather than calling this method. That means the following code which calls \text{Drawing} object itself:

\[
drawing(\text{image})
\]

is equivalent to the following code which calls \text{draw()} method:

\[
drawing\text{.draw(}\text{image})
\]

Parameters \text{image} \((\text{BaseImage})\) – the image to be drawn

ellipse \((\text{origin, radius, rotation}=(0,360))\)
Draws a ellipse at \text{origin} with independent x & y \text{radius}. Ellipse can be partial by setting start & end \text{rotation}.

Parameters

• \text{origin} \((\text{collections.abc.Sequence})\) – \((\text{Real, numbers.Real})\) pair which represents origin x and y of circle
• \text{radius} \((\text{collections.abc.Sequence})\) – \((\text{Real, numbers.Real})\) pair which represents radius x and radius y of circle
• \text{rotation} \((\text{collections.abc.Sequence})\) – \((\text{Real, numbers.Real})\) pair which represents start and end of ellipse. Default \((0,360)\)

New in version 0.4.0.

fill_color
\((\text{Color})\) The current color to fill. It also can be set.

fill_opacity
\((\text{Real})\) The current fill opacity. It also can be set.

New in version 0.4.0.

fill_rule
\((\text{basestring})\) The current fill rule. It can also be set. It’s a string value from \text{FILL_RULE_TYPES} list.

New in version 0.4.0.

font
\((\text{basestring})\) The current font name. It also can be set.

font_family
\((\text{basestring})\) The current font family. It also can be set.

New in version 0.4.0.

font_resolution
\((\text{Sequence})\) The current font resolution. It also can be set.

New in version 0.4.0.
**font_size**
(numbers.Real) The font size. It also can be set.

**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.

**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.

**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 from the given text.

   Parameters

   • **image** *(BaseImage)* – the image to be drawn
   • **text** *(basestring)* – the text string for get font metrics.
   • **multiline** *(boolean)* – text is multiline or not

**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=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 has been replace by \texttt{alpha()} in ImageMagick version 7. An \texttt{AttributeError} will be raised if attempting to call on a library without \texttt{DrawMatte} support.

New in version 0.4.0.

\texttt{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 \texttt{Drawing.fill\_opacity} & \texttt{Drawing.stroke\_opacity}

New in version 0.4.0.

\texttt{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.

\texttt{path\_curve(to=None, controls=None, smooth=False, relative=False)}

Draws a cubic Bezier curve from the current point to given \texttt{to (x,y)} coordinate using \texttt{controls} points at the beginning and the end of the curve. If \texttt{smooth} is set to True, only one \texttt{controls} is expected and the previous control is used, else two pair of coordinates are expected to define the control points. The \texttt{to} coordinate then becomes the new current point.

Parameters

\begin{itemize}
  \item \texttt{to (collections.abc.Sequence)} -- (Real, numbers.Real) pair which represents coordinates to draw to
  \item \texttt{controls (collections.abc.Sequence)} -- (Real, numbers.Real) coordinate to used to influence curve
  \item \texttt{smooth (bool)} -- bool assume last defined control coordinate
  \item \texttt{relative (bool)} -- treat given coordinates as relative to current point
\end{itemize}

New in version 0.4.0.

\texttt{path\_curve\_to\_quadratic\_bezier(to=None, control=None, smooth=False, relative=False)}

Draws a quadratic Bezier curve from the current point to given \texttt{to} coordinate. The control point is assumed to be the reflection of the control point on the previous command if \texttt{smooth} is True, else a pair of \texttt{control} coordinates must be given. Each coordinates can be relative, or absolute, to the current point by setting the \texttt{relative} flag. The \texttt{to} coordinate then becomes the new current point, and the \texttt{control} coordinate will be assumed when called again when \texttt{smooth} is set to true.

Parameters

\begin{itemize}
  \item \texttt{to (collections.abc.Sequence)} -- (Real, numbers.Real) pair which represents coordinates to draw to
  \item \texttt{control (collections.abc.Sequence)} -- (Real, numbers.Real) coordinate to used to influence curve
  \item \texttt{smooth (bool)} -- assume last defined control coordinate
  \item \texttt{relative (bool)} -- treat given coordinates as relative to current point
\end{itemize}

New in version 0.4.0.

\texttt{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 \texttt{to} coordinates. The \texttt{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 automatically to satisfy the constraints imposed by the other parameters.

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()**
Declarations 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`

```python
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 different styles without repeatedly setting the similar `fill_color` & `stroke_color` properties.

For example:

```python
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"
stroke "ORANGE"
push graphic-context
fill "RED"
text x1,y1 "this is RED with ORANGE outline"
push 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`.  

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
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*)

Applies the specified rotation to the current coordinate space.

**Parameters**

degree (*Real*) – degree to rotate

New in version 0.4.0.

**scale** (*x=None, y=None*)

Adjusts the scaling factor to apply in the horizontal and vertical directions to the current coordinate space.

**Parameters**

- **x** (*Real*) – Horizontal scale factor
- **y** (*Real*) – Vertical scale factor

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.

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.

stroke_color (Color) The current color of stroke. It also can be set.

New in version 0.3.3.

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.

stroke_dash_offset (numbers.Real) The stroke dash offset. It also can be set.

New in version 0.4.0.

stroke_line_cap (basestring) The stroke line cap. It also can be set.

New in version 0.4.0.

stroke_line_join (basestring) The stroke line join. It also can be set.

New in version 0.4.0.
stroke_miter_limit
(Integral) The current miter limit. It also can be set.
New in version 0.4.0.

stroke_opacity
(Real) The current stroke opacity. It also can be set.
New in version 0.4.0.

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

text_alignment
(basestring) The current text alignment setting. It’s a string value from TEXT_ALIGN_TYPES list. It also can be set.

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.

text_decoration
(basestring) The text decoration setting, a string from TEXT_DECORATION_TYPES list. It also can be set.

text_direction
(basestring) The text direction setting, a string from TEXT_DIRECTION_TYPES list. It also can be set.

text_encoding
(basestring) The internally used text encoding setting. Although it also can be set, but it’s not encouraged.

text_interline_spacing
(numbers.Real) The setting of the text line spacing. It also can be set.

text_interword_spacing
(numbers.Real) The setting of the word spacing. It also can be set.

text_kerning
(numbers.Real) The setting of the text kerning. It also can be set.

text_under_color
(Color) The color of a background rectangle to place under text annotations. It also can be set.

translate (x=None, y=None)
  Applies a translation to the current coordinate system which moves the coordinate system origin to the specified coordinate.

Parameters
- x (Real) – Skew horizontal direction
• \textbf{y (Real)} – Skew vertical direction

New in version 0.4.0.

\textbf{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.

\textbf{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.

\textbf{Parameters}

• \textbf{left (Integral)} – the left most point of the viewbox.
• \textbf{top (Integral)} – the top most point of the viewbox.
• \textbf{right (Integral)} – the right most point of the viewbox.
• \textbf{bottom (Integral)} – the bottom most point of the viewbox.

New in version 0.4.0.

\textbf{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.

\textbf{ascender}

Alias for field number 2

\textbf{character\_height}

Alias for field number 1

\textbf{character\_width}

Alias for field number 0

\textbf{descender}

Alias for field number 3

\textbf{maximum\_horizontal\_advance}

Alias for field number 6

\textbf{text\_height}

Alias for field number 5

\textbf{text\_width}

Alias for field number 4

\textbf{x}

Alias for field number 11

\textbf{x1}

Alias for field number 7

\textbf{x2}

Alias for field number 9

\textbf{y}

Alias for field number 12
y1
Alias for field number 8

y2
Alias for field number 10

4.1.5 `wand.sequence` — Sequences

New in version 0.3.0.

```python
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.

```python
append(image)
    S.append(value) – append value to the end of the sequence
```

```python
current_index
    (numbers.Integral) The current index of its internal iterator.
```

**Note:** It’s only for internal use.

```python
extend(images, offset=None)
    S.extend(iterable) – extend sequence by appending elements from the iterable
```

```python
index_context(index)
    Scoped setter of `current_index`. Should be used for with statement e.g.:

    ```python
    with image.sequence.index_context(3):
        print(image.size)
    ```

**Note:** It’s only for internal use.

```python
insert(index, image)
    S.insert(index, value) – insert value before index
```

```python
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.

```python
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.

```python
container = None
    (wand.image.Image) The container image.
```
delay
(numbers.Integral) The delay to pause before display the next image (in the sequence of its container). It’s hundredths of a second.

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.

wand.resource.genesis()
Instantiates the MagickWand API.

Warning: Don’t call this function directly. Use increment_refcount() and decrement_refcount() functions instead.

wand.resource.terminus()
Cleans up the MagickWand API.

Warning: Don’t call this function directly. Use increment_refcount() and decrement_refcount() functions instead.

wand.resource.increment_refcount()
Increments the reference_count and instantiates the MagickWand API if it is the first use.

wand.resource.decrement_refcount()
Decrements the reference_count and cleans up the MagickWand API if it will be no more used.

New in version 0.5.1.

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:

```python
with resource.allocate():
    resource.resource = library.NewResource()
```

c_clear_exception = NotImplemented
clear_exception (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
destroy_resource (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
get_exception (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
is_resource (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.

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.

```
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.
```

```
wand.resource.safe_copy (ptr)
Safely cast memory address to char pointer, convert to python string, and immediately free resources.
```

```
Parameters ptr (ctypes.c_void_p) – The memory address to convert to text string.

Returns tuple (ctypes.c_void_p, str)
New in version 0.5.3.
```

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.

```
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.

    (list) The list of (base_class, suffix) pairs (for each code). It would be zipped with DOMAIN_MAP pairs’ last element.

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.

e exception wand.exceptions.CorruptImageWarning
    Bases: wand.exceptions.BaseWarning, ValueError

    The image file may be corrupt.

wand.exceptions.DOMAIN_MAP = [('ResourceLimit', 'A program resource is exhausted e.g. not enough memory.', (class 'MemoryError'), ...]

 exception wand.exceptions.DelegateError
    Bases: wand.exceptions.BaseError

    An ImageMagick delegate failed to complete.

e exception wand.exceptions.DelegateFatalError
    Bases: wand.exceptions.BaseFatalError

    An ImageMagick delegate failed to complete.

e exception wand.exceptions.DelegateWarning
    Bases: wand.exceptions.BaseWarning

    An ImageMagick delegate failed to complete.

e exception wand.exceptions.DrawError
    Bases: wand.exceptions.BaseError

    A drawing operation failed.

e exception wand.exceptions.DrawFatalError
    Bases: wand.exceptions.BaseFatalError

    A drawing operation failed.

e exception wand.exceptions.DrawWarning
    Bases: wand.exceptions.BaseWarning

    A drawing operation failed.

e exception wand.exceptions.FileOpenError
    Bases: wand.exceptions.BaseError, OSError

    The image file could not be opened for reading or writing.

e exception wand.exceptions.FileOpenFatalError
    Bases: wand.exceptions.BaseFatalError, OSError

    The image file could not be opened for reading or writing.

e exception wand.exceptions.FileOpenWarning
    Bases: wand.exceptions.BaseWarning, OSError

    The image file could not be opened for reading or writing.

e 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.

wand.exceptions.TYPE_MAP = {300: <class 'wand.exceptions.ResourceLimitWarning'>, 305: <class 'wand.exceptions.TypeWarning'>}

(dict) The dictionary of (code, exc_type).

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.

class wand.api.AffineMatrix

class wand.api.MagickPixelPacket

wand.api.library
    (ctypes.CDLL) The MagickWand library.

wand.api.libc
    (ctypes.CDLL) The C standard library.

wand.api.libmagick
    (ctypes.CDLL) The ImageMagick library. It is the same with library on platforms other than Windows.
    New in version 0.1.10.

wand.api.load_library()
    Loads the MagickWand library.
    Returns the MagickWand library and the ImageMagick library
    Return type ctypes.CDLL

class wand.api.PixelInfo

class wand.api.PointInfo

4.1.9 wand.compat — Compatibility layer

This module provides several subtle things to support multiple Python versions (2.6, 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/latest/lib/python3.7/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

```python
wand.compat.binary_type
alias of builtins.bytes
```

```python
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
```

```python
wand.compat.file_types
alias of io.RawIOBase
```

```python
wand.compat.string_type
alias of builtins.str
```

```python
wand.compat.text_type
alias of builtins.str
```

```python
wand.compat.xrange
alias of builtins.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.

```python
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
0.5.8
$ python -m wand.version --verbose
Wand 0.5.8
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
```

(continues on next page)
New in version 0.2.0: The command line interface.

New in version 0.2.2: The \texttt{--verbose/-v} option which also prints ImageMagick library version for CLI.

New in version 0.4.1: The \texttt{--fonts}, \texttt{--formats}, \& \texttt{--config} option allows printing additional information about ImageMagick library.

\begin{verbatim}
$ 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
\end{verbatim}

wand.version.\texttt{VERSION} = '0.5.8'
\begin{enumerate}
\item \texttt{(basestring)} The version string e.g. '0.1.2'.
\item \texttt{Changed in version 0.1.9: Becomes string. (It was \texttt{tuple} before.)}
\end{enumerate}

wand.version.\texttt{VERSION_INFO} = (0, 5, 8)
\begin{enumerate}
\item \texttt{(tuple)} The version tuple e.g. (0, 1, 2).
\item \texttt{Changed in version 0.1.9: Becomes \texttt{tuple}. (It was string before.)}
\end{enumerate}

wand.version.\texttt{MAGICK_VERSION} = None
\begin{enumerate}
\item \texttt{(basestring)} The version string of the linked ImageMagick library. The exactly same string to the result of \texttt{GetMagickVersion()} function.
\item \texttt{Example:}
\begin{verbatim}
'ImageMagick 6.7.7-6 2012-06-03 Q16 http://www.imagemagick.org'
\end{verbatim}
\item \texttt{New in version 0.2.1.}
\end{enumerate}

wand.version.\texttt{MAGICK_VERSION_FEATURES} = 'Cipher DPC Modules OpenMP'
\begin{enumerate}
\item \texttt{(basestring)} A string of all features enabled. This value is identical to what is returned by \texttt{GetMagickFeatures()}
\item \texttt{New in version 0.5.0.}
\end{enumerate}

wand.version.\texttt{MAGICK_VERSION_INFO} = None
\begin{enumerate}
\item \texttt{(tuple)} The version tuple e.g. (6, 7, 7, 6) of \texttt{MAGICK_VERSION}.
\item \texttt{New in version 0.2.1.}
\end{enumerate}

wand.version.\texttt{MAGICK_VERSION_NUMBER} = 1792
\begin{enumerate}
\item \texttt{(numbers.Integral)} The version number of the linked ImageMagick library.
\item \texttt{New in version 0.2.1.}
\end{enumerate}

wand.version.\texttt{MAGICK_RELEASE_DATE} = None
\begin{enumerate}
\item \texttt{(datetime.date)} The release date of the linked ImageMagick library. Equivalent to the result of \texttt{GetMagickReleaseDate()} function.
\item \texttt{New in version 0.2.1.}
\end{enumerate}

wand.version.\texttt{MAGICK_RELEASE_DATE_STRING} = None
\begin{enumerate}
\item \texttt{(basestring)} The date string e.g. '2012-06-03' of \texttt{MAGICK_RELEASE_DATE_STRING}. This value is the exactly same string to the result of \texttt{GetMagickReleaseDate()} function.
\item \texttt{New in version 0.2.1.}
\end{enumerate}
New in version 0.2.1.

\texttt{wand.version.MAGICK_HDRI = None}

(bool) True if ImageMagick is compiled for High Dynamic Range Image.

\texttt{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.

\texttt{wand.version.QUANTUM_RANGE = None}

(numbers.Integral) The quantum range configuration of the linked ImageMagick library.

New in version 0.5.0.

\texttt{wand.version.configure_options}(\textit{pattern}='*')

Queries ImageMagick library for configurations options given at compile-time.

Example: Find where the ImageMagick documents are installed:

```python
>>> from wand.version import configure_options
>>> configure_options('DOC*')
{'DOCUMENTATION_PATH': '/usr/local/share/doc/ImageMagick-6'}
```

**Parameters** \texttt{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

\texttt{wand.version.fonts}(\textit{pattern}='*')

Queries ImageMagick library for available fonts.

Available fonts can be configured by defining \texttt{types.xml}, \texttt{type-ghostscript.xml}, or \texttt{type-windows.xml}. Use \texttt{wand.version.configure_options()} to locate system search path, and \texttt{resources} article for defining xml file.

Example: List all bold Helvetica fonts:

```python
>>> from wand.version import fonts
>>> fonts('Helvetica*Bold*')
['Helvetica-Bold', 'Helvetica-Bold-Oblique', 'Helvetica-BoldOblique', 'Helvetica-Narrow-Bold', 'Helvetica-Narrow-BoldOblique']
```

**Parameters** \texttt{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

\texttt{wand.version.formats}(\textit{pattern}='*')

Queries ImageMagick library for supported formats.

Example: List supported PNG formats:

```python
>>> 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`
5.1 Mailing list

Wand has the list for users. If you want to subscribe the list, just send a mail to:

wand@librelist.com

The list archive provided by Librelist is synchronized every hour.

5.2 Stack Overflow

There’s a Stack Overflow tag for Wand:

http://stackoverflow.com/questions/tagged/wand

Freely ask questions about Wand including troubleshooting. Thanks for sindikat’s contribution.

5.3 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.
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:

```bash
$ git clone git://github.com/emcconville/wand.git
```

If you find a bug, please notify to our issue tracker. Pull requests are always welcome!

We discuss about Wand’s development on IRC. Come #wand channel on freenode network.

Check out Wand Changelog also.
CHAPTER 7

Indices and tables

• genindex
• modindex
• search
Python Module Index

W

wand, 109
wand.api, 211
wand.color, 180
wand.compat, 211
wand.display, 212
wand.drawing, 184
wand.exceptions, 205
wand.font, 183
wand.image, 109
wand.resource, 202
wand.sequence, 201
wand.version, 212
A

abc (in module wand.compat), 211
adaptive_blur () (wand.image.BaseImage method), 122
adaptive_resize () (wand.image.BaseImage method), 122
adaptive_sharpen () (wand.image.BaseImage method), 122
adaptive_threshold () (wand.image.BaseImage method), 123
affine () (wand.drawing.Drawing method), 187
AffineMatrix (class in wand.api), 211
allocate () (wand.resource.Resource method), 202
alpha (wand.color.Color attribute), 181
alpha () (wand.drawing.Drawing method), 187
alpha_channel (wand.image.BaseImage attribute), 123
ALPHA_CHANNEL_TYPES (in module wand.image), 109
alpha_int8 (wand.color.Color attribute), 181
alpha_quantum (wand.color.Color attribute), 181
animation (wand.image.BaseImage attribute), 123
animation (wand.image.Image attribute), 174
annotate () (wand.image.BaseImage attribute), 123
antialias (wand.font.Font attribute), 184
antialias (wand.image.BaseImage attribute), 124
append () (wand.sequence.Sequence method), 201
arc () (wand.drawing.Drawing method), 188
area (wand.image.ConnectedComponentObject attribute), 179
artifacts (wand.image.Image attribute), 174
ArtifactTree (class in wand.image), 179
ascender (wand.drawing.FontMetrics attribute), 200
auto_gamma () (wand.image.BaseImage method), 124
auto_level () (wand.image.BaseImage method), 124
auto_orient () (wand.image.BaseImage method), 124
auto_threshold () (wand.image.BaseImage method), 124
AUTO_THRESHOLD_METHODS (in module wand.image), 110

B

background_color (wand.image.BaseImage attribute), 125
BaseError, 205
BaseFatalError, 205
BaseImage (class in wand.image), 122
BaseWarning, 205
bezier () (wand.drawing.Drawing method), 188
binary () (in module wand.compat), 211
binary_type (in module wand.compat), 212
black (wand.color.Color attribute), 181
black_int8 (wand.color.Color attribute), 181
black_quantum (wand.color.Color attribute), 181
black_threshold () (wand.image.BaseImage method), 125
blank () (wand.image.Image method), 174
BlobError, 205
BlobFatalError, 205
BlobWarning, 206
blue (wand.color.Color attribute), 181
blue_int8 (wand.color.Color attribute), 181
blue_primary (wand.image.BaseImage attribute), 125
blue_quantum (wand.color.Color attribute), 181
blue_shift () (wand.image.BaseImage method), 125
blur () (wand.image.BaseImage method), 125
border () (wand.image.BaseImage method), 125
border_color (wand.drawing.Drawing attribute), 188
border_color (wand.image.BaseImage attribute), 126
brightness_contrast () (wand.image.BaseImage method), 126
c_destroy_resource (wand.resource.Resource attribute), 203
c_equals() (wand.color.Color static method), 181
c_get_exception (wand.resource.Resource attribute), 203
c_is_resource (wand.resource.Resource attribute), 203
CacheError, 206
CacheFatalError, 206
CacheWarning, 206
canny() (wand.image.BaseImage method), 126
caption() (wand.image.BaseImage method), 126
cdl() (wand.image.BaseImage method), 127
center_x (wand.image.ConnectedComponentObject attribute), 179
center_y (wand.image.ConnectedComponentObject attribute), 179
centroid (wand.image.ConnectedComponentObject attribute), 179
channel_depths (wand.image.Image attribute), 174
channel_images (wand.image.Image attribute), 174
ChannelDepthDict (class in wand.image), 172
ChannelImageDict (class in wand.image), 172
CHANNELLS (in module wand.image), 110
character_height (wand.drawing.FontMetrics attribute), 200
character_width (wand.drawing.FontMetrics attribute), 200
charcoal() (wand.image.BaseImage method), 127
chop() (wand.image.BaseImage method), 127
circle() (wand.drawing.Drawing method), 188
clahe() (wand.image.BaseImage method), 127
clamp() (wand.image.BaseImage method), 127
clear() (wand.image.Image method), 174
clip_path (wand.drawing.Drawing attribute), 189
CLIP_PATH_UNITS (in module wand.drawing), 185
clip_rule (wand.drawing.Drawing attribute), 189
clip_units (wand.drawing.Drawing attribute), 189
circle() (wand.drawing.Drawing method), 189
circle() (wand.image.BaseImage method), 128
circle() (wand.image.Iterator method), 178
circle_from_cc_object_info() (wand.image.ConnectedComponentObject method), 179
close() (wand.image.Image method), 174
ClosedImageError, 172
clut() (wand.image.BaseImage method), 128
coalesce() (wand.image.BaseImage method), 128
CODE_MAP (in module wand.exceptions), 206
CoderError, 206
CoderFatalError, 206
CoderWarning, 206
Color (class in wand.color), 180
color (wand.font.Font attribute), 184
color() (wand.drawing.Drawing method), 189
color_decision_list() (wand.image.BaseImage method), 128
color_map() (wand.image.BaseImage method), 129
color_matrix() (wand.image.BaseImage method), 129
colorize() (wand.image.BaseImage method), 130
colors (wand.image.BaseImage attribute), 130
colorspace (wand.image.BaseImage attribute), 130
COLORSPACE_TYPES (in module wand.image), 110
comment() (wand.drawing.Drawing method), 189
compare() (wand.image.BaseImage method), 130
compare_layers() (wand.image.Image method), 175
COMPARE_METRICS (in module wand.image), 112
complex() (wand.image.BaseImage method), 131
compose() (wand.image.BaseImage attribute), 131
composite() (wand.drawing.Drawing method), 189
composite() (wand.image.BaseImage method), 132
composite_channel() (wand.image.BaseImage method), 132
COMPOSITE_OPERATORS (in module wand.image), 112
compression() (wand.image.BaseImage attribute), 133
compression_quality() (wand.image.BaseImage attribute), 133
COMPRESSION_TYPES (in module wand.image), 112
concat() (wand.image.BaseImage method), 133
configure_options() (in module wand.version), 214
ConfigureError, 206
ConfigureFatalError, 206
ConfigureWarning, 206
connected_components() (wand.image.BaseImage method), 133
ConnectedComponentObject (class in wand.image), 179
container() (wand.sequence.SingleImage attribute), 201
contrast() (wand.image.BaseImage method), 134
contrast_stretch() (wand.image.BaseImage method), 134
convert() (wand.image.Image method), 175
CorruptImageError, 206
CorruptImageFatalError, 206
CorruptImageWarning, 207
crop() (wand.image.BaseImage method), 134
current_index (wand.sequence.Sequence attribute), 201
cyan() (wand.color.Color attribute), 182
cyan_int8() (wand.color.Color attribute), 182
cyan_quantum() (wand.color.Color attribute), 182
cycle_color_map() (wand.image.BaseImage method), 135
deconstruct() (wand.image.BaseImage method), 135
decrement_refcount() (in module wand.resource), 202
delay(wand.sequence.SingleImage attribute), 201
depend Error, 207
deregate FatalError, 207
deregate Warning, 207
depth(wand.image.BaseImage attribute), 136
descender(wand.drawing.FontMetrics attribute), 200
despeckle() (wand.image.BaseImage method), 136
destroy() (wand.image.BaseImage method), 175
destroy() (wand.resource.Resource method), 203
destroyed Resource Error, 205
dirty (wand.color.Color attribute), 182
dirty (wand.image.BaseImage attribute), 136
display() (in module wand.display), 212
dispose(wand.image.BaseImage attribute), 136
DISPOSE_TYPES (in module wand.image), 113
distort() (wand.image.BaseImage method), 136
DISTORTION_METHODS (in module wand.image), 113
DITHER_METHODS (in module wand.image), 113
DOMAIN_MAP (in module wand.exceptions), 207
draw() (wand.drawing.Drawing method), 190
DrawError, 207
DrawFatalError, 207
drawing (class in wand.drawing), 187
drawing Warning, 207

e() (wand.image.BaseImage method), 137
e() (wand.drawing.Drawing method), 190
emboos() (wand.image.BaseImage method), 137
encode_filename() (in module wand.compat), 212
enhance() (wand.image.BaseImage method), 137
environment variable
MAGICK_HOME, 11–13
WAND_MAGICK_LIBRARY_SUFFIX, 13, 89
equalize() (wand.image.BaseImage method), 137
evaluate() (wand.image.BaseImage method), 138
EVALUATE_OPS (in module wand.image), 113
export_pixels() (wand.image.BaseImage method), 138
extend() (wand.sequence.Sequence method), 201
extent() (wand.image.BaseImage method), 139

features() (wand.image.BaseImage method), 139
fft() (wand.image.BaseImage method), 140
file_types (in module wand.compat), 212
FileOpenError, 207
FileOpenFatalError, 207

fill_color (wand.drawing.Drawing attribute), 190
fill_opacity (wand.drawing.Drawing attribute), 190
fill_rule (wand.drawing.Drawing attribute), 190
FILL_RULE_TYPES (in module wand.drawing), 185
FILTER_TYPES (in module wand.image), 114
flip() (wand.image.BaseImage method), 140
g() (wand.image.BaseImage method), 140
Font (class in wand.font), 184
font (wand.drawing.Drawing attribute), 190
font (wand.image.BaseImage attribute), 140
font_antialias (wand.image.BaseImage attribute), 140
font_family (wand.drawing.Drawing attribute), 190
FONT_METRICS_ATTRIBUTES (in module wand.drawing), 185
font_path (wand.image.BaseImage attribute), 140
font_resolution (wand.drawing.Drawing attribute), 190
font_size (wand.drawing.Drawing attribute), 190
font_size (wand.image.BaseImage attribute), 140
font_stretch (wand.drawing.Drawing attribute), 191
font_style (wand.drawing.Drawing attribute), 191
font_weight (wand.drawing.Drawing attribute), 191
FontMetrics (class in wand.drawing), 200
fonts() (in module wand.version), 214
format (wand.image.BaseImage attribute), 140
formats() (in module wand.version), 214
forward_fourier_transform() (wand.image.BaseImage method), 141
frame() (wand.image.BaseImage method), 141
from_array() (wand.image.Image class method), 175
from_hsl() (wand.color.Color class method), 182
function() (wand.image.BaseImage method), 141
FUNCTION_TYPES (in module wand.image), 115
fuzz (wand.image.BaseImage attribute), 142
fx() (wand.image.BaseImage method), 142

g() (wand.image.BaseImage method), 142
gaussian_blur() (wand.image.BaseImage method), 143
genesis() (in module wand.resource), 202
get_exception() (wand.resource.Resource method), 203
get_font_metrics() (wand.drawing.Drawing method), 191
get_resource_limit() (wand.resource.ResourceLimits method), 204
g() (wand.drawing.Drawing attribute), 191
g() (wand.image.BaseImage attribute), 143

Index
mean_channel() (wand.image.BaseImage method), 150
mean_color (wand.image.ConnectedComponentObject attribute), 180
mean_shift() (wand.image.BaseImage method), 150
merge_layers() (wand.image.BaseImage method), 150
Metadata (class in wand.image), 178
metadata (wand.image.Image attribute), 176
mimetype (wand.image.Image attribute), 176
minima (wand.image.BaseImage attribute), 150
MissingDelegateError, 208
MissingDelegateFatalError, 208
MissingDelegateWarning, 208
mode() (wand.image.BaseImage method), 151
modulate() (wand.image.BaseImage method), 151
ModuleError, 208
ModuleFatalError, 208
ModuleWarning, 208
MonitorError, 208
MonitorFatalError, 208
MonitorWarning, 208
morphology() (wand.image.BaseImage method), 151
MORPHOLOGY_METHODS (in module wand.image), 151
motion_blur() (wand.image.BaseImage method), 152

N
negate() (wand.image.BaseImage method), 152
next() (wand.image.Iterator method), 178
noise() (wand.image.BaseImage method), 152
NOISE_TYPES (in module wand.image), 119
normalize() (wand.image.BaseImage method), 152
normalized_string (wand.color.Color attribute), 183

O
offset (wand.image.ConnectedComponentObject attribute), 180
oil_paint() (wand.image.BaseImage method), 153
opacity (wand.drawing.Drawing attribute), 192
opaque_paint() (wand.image.BaseImage method), 153
optimize_layers() (wand.image.BaseImage method), 153
optimize_transparency() (wand.image.BaseImage method), 153
OptionDict (class in wand.image), 178
OptionError, 208
OptionFatalError, 208
options (wand.image.BaseImage attribute), 153
OptionWarning, 208
ordered_dither() (wand.image.BaseImage method), 153
orientation (wand.image.BaseImage attribute), 154
ORIENTATION_TYPES (in module wand.image), 119

P
page (wand.image.BaseImage attribute), 154
page_height (wand.image.BaseImage attribute), 154
page_width (wand.image.BaseImage attribute), 154
page_x (wand.image.BaseImage attribute), 154
page_y (wand.image.BaseImage attribute), 154
PAINT_METHOD_TYPES (in module wand.drawing), 186
parse_meta_geometry() (wand.image.BaseImage method), 155
path (wand.font.Font attribute), 184
path_close() (wand.drawing.Drawing method), 192
path_curve() (wand.drawing.Drawing method), 192
path_curve_to_quadratic_bezier() (wand.drawing.Drawing method), 192
path_elliptic_arc() (wand.drawing.Drawing method), 192
path_finish() (wand.drawing.Drawing method), 193
path_horizontal_line() (wand.drawing.Drawing method), 193
path_line() (wand.drawing.Drawing method), 193
path_move() (wand.drawing.Drawing method), 193
path_start() (wand.drawing.Drawing method), 194
path_vertical_line() (wand.drawing.Drawing method), 194
percent_escape() (wand.image.BaseImage method), 155
ping() (wand.image.Image class method), 176
PIXEL_INTERPOLATE_METHODS (in module wand.image), 120
PixelInfo (class in wand.api), 211
point() (wand.drawing.Drawing method), 194
PointInfo (class in wand.api), 211
polaroid() (wand.image.BaseImage method), 155
PolicyError, 209
PolicyFatalError, 209
PolicyWarning, 209
polygon() (wand.drawing.Drawing method), 194
polyline() (wand.drawing.Drawing method), 194
polynomial() (wand.image.BaseImage method), 155
pop() (wand.drawing.Drawing method), 194
pop_clip_path() (wand.drawing.Drawing method), 195
pop_defs() (wand.drawing.Drawing method), 195
pop_pattern() (wand.drawing.Drawing method), 195
posterize() (wand.image.BaseImage method), 156
ProfileDict (class in wand.image), 179
profiles (wand.image.Image attribute), 177
pseudo() (wand.image.Image method), 177
push() (wand.drawing.Drawing method), 195
push_clip_path() (wand.drawing.Drawing method), 196
push_def() (wand.drawing.Drawing method), 196
push_pattern() (wand.drawing.Drawing method), 196
PY3 (in module wand.compat), 211
QUANTUM_DEPTH (in module wand.version), 214
QUANTUM_RANGE (in module wand.version), 214
quantum_range (wand.image.BaseImage attribute), 157
raise_exception() (wand.resource.Resource method), 203
random_threshold() (wand.image.BaseImage method), 157
RandomError, 209
RandomFatalError, 209
RandomWarning, 209
range_channel() (wand.image.BaseImage method), 157
range_threshold() (wand.image.BaseImage method), 157
read() (wand.image.Image method), 177
read_mask() (wand.image.BaseImage method), 158
rectangle() (wand.drawing.Drawing method), 196
red (wand.color.Color attribute), 183
red_int8 (wand.color.Color attribute), 183
red_primary (wand.image.BaseImage attribute), 158
red_quantum (wand.color.Color attribute), 183
RegistryError, 209
RegistryFatalError, 209
RegistryWarning, 209
remap() (wand.image.BaseImage method), 158
rendering_intent (wand.image.BaseImage attribute), 158
RENDERING_INTENT_TYPES (in module wand.image), 120
resample() (wand.image.BaseImage method), 158
reset_coords() (wand.image.BaseImage method), 159
resize() (wand.image.BaseImage method), 159
resolution (wand.image.BaseImage attribute), 159
Resource (class in wand.resource), 202
resource (wand.resource.Resource attribute), 203
resource() (wand.resource.ResourceLimits method), 204
ResourceLimitError, 209
ResourceLimitFatalError, 209
ResourceLimits (class in wand.resource), 203
ResourceLimitWarning, 209
rotate() (wand.drawing.Drawing method), 197
rotate() (wand.image.BaseImage method), 159
rotational_blur() (wand.image.BaseImage method), 159
safe_copy() (in module wand.resource), 205
sample() (wand.image.Image method), 177
save() (wand.image.Image method), 160
scale() (wand.drawing.Drawing method), 197
scale() (wand.image.BaseImage method), 160
class in wand.color, 183
class in wand.resource, 203
scene (wand.image.BaseImage attribute), 160
seed (wand.image.BaseImage attribute), 160
selective_blur() (wand.image.BaseImage method), 160
sequence (class in wand.sequence), 201
sequence (wand.image.BaseImage attribute), 161
set_fill_pattern_url() (wand.drawing.Drawing method), 197
set_resource_limit() (wand.resource.ResourceLimits method), 204
set_stroke_pattern_url() (wand.drawing.Drawing method), 198
shade() (wand.image.BaseImage method), 161
shadow() (wand.image.BaseImage method), 161
sharpen() (wand.image.BaseImage method), 161
shear() (wand.image.BaseImage method), 162
sigmoidal_contrast() (wand.image.BaseImage method), 162
signature (wand.image.BaseImage attribute), 162
similar() (wand.image.BaseImage method), 163
SingleImage (class in wand.sequence), 201
size (wand.font.Font attribute), 184
size (wand.image.BaseImage attribute), 163
size (wand.image_CONNECTEDCOMPONENTOBJECT attribute), 180
sketch() (wand.image.BaseImage method), 163
skew() (wand.drawing.Drawing method), 198
skewness (wand.image.BaseImage attribute), 164
smush() (wand.image.BaseImage method), 164
solarize() (wand.image.BaseImage method), 164
sparse_color() (wand.image.BaseImage method), 164
SPARSE_COLOR_METHODS (in module wand.image), 120
splice() (wand.image.BaseImage method), 165
spread() (wand.image.BaseImage method), 165
Index 231

standard_deviation (wand.image.BaseImage attribute), 166
statistic() (wand.image.BaseImage method), 166
STATISTIC_TYPES (in module wand.image), 120
stegano() (wand.image.BaseImage method), 166
stereogram() (wand.image.Image class method), 177
STORAGE_TYPES (in module wand.image), 121
StreamError, 209
StreamFatalError, 210
StreamWarning, 210
STRETCH_TYPES (in module wand.drawing), 186
string (wand.color.Color attribute), 183
string_type (in module wand.compat), 212
strip() (wand.image.BaseImage method), 166
stroke_antialias (wand.drawing.Drawing attribute), 198
stroke_color (wand.drawing.Drawing attribute), 198
stroke_color (wand.font.Font attribute), 184
stroke_dash_array (wand.drawing.Drawing attribute), 198
stroke_dash_offset (wand.drawing.Drawing attribute), 198
stroke_line_cap (wand.drawing.Drawing attribute), 198
stroke_line_join (wand.drawing.Drawing attribute), 198
stroke_miter_limit (wand.drawing.Drawing attribute), 198
stroke_opacity_limit (wand.drawing.Drawing attribute), 199
stroke_width (wand.drawing.Drawing attribute), 199
stroke_width (wand.font.Font attribute), 184
STYLE_TYPES (in module wand.drawing), 186
swirl() (wand.image.BaseImage method), 166

text_encoding (wand.drawing.Drawing attribute), 199
text_height (wand.drawing.FontMetrics attribute), 200
text_interline_spacing (wand.drawing.Drawing attribute), 199
text_interword_spacing (wand.drawing.Drawing attribute), 199
text_kerning (wand.drawing.Drawing attribute), 199
text_type (in module wand.compat), 212
text_under_color (wand.drawing.Drawing attribute), 199
text_width (wand.drawing.FontMetrics attribute), 200
texture() (wand.image.BaseImage method), 167
threshold() (wand.image.BaseImage method), 167
thumbnail() (wand.image.BaseImage method), 167
ticks_per_second (wand.image.BaseImage attribute), 167
tint() (wand.image.BaseImage method), 167
top (wand.image.ConnectedComponentObject attribute), 180
transform() (wand.image.BaseImage method), 168
transform_colorspace() (wand.image.BaseImage method), 169
translate() (wand.drawing.Drawing method), 199
transparent_color() (wand.image.BaseImage method), 169
transparentize() (wand.image.BaseImage method), 169
transpose() (wand.image.BaseImage method), 169
transverse() (wand.image.BaseImage method), 169
type() (wand.image.BaseImage attribute), 170
TYPE_MAP (in module wand.exceptions), 210
TypeError, 210
TypeError, 210
TypeWarning, 210

UNIT_TYPES (in module wand.image), 122
units (wand.image.BaseImage attribute), 170
unsharp_mask() (wand.image.BaseImage method), 170

vector_graphics (wand.drawing.Drawing attribute), 200
VERSION (in module wand.version), 213
VERSION_INFO (in module wand.version), 213
viewbox() (wand.drawing.Drawing method), 200
vignette() (wand.image.BaseImage method), 170  yellow_quantum (wand.color.Color attribute), 183
virtual_pixel (wand.image.BaseImage attribute), 171
VIRTUAL_PIXEL_METHOD (in module wand.image), 121

W
wand (module), 109
wand (wand.image.BaseImage attribute), 171
wand.api (module), 211
wand.color (module), 180
wand.compat (module), 211
wand.display (module), 212
wand.drawing (module), 184
wand.exceptions (module), 205
wand.font (module), 183
wand.image (module), 109
wand.resource (module), 202
wand.sequence (module), 201
wand.version (module), 212
WAND_MAGICK_LIBRARY_SUFFIX, 13, 89
WandError, 210
WandException, 210
WandFatalError, 210
WandLibraryVersionError, 210
WandRuntimeError, 210
WandWarning, 210
watermark() (wand.image.BaseImage method), 171
wave() (wand.image.BaseImage method), 171
wavelet_denoise() (wand.image.BaseImage method), 171
white_point (wand.image.BaseImage attribute), 171
white_threshold() (wand.image.BaseImage method), 171
width (wand.image.BaseImage attribute), 172
width (wand.image.ConnectedComponentObject attribute), 180
write_mask() (wand.image.BaseImage method), 172

X
x (wand.drawing.FontMetrics attribute), 200
x1 (wand.drawing.FontMetrics attribute), 200
x2 (wand.drawing.FontMetrics attribute), 200
xrange (in module wand.compat), 212
XServerError, 210
XServerFatalError, 211
XServerWarning, 211

Y
y (wand.drawing.FontMetrics attribute), 200
y1 (wand.drawing.FontMetrics attribute), 200
y2 (wand.drawing.FontMetrics attribute), 201
yellow (wand.color.Color attribute), 183
yellow_int8 (wand.color.Color attribute), 183