Wand Documentation

Release 0.5.4

Hong Minhee

Jul 09, 2019

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 . . . 13
- 3.4 Writing images . . . 16
- 3.5 Resizing and cropping . . . 18
- 3.6 Transformation . . . 22
- 3.7 Colorspace . . . 34
- 3.8 Color Enhancement . . . 35
- 3.9 Distortion . . . 43
- 3.10 Drawing . . . 50
- 3.11 Reading EXIF . . . 61
- 3.12 Layers . . . 62
- 3.13 Sequence . . . 68
- 3.14 Resource management . . . 70
- 3.15 Running tests . . . 70
- 3.16 Roadmap . . . 72
- 3.17 Wand Changelog . . . 72
- 3.18 Talks and Presentations . . . 87

4 References 89
- 4.1 `wand` — Simple MagickWand API binding for Python . . . 89

5 Troubleshooting 99
- 5.1 Mailing list . . . 99
- 5.2 Stack Overflow . . . 99
- 5.3 Documentation . . . 99

6 Open source 101

7 Indices and tables 103

Python Module Index 105

Index 107

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

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

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

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

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

CHAPTER 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

CHAPTER 3

User's guide

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.

```
import numpy as np
from 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.

```
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.

```
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.

```
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` paramater. 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 t0 7

A few key changes worth reviewing.

HDRI by Default

Vanilla installs of ImageMagick-7 include HDRI enabled by default. Users may experiences 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 experiences color-value underflow / overflow during arithmetic operations when upgrading.

An example of an underflow between versions:

```
# 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.:

```
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:

```
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 /

enumerated value.

For example:

```
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*
- *Fefora*
- *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:

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

3.2.3 Install ImageMagick on Mac

You need one of Homebrew or MacPorts to install ImageMagick.

Homebrew

```
$ brew install imagemagick
```

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

```
$ brew install imagemagick --with-liblqr
```

MacPorts

```
$ sudo port install imagemagick
```

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

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

3.2.4 Install ImageMagick on Windows

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

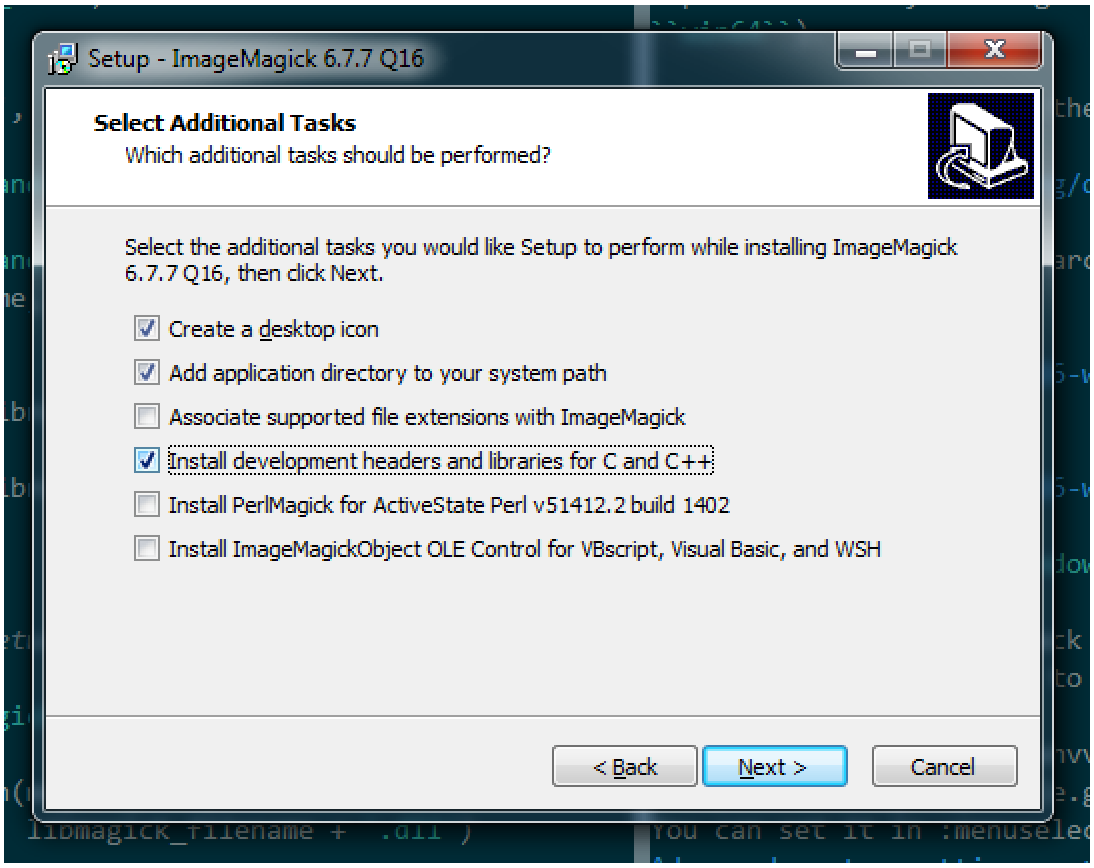
You can download it from the following link:

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.

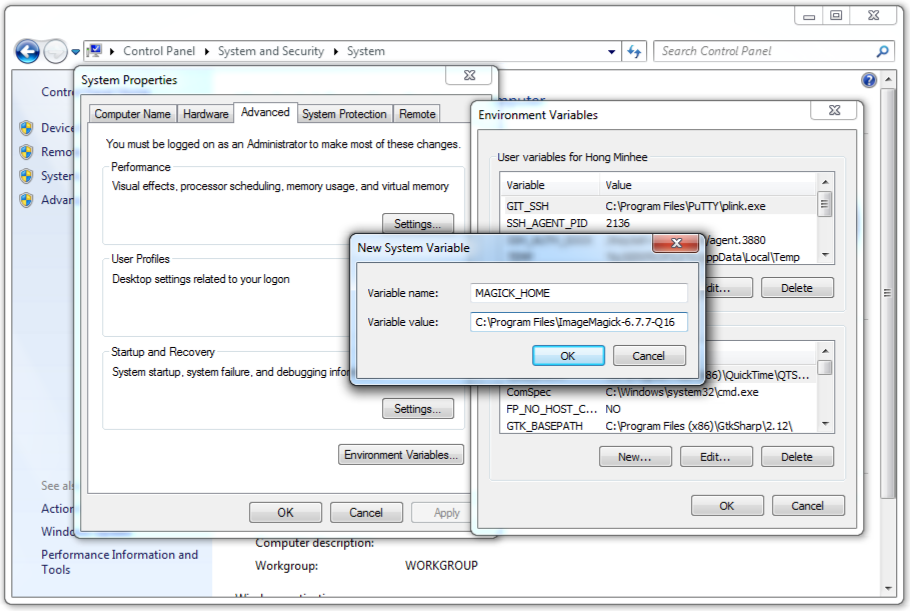

Setup - ImageMagick 6.7.7 Q16

Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing ImageMagick 6.7.7 Q16, then click Next.

- [x] Create a desktop icon
- [x] Add application directory to your system path
- [ ] Associate supported file extensions with ImageMagick
- [x] Install development headers and libraries for C and C++
- [ ] Install PerlMagick for ActiveState Perl v51412.2 build 1402
- [ ] Install ImageMagickObject OLE Control for VBscript, Visual Basic, and WSH

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



You must be logged on as an Administrator to make most of these changes.

Performance: Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles: Desktop settings related to your logon

Startup and Recovery: System startup, system failure, and debugging infor

Environment Variables

User variables for Hong Minhee

Variable	Value
GIT_SSH	C:\Program Files\PuTTY\plink.exe
SSH_AGENT_PID	2136

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
GTK_BASEPATH	C:\Program Files (x86)\GtkSharp\2.12\

New System Variable

Variable name: MAGICK_HOME

Variable value: C:\Program Files\ImageMagick-6.7.7-Q16

Computer description:

Workgroup: WORKGROUP

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 Install Wand on Debian/Ubuntu

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

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

3.2.7 Install Wand on Fedora

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

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

3.2.8 Install Wand on FreeBSD

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

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

3.3 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`:

```
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:

```
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:

```
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:

```
from wand.image import Image

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

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

```
from wand.image import Image

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

3.3.5 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:

```
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 accessable through coder protocols.

```
from wand.image import Image

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

Commun Pseudo images

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

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

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

3.4 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.

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

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

```
from wand.image import Image

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

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

```
from wand.image import Image

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

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

```
$ identify -list format
```

3.4.2 Save to file

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

```
from wand.image import Image

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

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

```
from wand.image import Image

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

3.4.3 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`:

```
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:

```
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 `width` and `height` properties:

```
>>> from urllib.request import urlopen
>>> from wand.image import Image
>>> f = urlopen('http://pbs.twimg.com/profile_images/712673855341367296/WY6aLbBV_
↪normal.jpg')
>>> with Image(file=f) as img:
...     width = img.width
...     height = img.height
...
>>> f.close()
>>> width
48
>>> height
48
```

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

Note: These three properties are all readonly.

3.5.1 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.

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

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

3.5.2 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:

```
>>> 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.

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

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

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

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

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

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

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

3.5.4 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:

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

Other example calls:

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

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

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

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

See also:

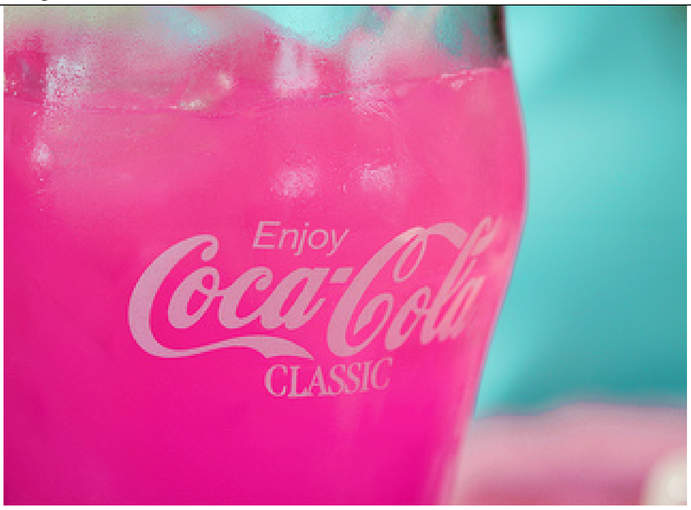
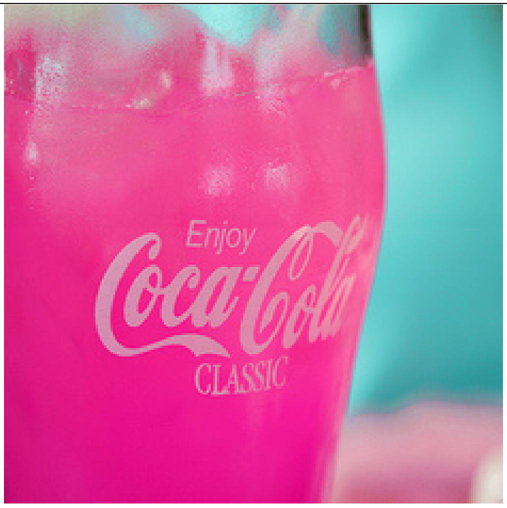
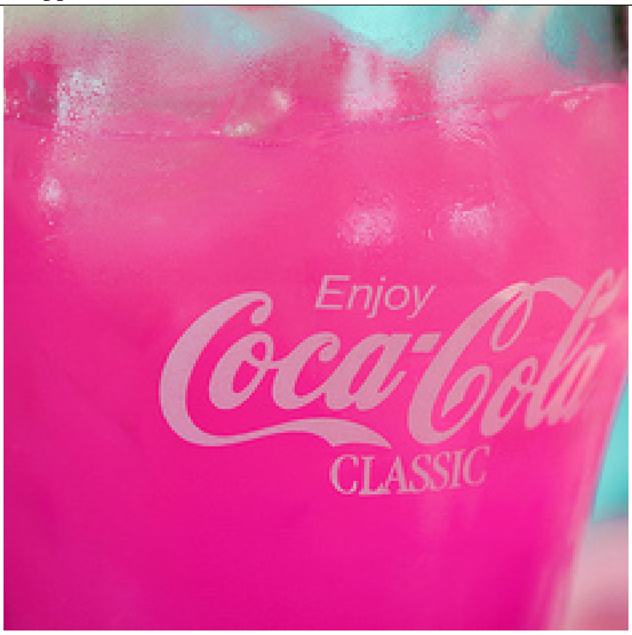
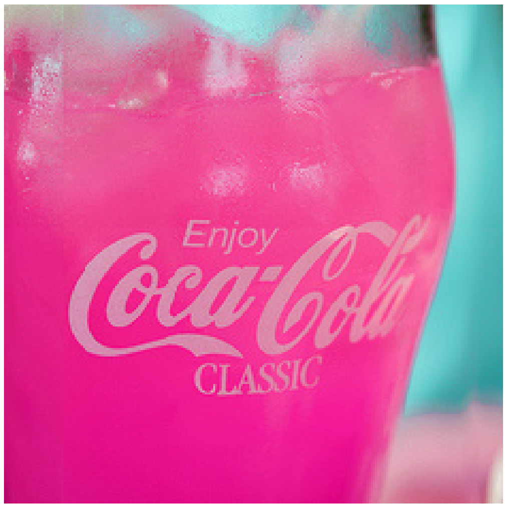
ImageMagick Geometry Specifications Cropping and resizing geometry for the `transform` method are specified according to ImageMagick's geometry string format. The ImageMagick documentation provides more information about geometry strings.

3.5.5 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:

Original	Resized
	
Cropped	**Seam carving**
	

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

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

(continued from previous 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 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.

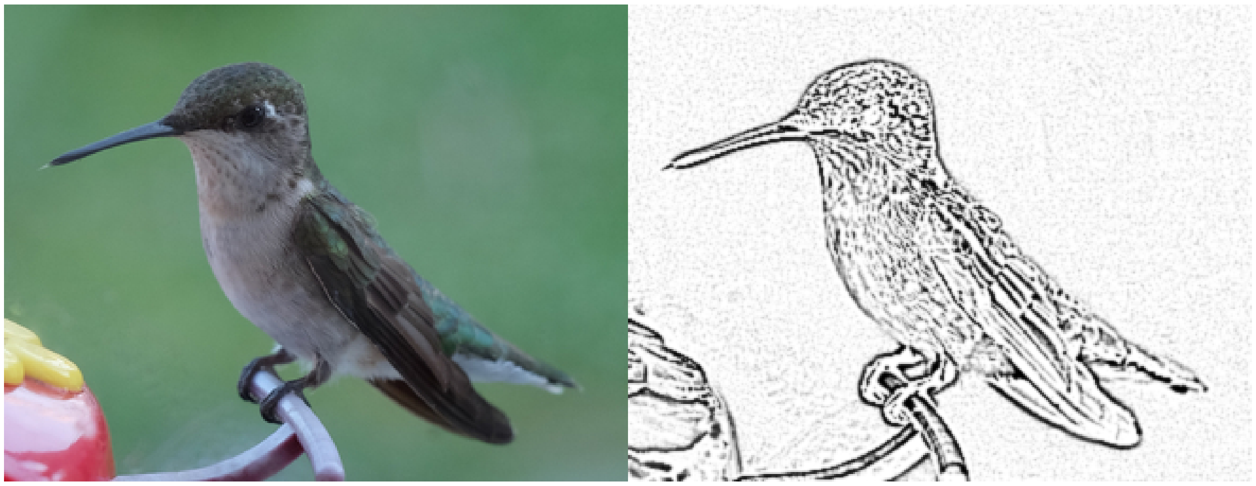
3.6.1 Charcoal

New in version 0.5.3.

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

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.charcoal(radius=1.5, sigma=0.5)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-charcoal.jpg")
```



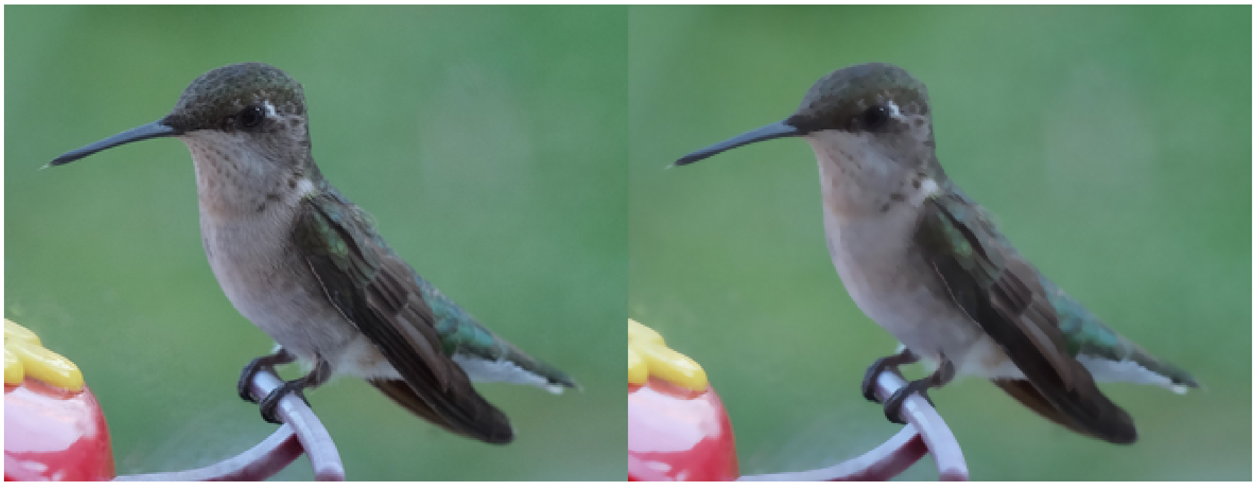
3.6.2 Despeckle

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*.

```
from wand.image import Image

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



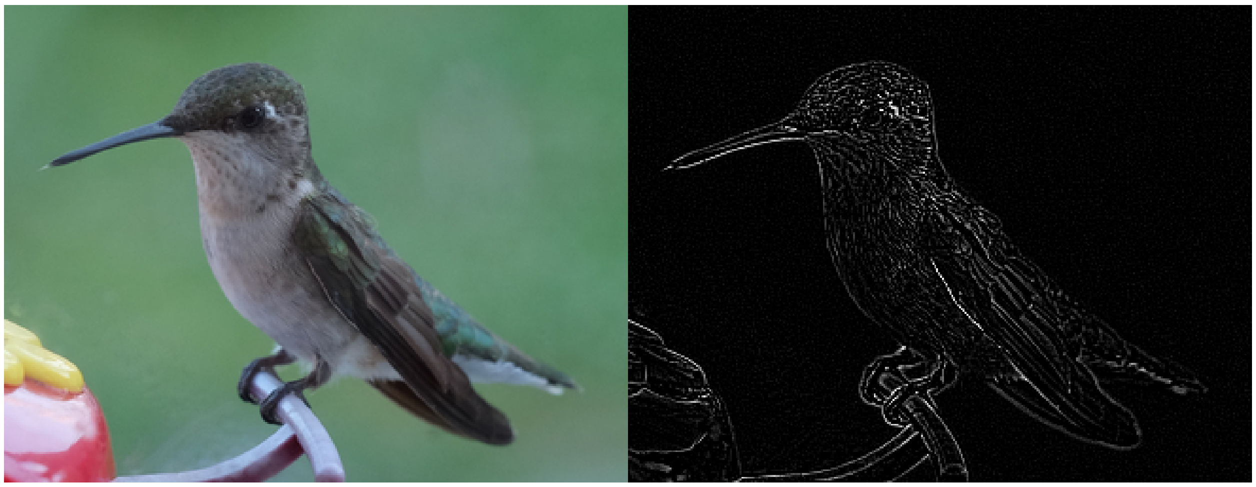
3.6.3 Edge

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.

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.transform_colorspace("gray")
        right.edge(1)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-edge.jpg")
```



3.6.4 Emboss

New in version 0.5.0.

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

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.transform_colorspace("gray")
        right.emboss(radius=3.0, sigma=1.75)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-emboss.jpg")
```

3.6.5 Enhance

New in version 0.5.0.

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

```
from wand.image import Image

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

3.6.6 Flip and flop

New in version 0.3.0.

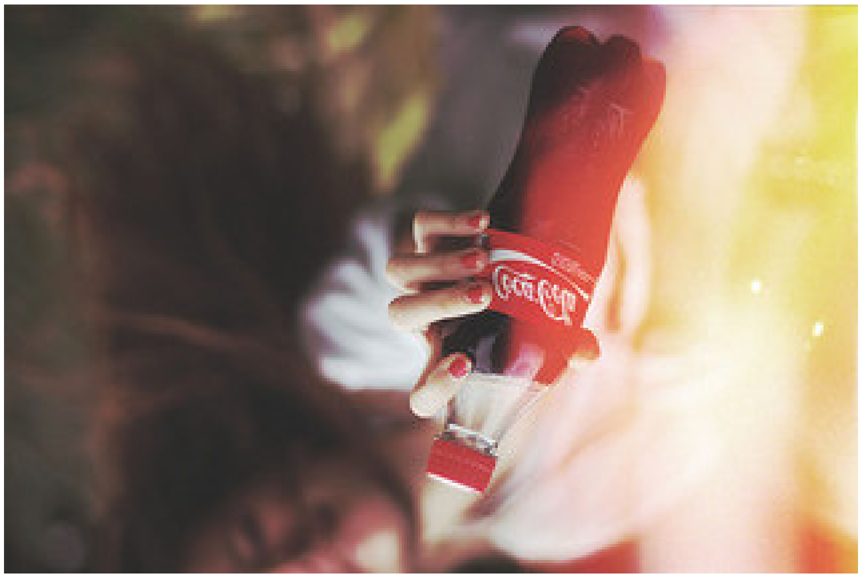
You can make a mirror image by reflecting the pixels around the central x- or y-axis. For example, where the given image `transform.jpg`:

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

```
from wand.image import Image

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

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



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

```
from wand.image import Image

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

(continues on next page)

(continued from previous page)

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

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

3.6.7 Noise

New in version 0.5.3.

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

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

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

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.noise("laplacian", attenuate=1.0)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-noise.jpg")
```

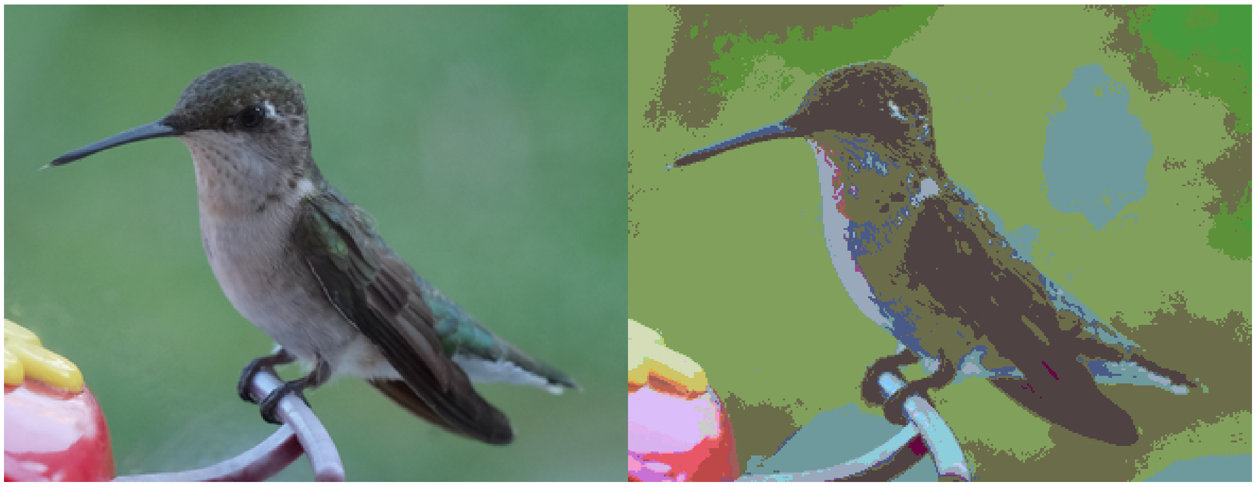
3.6.8 Remap

New in version 0.5.3.

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

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as 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")
```

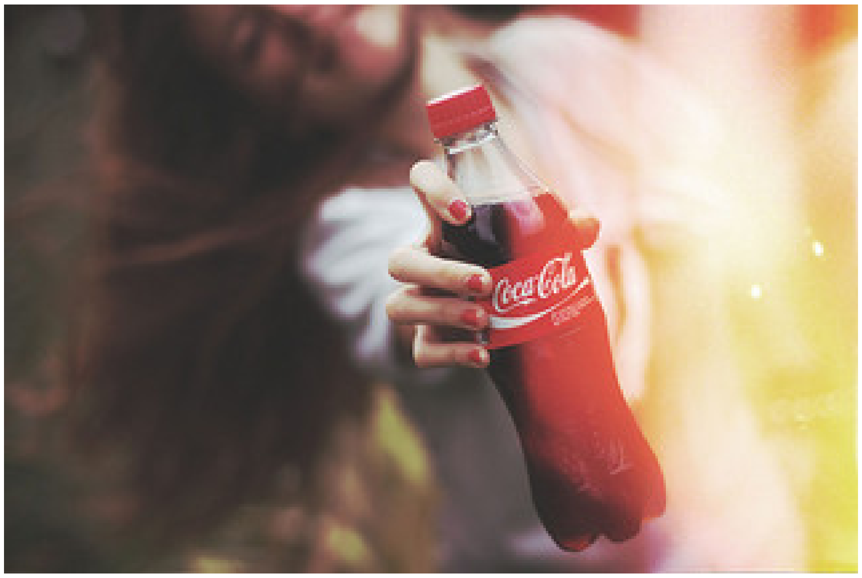


3.6.9 Rotation

New in version 0.1.8.

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

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



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

```
from wand.image import Image

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

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

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

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

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

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

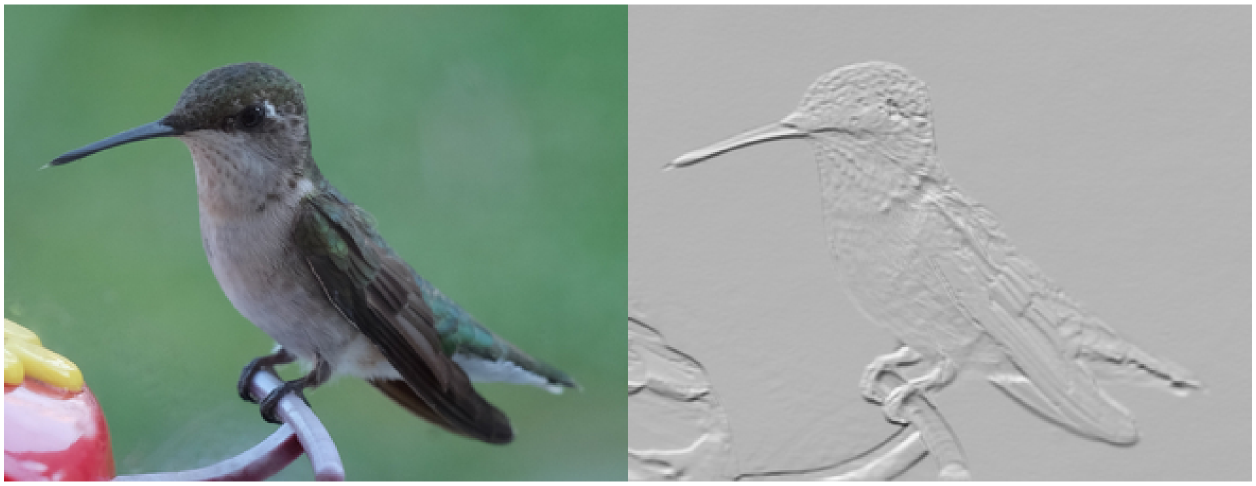
3.6.10 Shade

New in version 0.5.0.

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

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.shade(gray=True,
                    azimuth=286.0,
                    elevation=45.0)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-shade.jpg")
```



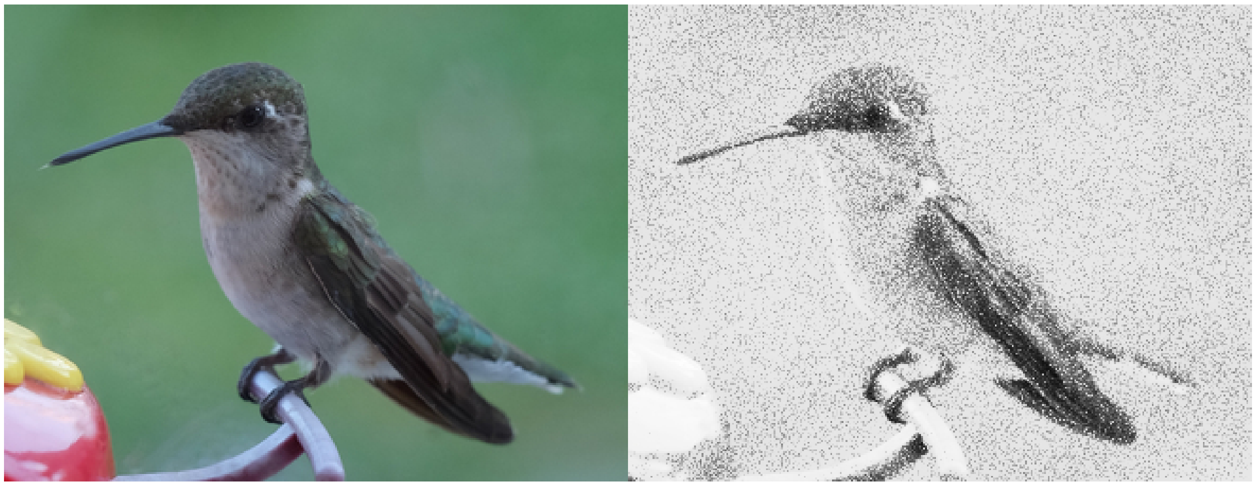
3.6.11 Sketch

New in version 0.5.3.

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

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.transform_colorspace("gray")
        right.sketch(0.5, 0.0, 98.0)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-sketch.jpg")
```



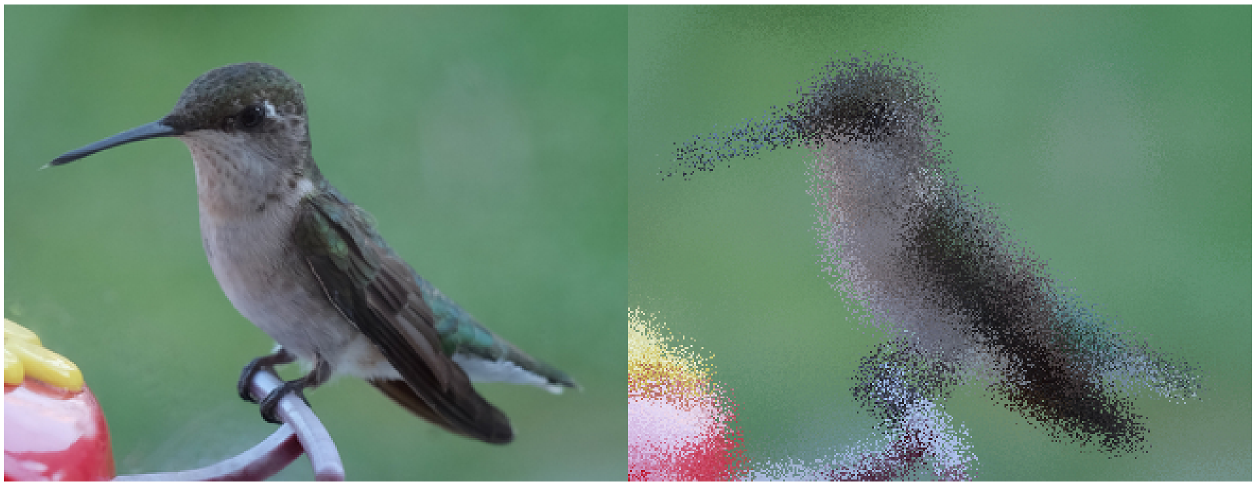
3.6.12 Spread

New in version 0.5.3.

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

```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.spread(8.0)
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-spread.jpg")
```



3.6.13 Statistic

New in version 0.5.3.

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

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

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

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

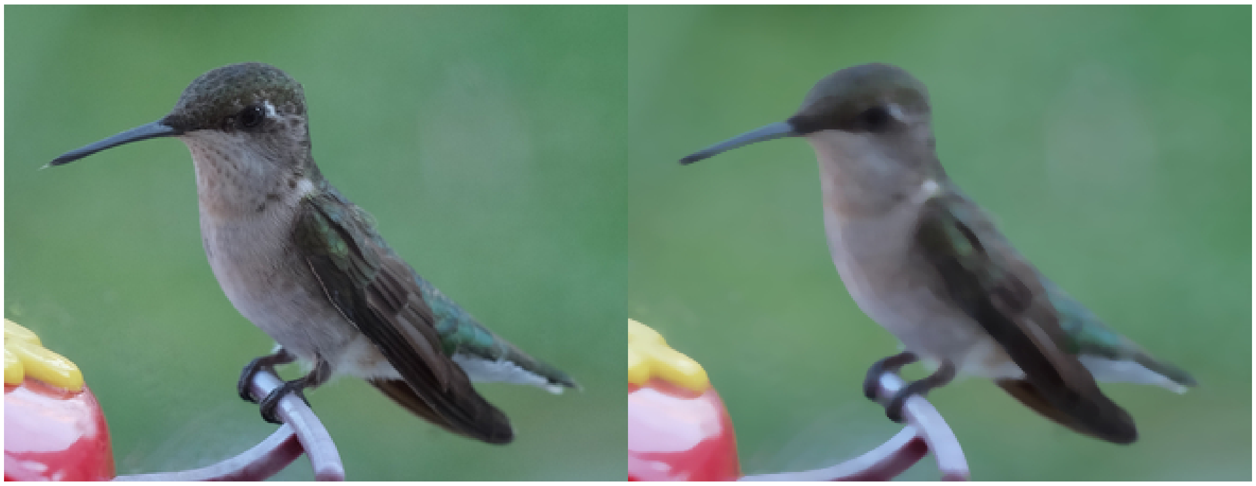
```
from wand.image import Image

with Image(filename="hummingbird.jpg") as left:
    with left.clone() as right:
        right.statistic("median", width=8, height=5)
```

(continues on next page)

(continued from previous page)

```
        left.extent(width=left.width*2)
        left.composite(right, top=0, left=right.width)
    left.save(filename="hummingbird-statistic.jpg")
```



3.7 Colorspace

3.7.1 Image types

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

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

You can change this value:

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

See also:

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

3.7.2 Enable alpha channel

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

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

It's a writable property:

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

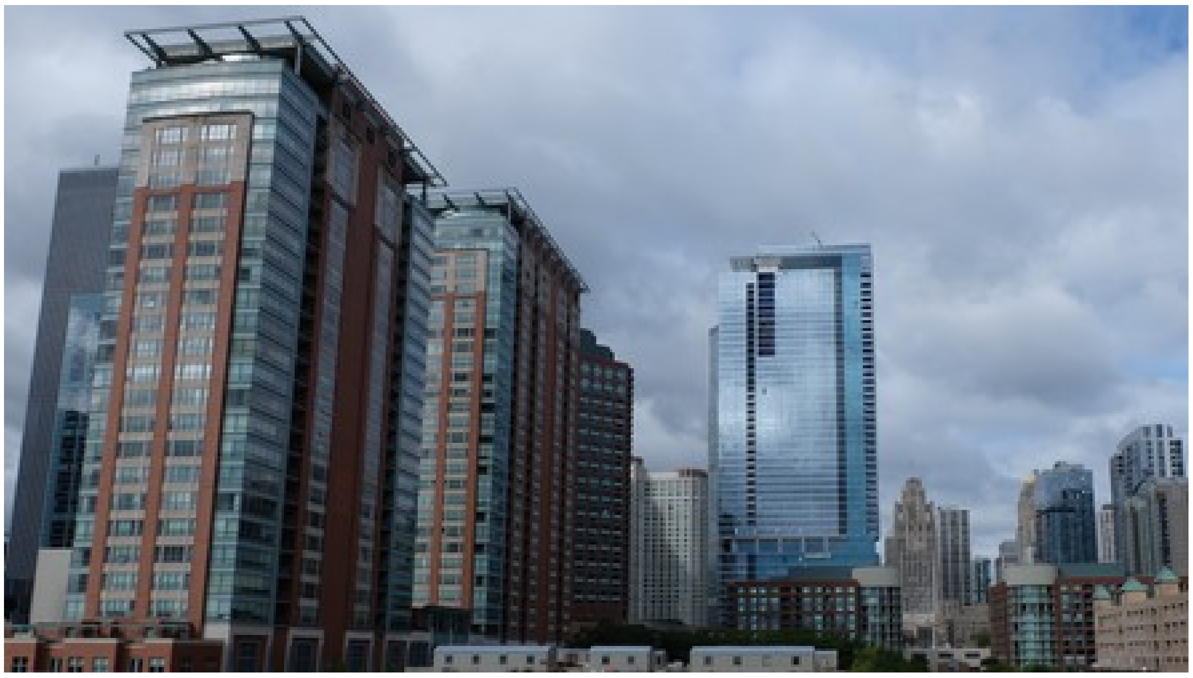
3.8 Color Enhancement

3.8.1 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:

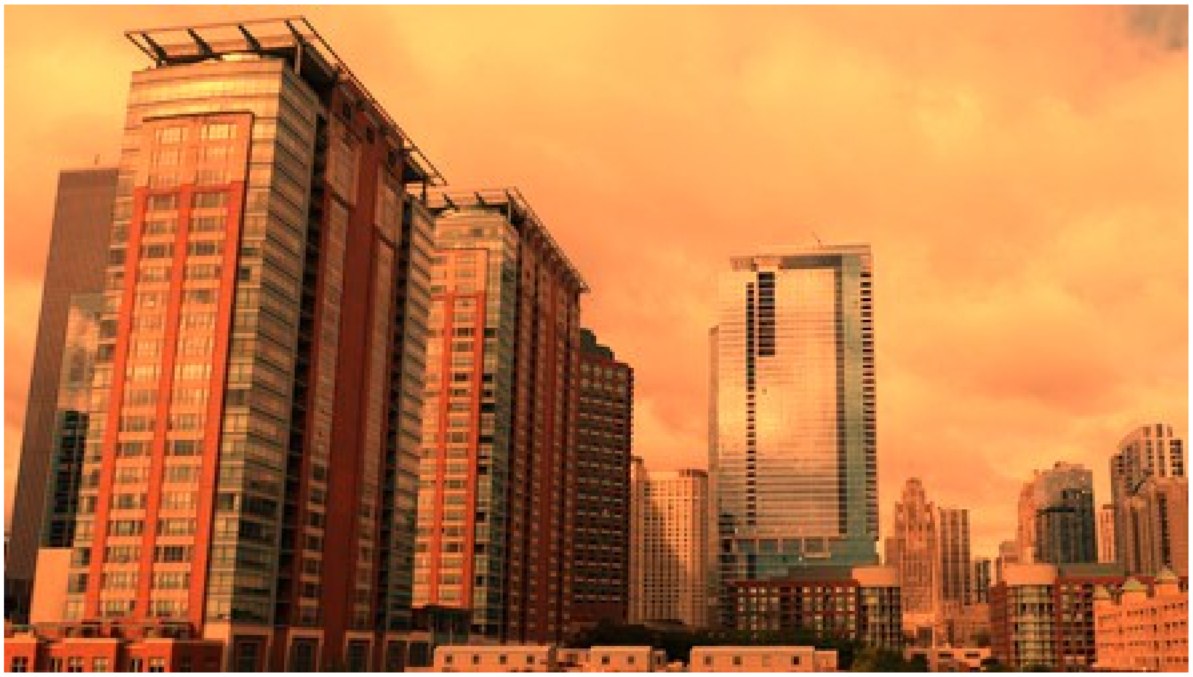
```
from wand.image import Image

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

(continues on next page)

(continued from previous page)

```
    # B >> 1
    img.evaluate(operator='rightshift', value=1, channel='blue')
    # R << 1
    img.evaluate(operator='leftshift', value=1, channel='red')
```



3.8.2 Function Expression

New in version 0.4.1.

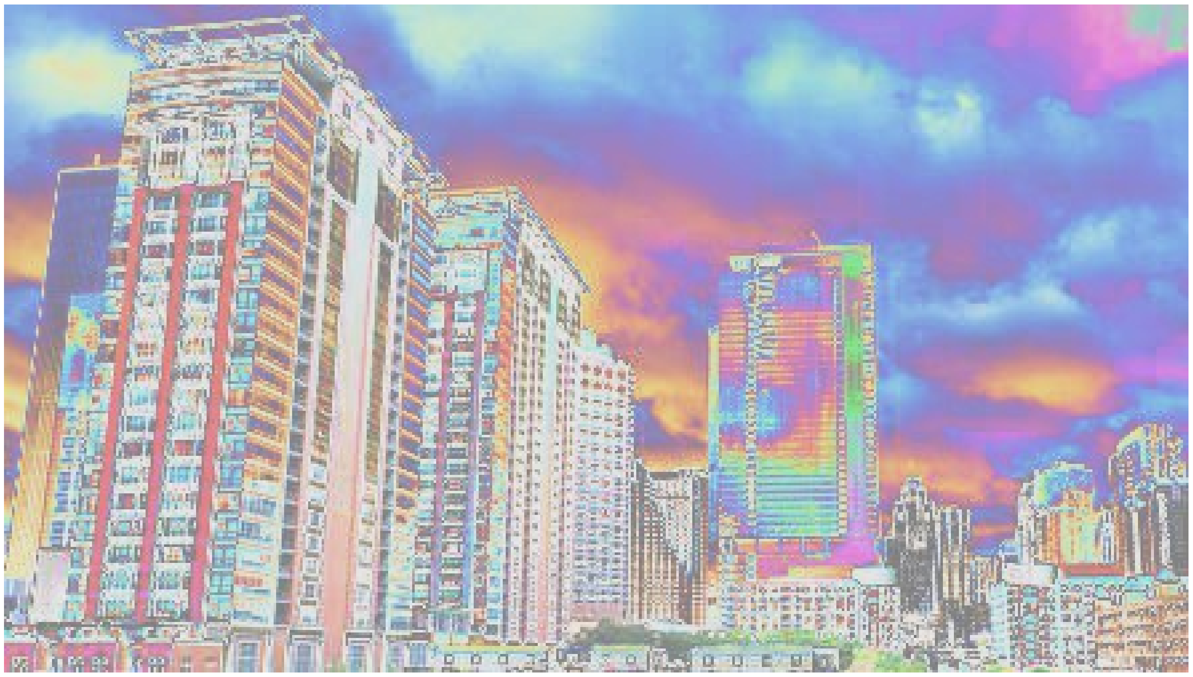
Similar to `evaluate()`, `function()` applies a multi-argument function to pixel channels. See `FUNCTION_TYPES` for a list of available function formulas.

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

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

```
from wand.image import Image

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



3.8.3 FX Expressions

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.

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

We can create a custom DIY filter that will turn the image black & white, except colors with a hue between 195° & 252°:

```
from wand.image import Image

fx_filter='(hue > 0.55 && hue < 0.7) ? u : lightness'

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

3.8.4 Gamma

New in version 0.4.1.

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

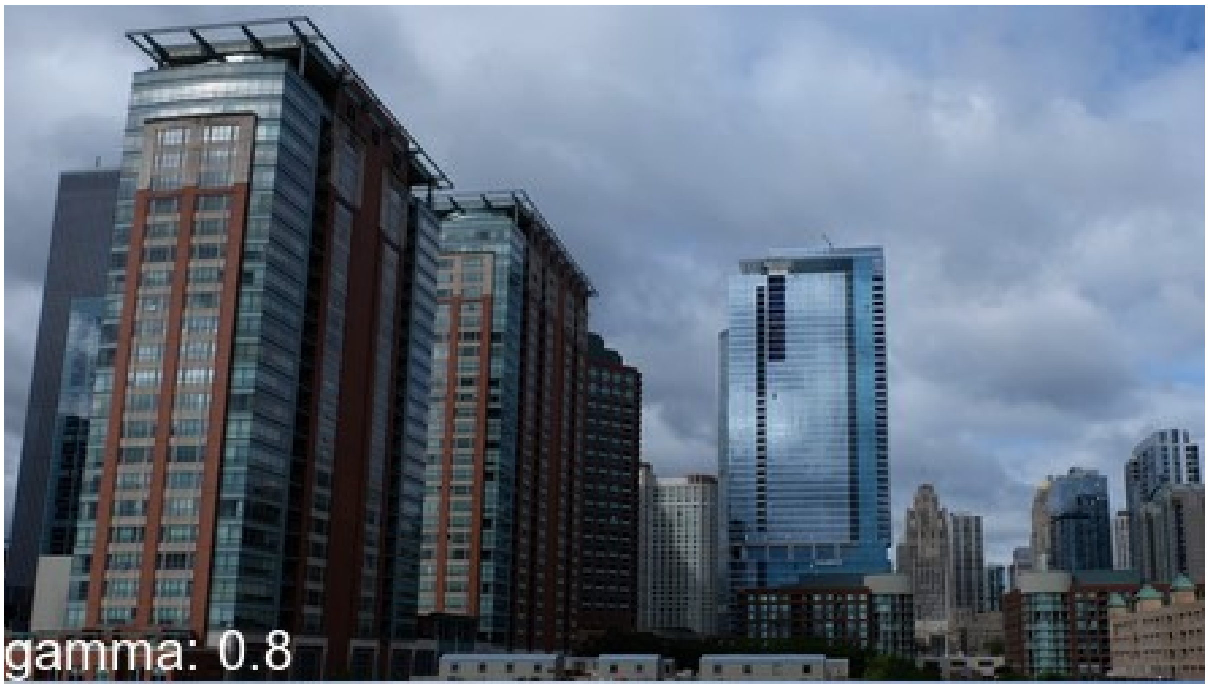
The `level()` method can also adjust `gamma` value.

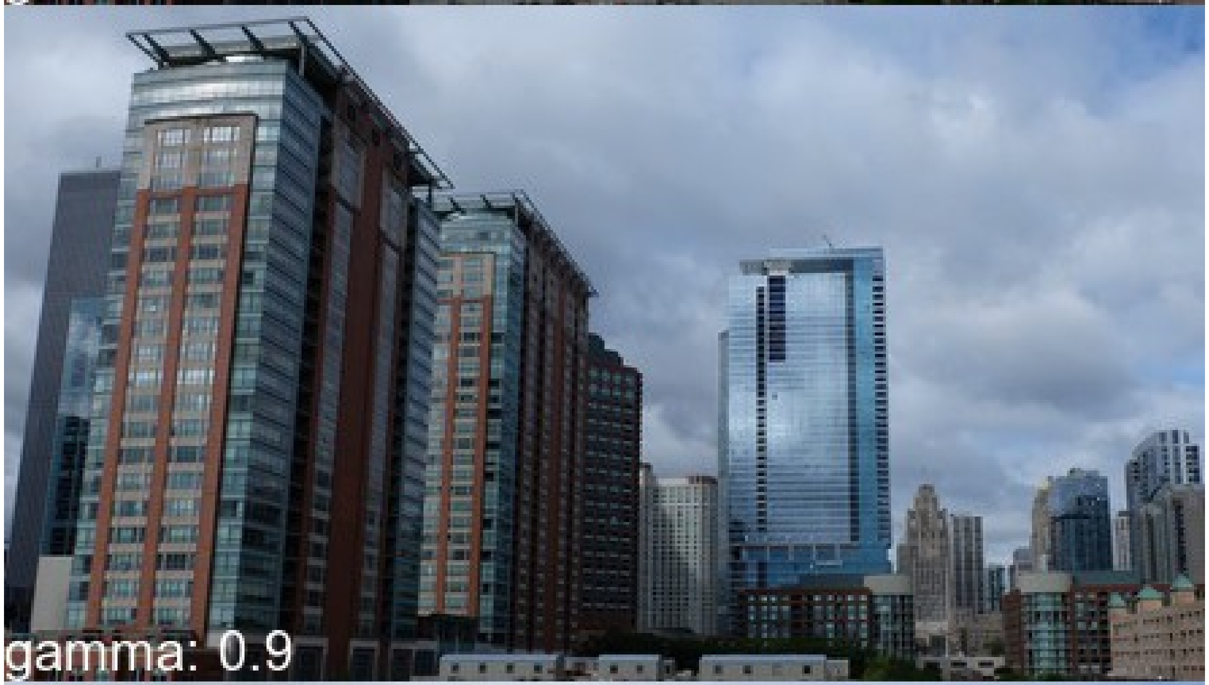
For example, when given image `enhancement.jpg`:

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

```
from wand.image import Image

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


gamma: 0.8


gamma: 0.9

3.8.5 Level

New in version 0.4.1.

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

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

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

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

```
from wand.image import Image

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

3.9 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.

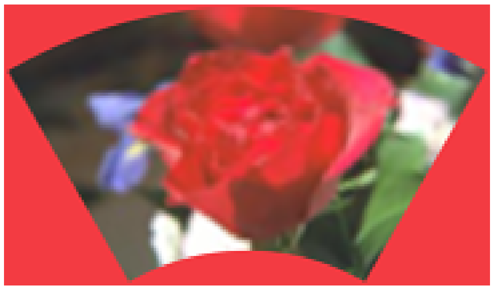
3.9.1 Controlling Resulting Images

Virtual Pixels

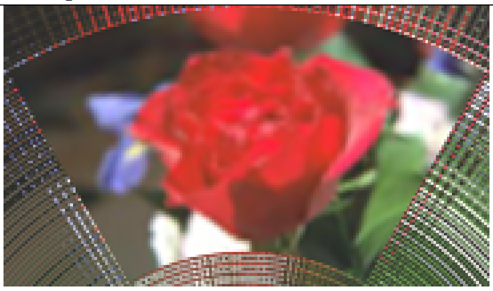
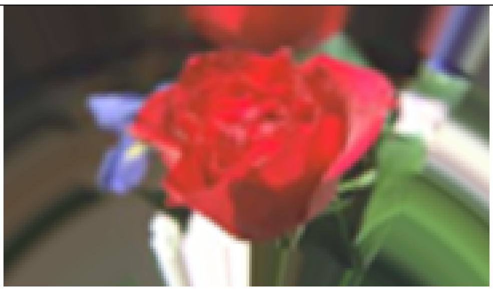
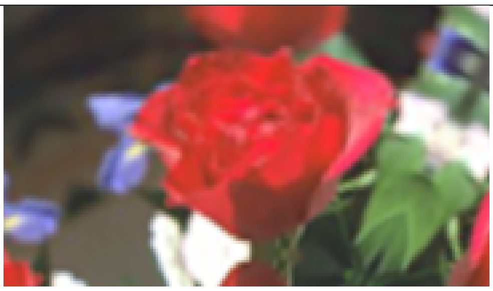
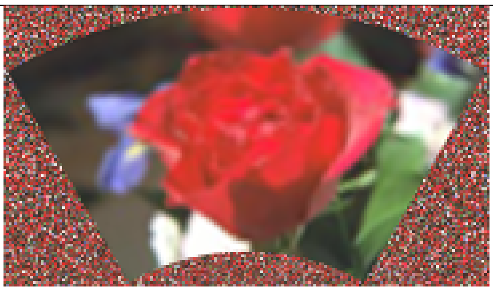
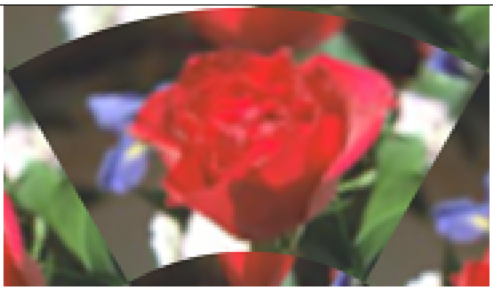
When performing distortion on raster images, the resulting image often includes pixels that are outside original bounding raster. These regions are referred to as vertical pixels, and can be controlled by setting `Image.virtual_pixel` to any value defined in `VIRTUAL_PIXEL_METHOD`.

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

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



Other `virtual_pixel` values can create special effects.

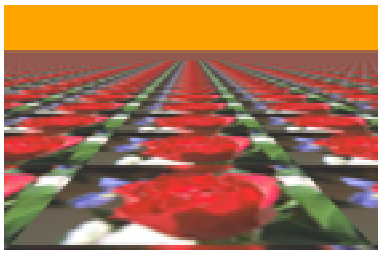
Virtual Pixel	Example
`dither`	
`edge`	
`mirror`	
`random`	
`tile`	

Matte Color

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

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

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



Rendering Size

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

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

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

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

3.9.2 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.

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

For example...

A single argument would be treated as an angle:

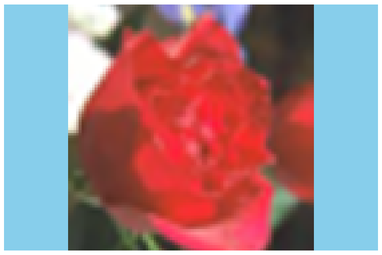
```
from wand.color import Color
from wand.image import Image

```

(continues on next page)

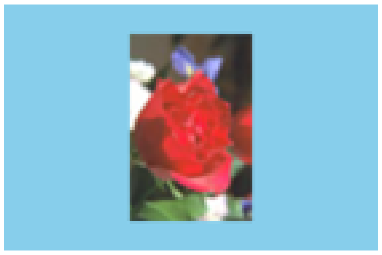
(continued from previous page)

```
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,))
```



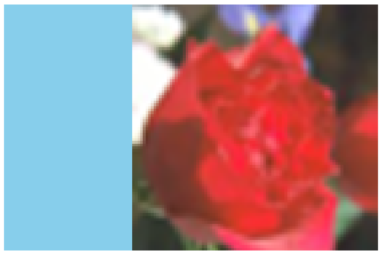
Two arguments would be treated as a scale & angle:

```
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.

3.9.3 Perspective

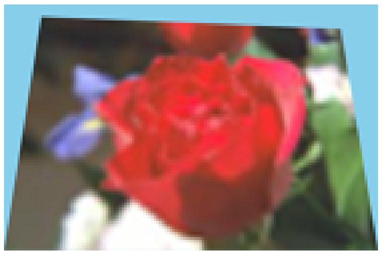
Perspective distortion requires 4 pairs of points which is a total of 16 doubles. The order of the `arguments` are groups of source & destination coordinate pairs.

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

For example:

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

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



3.9.4 Affine

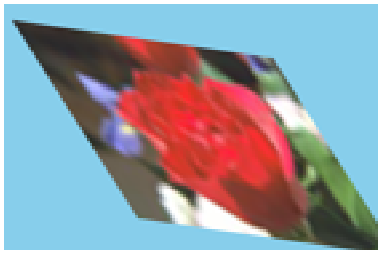
Affine distortion performs a shear operation. The arguments are similar to perspective, but only need a pair of 3 points, or 12 real numbers.

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

For example:

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

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



3.9.5 Affine Projection

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

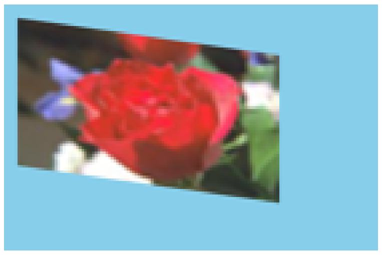
`Scale$_x$, Rotate$_x$, Rotate$_y$, Scale$_y$, Translate$_x$, Translate$_y$`

For example:

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

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

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



3.10 Drawing

New in version 0.3.0.

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

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

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

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

3.10.1 Arc

New in version 0.4.0.

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

An example:

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

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

3.10.2 Bezier

New in version 0.4.0.

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

For example:

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

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

Control width & color of curve with the drawing properties:

- `stroke_color`
- `stroke_width`

3.10.3 Circle

New in version 0.4.0.

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

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

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)
```

3.10.4 Color & Matte

New in version 0.4.0.

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

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

Example fill all to green boarder:

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

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

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

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

3.10.5 Composite

New in version 0.4.0.

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

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

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

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

3.10.6 Ellipse

New in version 0.4.0.

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

An example of a full ellipse:

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

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

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

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

3.10.7 Lines

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

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

Or you can turn this diagonal line upside down:

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

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

```
from wand.color import Color

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

3.10.8 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 accross top-left to center
    draw.path_curve(to=(40, 0),
                    controls=[(10, -40), (30,-40)],
                    relative=True)
    # Continue curve accross bottom-right
    draw.path_curve(to=(40, 0),
                    controls=(30, 40),
                    smooth=True,
                    relative=True)
    # Line to top-right
    draw.path_vertical_line(10)
    # Diagonal line to bottom-left
    draw.path_line(to=(10, 90))
    # Close first & last points
    draw.path_close()
    draw.path_finish()
    with Image(width=100, height=100, background=Color('lightblue')) as image:
        draw(image)
```

3.10.9 Point

New in version 0.4.0.

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

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

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

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

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

- `fill_color`

3.10.10 Polygon

New in version 0.4.0.

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

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

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

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

Control the fill & stroke with the following properties:

- `stroke_color`
- `stroke_dash_array`
- `stroke_dash_offset`

- stroke_line_cap
- stroke_line_join
- stroke_miter_limit
- stroke_opacity
- stroke_width
- fill_color
- fill_opacity
- fill_rule

3.10.11 Polyline

New in version 0.4.0.

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

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

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

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

Control the fill & stroke with the following properties:

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

3.10.12 Push & Pop

New in version 0.4.0.

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

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

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

3.10.13 Rectangles

New in version 0.3.6.

Changed in version 0.4.0.

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

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

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

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

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

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

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

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

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

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

3.10.14 Texts

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

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

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

- `font`
- `font_family`
- `font_resolution`
- `font_size`
- `font_stretch`
- `font_style`
- `font_weight`
- `gravity`
- `text_alignment`
- `text_antialias`
- `text_decoration`

- text_direction
- text_interline_spacing
- text_interword_spacing
- text_kerning
- text_under_color

3.10.15 Word Wrapping

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

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

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


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


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

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

    def shrink_text():
        """Reduce point-size & restore original text"""
        ctx.font_size = ctx.font_size - 0.75
        mutable_message = text

    while ctx.font_size > 0 and iteration_attempts:
        iteration_attempts -= 1
        width, height = eval_metrics(mutable_message)
        if height > roi_height:
            shrink_text()
        elif width > roi_width:
            columns = len(mutable_message)
            while columns > 0:
```

(continues on next page)

(continued from previous page)

```
                columns -= 1
                mutable_message = '\n'.join(wrap(mutable_message, columns))
                wrapped_width, _ = eval_metrics(mutable_message)
                if wrapped_width <= roi_width:
                    break
            if columns < 1:
                shrink_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')
```

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

3.11 Reading EXIF

New in version 0.3.0.

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

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

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

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

3.11.1 Image Profiles

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

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

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

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

3.12 Layers

3.12.1 Coalesce Layers

New in version 0.5.0.

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

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

3.12.2 Optimizing Layers

New in version 0.5.0.

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

To start, we can quickly create an animated gif.

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

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

(continued from previous page)

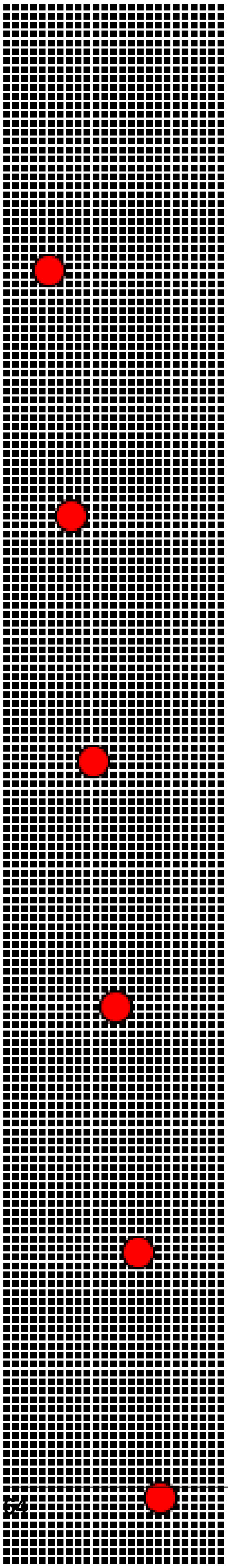
```
            canvas.sequence.append(frame)
    canvas.save(filename='layers.gif')
```

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

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

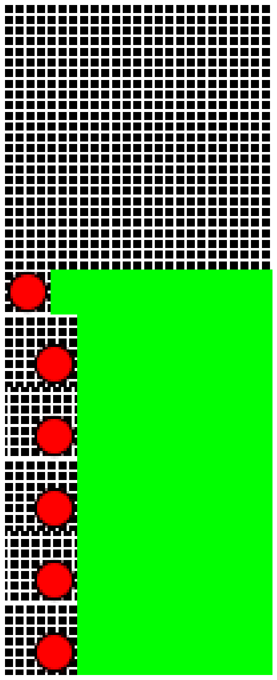
We can debug the previously created `layers.gif` by running the following:

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



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

```
>>> with Image(filename='layers.gif') as img:
...     img.optimize_layers()
...     debug_layers(img, 'layers-optmized-layers.png')
Debugging to file layers-optmized-layers.png
Frame 0 size : (100, 100) page: (100, 100, 0, 0)
Frame 1 size : (17, 17) page: (100, 100, 12, 12)
Frame 2 size : (26, 27) page: (100, 100, 12, 12)
Frame 3 size : (26, 27) page: (100, 100, 23, 22)
Frame 4 size : (26, 27) page: (100, 100, 32, 32)
Frame 5 size : (26, 27) page: (100, 100, 43, 42)
Frame 6 size : (26, 27) page: (100, 100, 52, 52)
```



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

3.12.3 Optimizing Transparency

New in version 0.5.0.

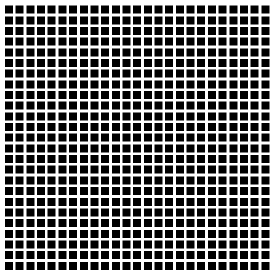
Following the above examples, we can also optimize by forcing pixels transparent if they are unchanged since the previous frame.

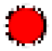
```
>>> with Image(filename='layers.gif') as img:
...    img.optimize_transparency()
...    debug_layers(img, 'layers-optmized-transparent.png')
Debugging to file layers-optmized-transparent.png
Frame 0 size : (100, 100) page: (100, 100, 0, 0)
Frame 1 size : (100, 100) page: (100, 100, 0, 0)
Frame 2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

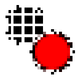
(continues on next page)

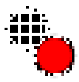
(continued from previous page)

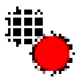
```
Frame 3 size : (100, 100) page: (100, 100, 0, 0)
Frame 4 size : (100, 100) page: (100, 100, 0, 0)
Frame 5 size : (100, 100) page: (100, 100, 0, 0)
Frame 6 size : (100, 100) page: (100, 100, 0, 0)
```

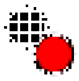


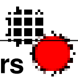








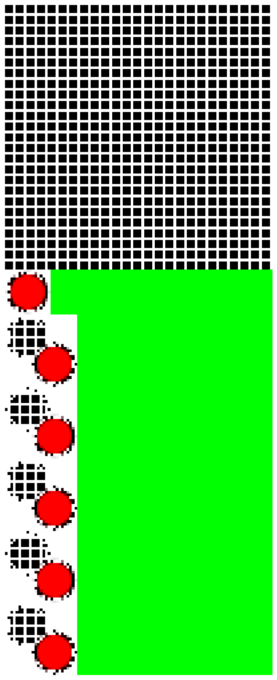




Notice both the size of each frame, and the page offset are unchanged. This technique only really saves if the subject already contains transparency color channels, and so most modern gif animations would not benefit from this method.

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

```
>>> with Image(filename='layers.gif') as img:
...     img.optimize_layers()
...     img.optimize_transparency()
...     debug_layers(img, 'layers-optmized-layers-transparent.png')
Debugging to file layers-optmized-layers-transparent.png
Frame 0 size : (100, 100) page: (100, 100, 0, 0)
Frame 1 size : (17, 17) page: (100, 100, 12, 12)
Frame 2 size : (26, 27) page: (100, 100, 12, 12)
Frame 3 size : (26, 27) page: (100, 100, 23, 22)
Frame 4 size : (26, 27) page: (100, 100, 32, 32)
Frame 5 size : (26, 27) page: (100, 100, 43, 42)
Frame 6 size : (26, 27) page: (100, 100, 52, 52)
```



Note: Lime-green background added for visibility cue.

3.13 Sequence

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

New in version 0.3.0.

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

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

3.13.1 `sequence` is a `Sequence`

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

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

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

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

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

Or slice it:

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

3.13.2 `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.

3.13.3 Manipulating `SingleImage`

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

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

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

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

3.14 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.

3.15 Running tests

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

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

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

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

```
$ pip install pytest
$ py.test
```

3.15.1 Skipping tests

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

```
$ py.test --skip-slow
```

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

```
$ py.test --skip-pdf
```

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

```
$ py.test -k image
```

3.15.2 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
```

3.15.3 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

3.15.4 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

3.16 Roadmap

3.16.1 Very future versions

CFFI Wand will move to CFFI from ctypes.

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

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

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

- Global monkeypatcher which changes `sys.modules`:

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

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

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

3.17 Wand Changelog

3.17.1 0.5 series

Version 0.5.5

Unreleased.

Version 0.5.4

Released on May 25th, 2019.

- Rewrote *`libc`* library loader. [#409]
- Respect `background` paramater in `Image.__init__()` constructor. [#410]
- Fixed `Drawing.get_font_metrics()` not raising internal ImageMagick exception on rendering error. [#411]

- Fixed deleting image artifact value.
- Fixed offset memory calculation in `Image.export_pixels()` & `Image.import_pixels()` methods. [#413]
- Added `Image.auto_gamma()` method.
- Added `Image.auto_level()` method.
- Added `Image.border_color` property.
- Added `Image.brightness_contrast()` method.
- Added `Image.mode()` method.
- Added `Image.motion_blur()` method.
- Added `Image.oil_paint()` method.
- Added `Image.opaque_paint()` method.
- Added `Image.polaroid()` method.
- Added `Image.rendering_intent` property.
- Added `Image.rotational_blur()` method.
- Added `Image.scene` property.
- Added `Image.shear()` method.
- Added `Image.sigmoidal_contrast()` method.
- Added `Image.similarity()` method.
- Added `Image.stegano()` method.
- Added `Image.stereogram()` class method.
- Added `Image.texture()` method.
- Added `Image.thumbnail()` method. [#357 by yoch]
- Added `Image.ticks_per_second` property.

Version 0.5.3

Released on April 20, 2019.

- Fixed alpha channel set to "on" & "off" values for ImageMagick-7. [#404]
- Updated `Image.composite` & `Image.composite_channel` to include optional arguments for composite methods that require extra controls.
- Updated `Image.composite` & `Image.composite_channel` to include optional gravity argument.
- **Support for numpy arrays. [#65]**
  - Added `Image.from_array` class method.
- **Support color map / palette manipulation. [#403]**
  - Added `Image.colors` property.
  - Added `Image.color_map()` method.
  - Added `Image.cycle_color_map()` method.
- Support for `highlight` & `lowlight` has been added to `Image.compare()` method.

- Support for PEP-519 for objects implementing `__fspath__`, in `encode_filename()`.
- Added `Image.adaptive_blur()` method.
- Added `Image.adaptive_resize()` method.
- Added `Image.adaptive_sharpen()` method.
- Added `Image.adaptive_threshold()` method.
- Added `Image.black_threshold()` method.
- Added `Image.blue_shift()` method.
- Added `Image.charcoal()` method.
- Added `Image.color_matrix()` method.
- Added `Image.colorize()` method.
- Added `Image.fuzz` property.
- Added `Image.kurtosis` property.
- Added `Image.kurtosis_channel()` method
- Added `Image.maxima` property.
- Added `Image.mean` property.
- Added `Image.mean_channel()` method
- Added `Image.minima` property.
- Added `Image.noise()` method.
- Added `Image.range_channel()` method
- Added `Image.remap()` method.
- Added `Image.selective_blur()` method.
- Added `Image.skewness` property.
- Added `Image.sketch()` method.
- Added `Image.smush()` method.
- Added `Image.sparse_color()` method.
- Added `Image.splice()` method.
- Added `Image.spread()` method.
- Added `Image.standard_deviation` property.
- Added `Image.statistic()` method.
- Added `Image.tint()` method.

Special thanks to Fred Weinhaus for helping test this release.

Version 0.5.2

Released on March 24, 2019.

- Import `collections.abc` explicitly. [#398 by Stefan Naumann]
- Fixed memory leak in `HistogramDict`. [#397]

- Fixed compression & compression quality bug. [#202 & #278]
- `Image.read()` will raise *WandRuntimeError* if `MagickReadImage()` returns `MagickFalse`, but does not emit exception. [#319]
- Added `Image.implode()` method.
- Added `Image.vignette()` method.
- Added `Image.wave()` method.
- Added `Image.white_threshold()` method.
- Added `Image.blue_primary` property.
- Added `Image.green_primary` property.
- Added `Image.interlace_scheme` property.
- Added `Image.interpolate_method` property.
- Added `Image.red_primary` property.
- Added `Image.white_point` property.

Version 0.5.1

Released on February 15, 2019.

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

Version 0.5.0

Released on January 1, 2019.

- Support for ImageMagick-7.
- Improved support for 32-bit systems.
- Improved support for non-Q16 libraries.
- Removed *README.rst* from setup.py's *data_files*. [#336]
- Improved *EXIF:ORIENTATION* handling. [#364 by M. Skrzypek]

- Tolerate failures while accessing wand.api. [#220 by Utkarsh Upadhyay]
- Added support for Image Artifacts through `Image.artifacts`. [#369]
- Added optional stroke color/width parameters for `Font`.
- Image layers support (#22)
  - Added `Image.coalesce()` method.
  - Added `Image.deconstruct` method.
  - Added `Image.dispose` property.
  - Added `Image.optimize_layers()` method.
  - Added `Image.optimize_transparency()` method.
- Implemented `__array_interface__()` for NumPy [#65]
- Migrated the following methods & attributes from `Image` to `BaseImage` for a more uniformed code-base.
  - `Image.compression`
  - `Image.format`
  - `Image.auto_orient()`
  - `Image.border()`
  - `Image.contrast_stretch()`
  - `Image.gamma()`
  - `Image.level()`
  - `Image.linear_stretch()`
  - `Image.normalize()`
  - `Image.strip()`
  - `Image.transpose()`
  - `Image.transverse()`
  - `Image.trim()`
- Added `Image.clut()` method.
- Added `Image.concat()` method. [#177]
- Added `Image.deskew()` method.
- Added `Image.despeckle()` method.
- Added `Image.edge()` method.
- Added `Image.emboss()` method. [#196]
- Added `Image.enhance()` method. [#132]
- Added `Image.export_pixels()` method.
- Added `Image.import_pixels()` method.
- Added `Image.morphology()` method. [#132]
- Added `Image.posterize()` method.
- Added `Image.shade()` method.

- Added `Image.shadow()` method.
- Added `Image.sharpen()` method. [#132]
- Added `Image.shave()` method.
- Added `Image.unique_colors()` method.
- Method `Drawing.draw()` now accepts `BaseImage` for folks extended classes.
- Added `Image.loop` property. [#227]
- Fixed `SingleImage.delay` property. [#153]
- Attribute `Image.font_antialias` has been deprecated in favor of `Image.antialias`. [#218]
- Fixed ordering of `COMPRESSION_TYPES` based on ImageMagick version. [#309]
- Fixed drawing on `SingleImage`. [#289]
- Fixed wrapping issue for larger offsets when using *gravity* kwarg in `Image.crop()` method. [#367]

3.17.2 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 (`MagickTransformImageColorspace()`). [#152 by Adrian Jung, #262 by Jeong YunWon]
- Now ImageMagick DLL can be loaded on Windows even if its location is stored in the 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 This guide introduces what's new in Wand 0.4.

- Complete `wand.drawing` API. The whole work was done by Eric McConville. Huge thanks for his effort! [#194 by Eric McConville]
  - Added `Drawing.arc()` method (*Arc*).
  - Added `Drawing.bezier()` method (*Bezier*).
  - Added `Drawing.circle()` method (*Circle*).
  - *Color & Matte*
    * Added `wand.drawing.PAINT_METHOD_TYPES` constant.
    * Added `Drawing.color()` method.
    * Added `Drawing matte()` method.
  - Added `Drawing.composite()` method (*Composite*).
  - Added `Drawing.ellipse()` method (*Ellipse*).
  - *Paths*
    * Added `path_start()` method.
    * Added `path_finish()` method.
    * Added `path_close()` method.
    * Added `path_curve()` method.
    * Added `path_curve_to_quadratic_bezier()` method.
    * Added `path_elliptic_arc()` method.
    * Added `path_horizontal_line()` method.
    * Added `path_line()` method.
    * Added `path_move()` method.
    * Added `path_vertical_line()` method.
  - Added `Drawing.point()` method (*Point*).
  - Added `Drawing.polygon()` method (*Polygon*).
  - Added `Drawing.polyline()` method (*Polyline*).
  - *Push & Pop*

        * Added `push()` method.
        * Added `push_clip_path()` method.
        * Added `push_defs()` method.
        * Added `push_pattern()` method.
        * Added `clip_path` property.
        * Added `set_fill_pattern_url()` method.
        * Added `set_stroke_pattern_url()` method.
        * Added `pop()` method.
    - Added `Drawing.rectangle()` method (*Rectangles*).
    - Added `stroke_dash_array` property.
    - Added `stroke_dash_offset` property.
    - Added `stroke_line_cap` property.
    - Added `stroke_line_join` property.
    - Added `stroke_miter_limit` property.
    - Added `stroke_opacity` property.
    - Added `stroke_width` property.
    - Added `fill_opacity` property.
    - Added `fill_rule` property.
- Error message of *`MissingDelegateError`* raised by `Image.liquid_rescale()` became nicer.

3.17.3 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.
- Ranamed `wand.image.COMPOSITE_OPS` to `COMPOSITE_OPERATORS`.
- Now it shows helpful error message when ImageMagick library cannot be found.
- Added IPython-specialized formatter.
- Added `QUANTUM_DEPTH` constant.
- Added these properties to `Color` class:
  - `red_quantum`
  - `green_quantum`
  - `blue_quantum`
  - `alpha_quantum`
  - `red_int8`
  - `green_int8`
  - `blue_int8`
  - `alpha_int8`
- Added `Image.normalize()` method. [#95 by Michael Curry]
- Added `Image.transparent_color()` method. [#98 by Lionel Koenig]
- Started supporting resizing and cropping of GIF images. [#88 by Bear Dong, #112 by Taeho Kim]
- Added `Image.flip()` method.
- Added `Image.flop()` method.
- Added `Image.orientation` property. [88574468a38015669dae903185fb328abdd717c0 by Taeho Kim]
- `wand.resource.DestroyedResourceError` becomes a subtype of `wand.exceptions.WandException`.
- `Color` is now hashable, so can be used as a key of dictionaries, or an element of sets. [#114 by klutzy]
- `Color` has `normalized_string` property.
- `Image` has `histogram` dictionary.
- Added optional `fuzz` parameter to `Image.trim()` method. [#113 by Evaldo Junior]

3.17.4 0.2 series

Version 0.2.4

Released on May 28, 2013.

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

Version 0.2.3

Released on January 25, 2013.

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

Version 0.2.2

Released on September 24, 2012.

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

Version 0.2.1

Released on August 19, 2012. Beta version.

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

  - *MAGICK_VERSION_INFO* (GetMagickVersion())
  - *MAGICK_VERSION_NUMBER* (GetMagickVersion())
  - *MAGICK_RELEASE_DATE* (GetMagickReleaseDate())
  - *MAGICK_RELEASE_DATE_STRING* (GetMagickReleaseDate())

Version 0.2.0

Released on June 20, 2012. Alpha version.

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

3.17.5 0.1 series

Version 0.1.10

Released on May 8, 2012. Still alpha version.

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

Version 0.1.9

Released on December 23, 2011. Still alpha version.

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

- Changed the theme of Sphinx documentation.
- Changed the start example of the documentation.

Version 0.1.8

Released on December 2, 2011. Still alpha version.

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

Version 0.1.7

Released on November 10, 2011. Still alpha version.

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

Version 0.1.6

Released on October 31, 2011. Still alpha version.

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

Version 0.1.5

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

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

Version 0.1.4

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

Version 0.1.3

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

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

Version 0.1.2

Released on October 16, 2011. Still alpha version.

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

Version 0.1.1

Released on October 4, 2011. Still alpha version.

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

Version 0.1.0

Released on October 1, 2011. Very alpha version.

3.18 Talks and Presentations

3.18.1 Talks in 2012

- Lightning talk at Python Korea November 2012

CHAPTER 4

References

4.1 `wand` — Simple MagickWand API binding for Python

4.1.1 `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.

wand.exceptions.**CODE_MAP = [(<class 'wand.exceptions.BaseWarning'>, 'Warning'), (<class 'wa**

(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.

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 e**
(list) A list of error/warning domains, these descriptions and codes. The form of elements is like: (domain name, description, codes).

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

An ImageMagick delegate failed to complete.

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

An ImageMagick delegate failed to complete.

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

An ImageMagick delegate failed to complete.

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

A drawing operation failed.

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

A drawing operation failed.

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

A drawing operation failed.

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

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

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

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

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

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

exception wand.exceptions.**ImageError**
Bases: *wand.exceptions.BaseError*

The operation could not complete due to an incompatible image.

exception `wand.exceptions.`**`ImageFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*

The operation could not complete due to an incompatible image.

exception `wand.exceptions.`**`ImageWarning`**
Bases: *`wand.exceptions.BaseWarning`*

The operation could not complete due to an incompatible image.

exception `wand.exceptions.`**`MissingDelegateError`**
Bases: *`wand.exceptions.BaseError`*, `ImportError`

The image type can not be read or written because the appropriate; delegate is missing.

exception `wand.exceptions.`**`MissingDelegateFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*, `ImportError`

The image type can not be read or written because the appropriate; delegate is missing.

exception `wand.exceptions.`**`MissingDelegateWarning`**
Bases: *`wand.exceptions.BaseWarning`*, `ImportError`

The image type can not be read or written because the appropriate; delegate is missing.

exception `wand.exceptions.`**`ModuleError`**
Bases: *`wand.exceptions.BaseError`*

There was a problem with an image module.

exception `wand.exceptions.`**`ModuleFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*

There was a problem with an image module.

exception `wand.exceptions.`**`ModuleWarning`**
Bases: *`wand.exceptions.BaseWarning`*

There was a problem with an image module.

exception `wand.exceptions.`**`MonitorError`**
Bases: *`wand.exceptions.BaseError`*

There was a problem activating the progress monitor.

exception `wand.exceptions.`**`MonitorFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*

There was a problem activating the progress monitor.

exception `wand.exceptions.`**`MonitorWarning`**
Bases: *`wand.exceptions.BaseWarning`*

There was a problem activating the progress monitor.

exception `wand.exceptions.`**`OptionError`**
Bases: *`wand.exceptions.BaseError`*

A command-line option was malformed.

exception `wand.exceptions.`**`OptionFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*

A command-line option was malformed.

exception `wand.exceptions.`**`OptionWarning`**
Bases: *`wand.exceptions.BaseWarning`*

A command-line option was malformed.

exception `wand.exceptions.`**`PolicyError`**
Bases: *`wand.exceptions.BaseError`*

A policy denies access to a delegate, coder, filter, path, or resource.

exception `wand.exceptions.`**`PolicyFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*

A policy denies access to a delegate, coder, filter, path, or resource.

exception `wand.exceptions.`**`PolicyWarning`**
Bases: *`wand.exceptions.BaseWarning`*

A policy denies access to a delegate, coder, filter, path, or resource.

exception `wand.exceptions.`**`RandomError`**
Bases: *`wand.exceptions.BaseError`*

There is a problem generating a true or pseudo-random number.

exception `wand.exceptions.`**`RandomFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*

There is a problem generating a true or pseudo-random number.

exception `wand.exceptions.`**`RandomWarning`**
Bases: *`wand.exceptions.BaseWarning`*

There is a problem generating a true or pseudo-random number.

exception `wand.exceptions.`**`RegistryError`**
Bases: *`wand.exceptions.BaseError`*

There was a problem getting or setting the registry.

exception `wand.exceptions.`**`RegistryFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*

There was a problem getting or setting the registry.

exception `wand.exceptions.`**`RegistryWarning`**
Bases: *`wand.exceptions.BaseWarning`*

There was a problem getting or setting the registry.

exception `wand.exceptions.`**`ResourceLimitError`**
Bases: *`wand.exceptions.BaseError`*, `MemoryError`

A program resource is exhausted e.g. not enough memory.

exception `wand.exceptions.`**`ResourceLimitFatalError`**
Bases: *`wand.exceptions.BaseFatalError`*, `MemoryError`

A program resource is exhausted e.g. not enough memory.

exception `wand.exceptions.`**`ResourceLimitWarning`**
Bases: *`wand.exceptions.BaseWarning`*, `MemoryError`

A program resource is exhausted e.g. not enough memory.

exception wand.exceptions.**StreamError**
Bases: *wand.exceptions.BaseError*, OSError

There was a problem reading or writing from a stream.

exception wand.exceptions.**StreamFatalError**
Bases: *wand.exceptions.BaseFatalError*, OSError

There was a problem reading or writing from a stream.

exception wand.exceptions.**StreamWarning**
Bases: *wand.exceptions.BaseWarning*, OSError

There was a problem reading or writing from a stream.

wand.exceptions.**TYPE_MAP = {300: <class 'wand.exceptions.ResourceLimitWarning'>, 305: <cl**
(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.

4.1.2 `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**

4.1.3 `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/use**
(`module`) Module containing abstract base classes. `collections` in Python 2 and `collections.abc` in Python 3.

wand.compat.**binary**(*string*, *var=None*)
Makes `string` to `str` in Python 2. Makes `string` to `bytes` in Python 3.

Parameters

- **string** (bytes, str, unicode) – a string to cast it to *binary_type*
- **var** (str) – an optional variable name to be used for error message

wand.compat.**binary_type**
alias of builtins.bytes

wand.compat.**encode_filename** (*filename*)
If filename is a *text_type*, encode it to *binary_type* according to filesystem's default encoding.

Changed in version 0.5.3: Added support for PEP-519 https://github.com/emcconville/wand/pull/339

wand.compat.**file_types**
alias of io.RawIOBase

wand.compat.**string_type**
alias of builtins.str

wand.compat.**text_type**
alias of builtins.str

wand.compat.**xrange**
alias of builtins.range

4.1.4 wand.version — Version data

You can find the current version in the command line interface:

```
$ python -m wand.version
0.5.4
$ python -m wand.version --verbose
Wand 0.5.4
ImageMagick 6.7.7-6 2012-06-03 Q16 http://www.imagemagick.org
$ python -m wand.version --config | grep CC | cut -d : -f 2
gcc -std=gnu99 -std=gnu99
$ python -m wand.version --fonts | grep Helvetica
Helvetica
Helvetica-Bold
Helvetica-Light
Helvetica-Narrow
Helvetica-Oblique
$ python -m wand.version --formats | grep CMYK
CMYK
CMYKA
```

New in version 0.2.0: The command line interface.

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

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

wand.version.**VERSION = '0.5.4'**
(basestring) The version string e.g. '0.1.2'.

Changed in version 0.1.9: Becomes string. (It was tuple before.)

wand.version.**VERSION_INFO = (0, 5, 4)**
(tuple) The version tuple e.g. (0, 1, 2).

Changed in version 0.1.9: Becomes tuple. (It was string before.)

`wand.version.`**`MAGICK_VERSION = None`**

(`basestring`) The version string of the linked ImageMagick library. The exactly same string to the result of `GetMagickVersion()` function.

Example:

```
'ImageMagick 6.7.7-6 2012-06-03 Q16 http://www.imagemagick.org'
```

New in version 0.2.1.

`wand.version.`**`MAGICK_VERSION_FEATURES = 'Cipher DPC Modules OpenMP '`**

(`basestring`) A string of all features enabled. This value is identical to what is returned by `GetMagickFeatures()`

New in version 0.5.0.

`wand.version.`**`MAGICK_VERSION_INFO = None`**

(`tuple`) The version tuple e.g. `(6, 7, 7, 6)` of *`MAGICK_VERSION`*.

New in version 0.2.1.

`wand.version.`**`MAGICK_VERSION_NUMBER = None`**

(`numbers.Integral`) The version number of the linked ImageMagick library.

New in version 0.2.1.

`wand.version.`**`MAGICK_RELEASE_DATE = None`**

(`datetime.date`) The release date of the linked ImageMagick library. Equivalent to the result of `GetMagickReleaseDate()` function.

New in version 0.2.1.

`wand.version.`**`MAGICK_RELEASE_DATE_STRING = None`**

(`basestring`) The date string e.g. `'2012-06-03'` of *`MAGICK_RELEASE_DATE_STRING`*. This value is the exactly same string to the result of `GetMagickReleaseDate()` function.

New in version 0.2.1.

`wand.version.`**`MAGICK_HDRI = None`**

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

`wand.version.`**`QUANTUM_DEPTH = None`**

(`numbers.Integral`) The quantum depth configuration of the linked ImageMagick library. One of 8, 16, 32, or 64.

New in version 0.3.0.

`wand.version.`**`QUANTUM_RANGE = None`**

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

New in version 0.5.0.

`wand.version.`**`configure_options`**(*pattern='*'*)

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

Example: Find where the ImageMagick documents are installed:

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

Parameters **`pattern`** (`basestring`) – A term to filter queries against. Supports wildcard '*' characters. Default patterns '*' for all options.

Returns Directory of configuration options matching given pattern

Return type `collections.defaultdict`

`wand.version.`**`fonts`**`(`*pattern='*'*`)`

Queries ImageMagick library for available fonts.

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

Example: List all bold Helvetica fonts:

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

Parameters **`pattern`** (`basestring`) – A term to filter queries against. Supports wildcard '*' characters. Default patterns '*' for all options.

Returns Sequence of matching fonts

Return type `collections.Sequence`

`wand.version.`**`formats`**`(`*pattern='*'*`)`

Queries ImageMagick library for supported formats.

Example: List supported PNG formats:

```
>>> from wand.version import formats
>>> formats('PNG*')
['PNG', 'PNG00', 'PNG8', 'PNG24', 'PNG32', 'PNG48', 'PNG64']
```

Parameters **`pattern`** (`basestring`) – A term to filter formats against. Supports wildcards '*' characters. Default pattern '*' for all formats.

Returns Sequence of matching formats

Return type `collections.Sequence`

CHAPTER 5

Troubleshooting

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.

CHAPTER 6

Open source

Wand is an open source software initially written by Hong Minhee (for StyleShare), and is currently maintained by E. McConville. See also the complete list of contributors as well. The source code is distributed under MIT license and you can find it at GitHub repository. Check out now:

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

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

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, 89
wand.api, 95
wand.compat, 95
wand.exceptions, 89
wand.version, 96

Index

A

abc (*in module wand.compat*), 95
AffineMatrix (*class in wand.api*), 95

B

BaseError, 89
BaseFatalError, 89
BaseWarning, 89
binary() (*in module wand.compat*), 95
binary_type (*in module wand.compat*), 96
BlobError, 89
BlobFatalError, 89
BlobWarning, 90

C

CacheError, 90
CacheFatalError, 90
CacheWarning, 90
CODE_MAP (*in module wand.exceptions*), 90
CoderError, 90
CoderFatalError, 90
CoderWarning, 90
configure_options() (*in module wand.version*), 97
ConfigureError, 90
ConfigureFatalError, 90
ConfigureWarning, 90
CorruptImageError, 90
CorruptImageFatalError, 90
CorruptImageWarning, 91

D

DelegateError, 91
DelegateFatalError, 91
DelegateWarning, 91
DOMAIN_MAP (*in module wand.exceptions*), 91
DrawError, 91
DrawFatalError, 91
DrawWarning, 91

E

encode_filename() (*in module wand.compat*), 96
environment variable
  MAGICK_HOME, 11–13

F

file_types (*in module wand.compat*), 96
FileOpenError, 91
FileOpenFatalError, 91
FileOpenWarning, 91
fonts() (*in module wand.version*), 98
formats() (*in module wand.version*), 98

I

ImageError, 91
ImageFatalError, 91
ImageWarning, 92

L

libc (*in module wand.api*), 95
libmagick (*in module wand.api*), 95
library (*in module wand.api*), 95
load_library() (*in module wand.api*), 95

M

MAGICK_HDRI (*in module wand.version*), 97
MAGICK_HOME, 11–13
MAGICK_RELEASE_DATE (*in module wand.version*), 97
MAGICK_RELEASE_DATE_STRING (*in module wand.version*), 97
MAGICK_VERSION (*in module wand.version*), 96
MAGICK_VERSION_FEATURES (*in module wand.version*), 97
MAGICK_VERSION_INFO (*in module wand.version*), 97
MAGICK_VERSION_NUMBER (*in module wand.version*), 97
MagickPixelPacket (*class in wand.api*), 95

MissingDelegateError, 92
MissingDelegateFatalError, 92
MissingDelegateWarning, 92
ModuleError, 92
ModuleFatalError, 92
ModuleWarning, 92
MonitorError, 92
MonitorFatalError, 92
MonitorWarning, 92

O

OptionError, 92
OptionFatalError, 92
OptionWarning, 92

P

PixelInfo (*class in wand.api*), 95
PointInfo (*class in wand.api*), 95
PolicyError, 93
PolicyFatalError, 93
PolicyWarning, 93
PY3 (*in module wand.compat*), 95

Q

QUANTUM_DEPTH (*in module wand.version*), 97
QUANTUM_RANGE (*in module wand.version*), 97

R

RandomError, 93
RandomFatalError, 93
RandomWarning, 93
RegistryError, 93
RegistryFatalError, 93
RegistryWarning, 93
ResourceLimitError, 93
ResourceLimitFatalError, 93
ResourceLimitWarning, 93

S

StreamError, 93
StreamFatalError, 94
StreamWarning, 94
string_type (*in module wand.compat*), 96

T

text_type (*in module wand.compat*), 96
TYPE_MAP (*in module wand.exceptions*), 94
TypeError, 94
TypeFatalError, 94
TypeWarning, 94

V

VERSION (*in module wand.version*), 96
VERSION_INFO (*in module wand.version*), 96

W

wand (*module*), 89
wand.api (*module*), 95
wand.compat (*module*), 95
wand.exceptions (*module*), 89
wand.version (*module*), 96
WandError, 94
WandException, 94
WandFatalError, 94
WandLibraryVersionError, 94
WandRuntimeError, 94
WandWarning, 94

X

xrange (*in module wand.compat*), 96
XServerError, 94
XServerFatalError, 95
XServerWarning, 95