Wand Documentation

Release 0.2.4

Hong Minhee

August 04, 2013

CONTENTS

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
$ easy_install Wand
```

CHAPTER

ONE

WHY JUST ANOTHER BINDING?

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

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

CHAPTER

TWO

REQUIREMENTS

- Python 2.6 or higher
  - CPython 2.6 or higher
  - PyPy 1.5 or higher
- MagickWand library
  - `libmagickwand-dev` for APT on Debian/Ubuntu
  - `imagemagick` for MacPorts/Homebrew on Mac
  - `ImageMagick-devel` for Yum on CentOS

CHAPTER

THREE

USER'S GUIDE

3.1 Installation

Wand itself can be installed from PyPI using **easy_install** or **pip**:

```
$ easy_install Wand  # or
$ pip install Wand
```

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

- *Debian/Ubuntu*
- *Fedora/CentOS*
- *Mac*
- *Windows*
- *FreeBSD*

3.1.1 Install ImageMagick on Debian/Ubuntu

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

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

3.1.2 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
```

3.1.3 Install ImageMagick on Mac

You need one of Homebrew or MacPorts to install ImageMagick.

Homebrew

```
$ brew install imagemagick
```

MacPorts

```
$ sudo port install imagemagick
```

3.1.4 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`).

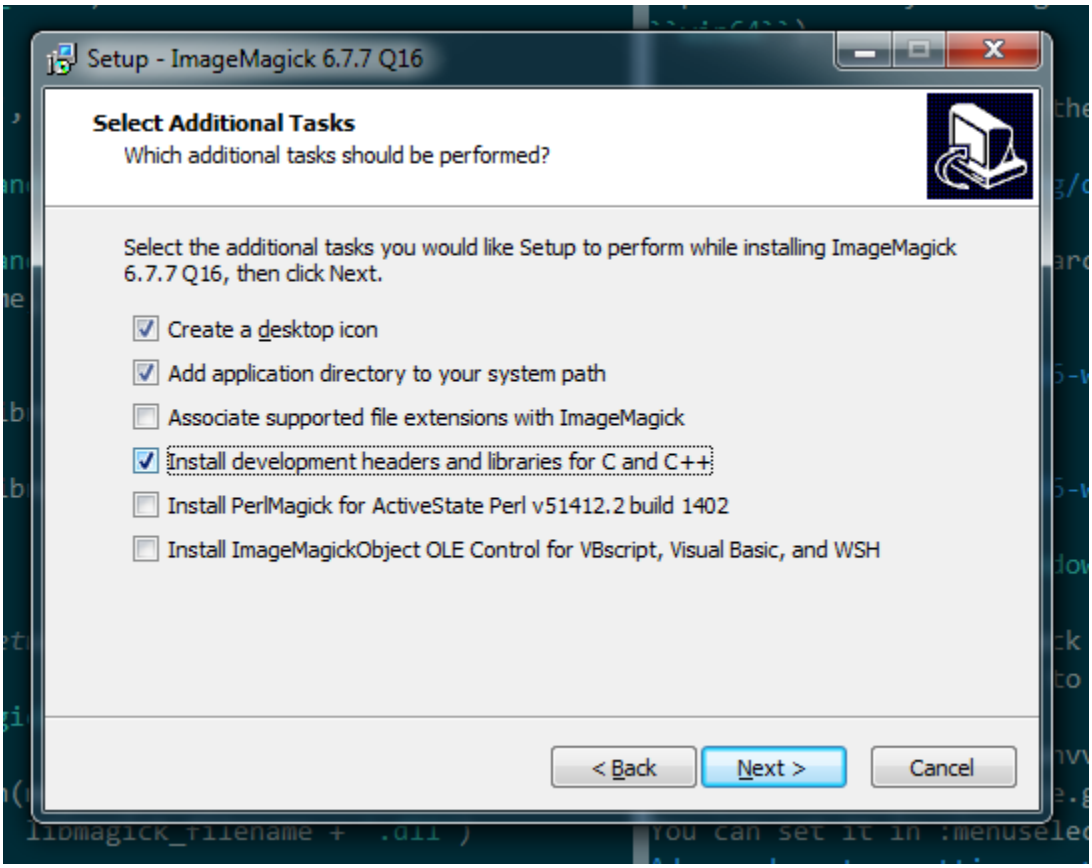
You can download it from the following link:

http://www.imagemagick.org/download/binaries/

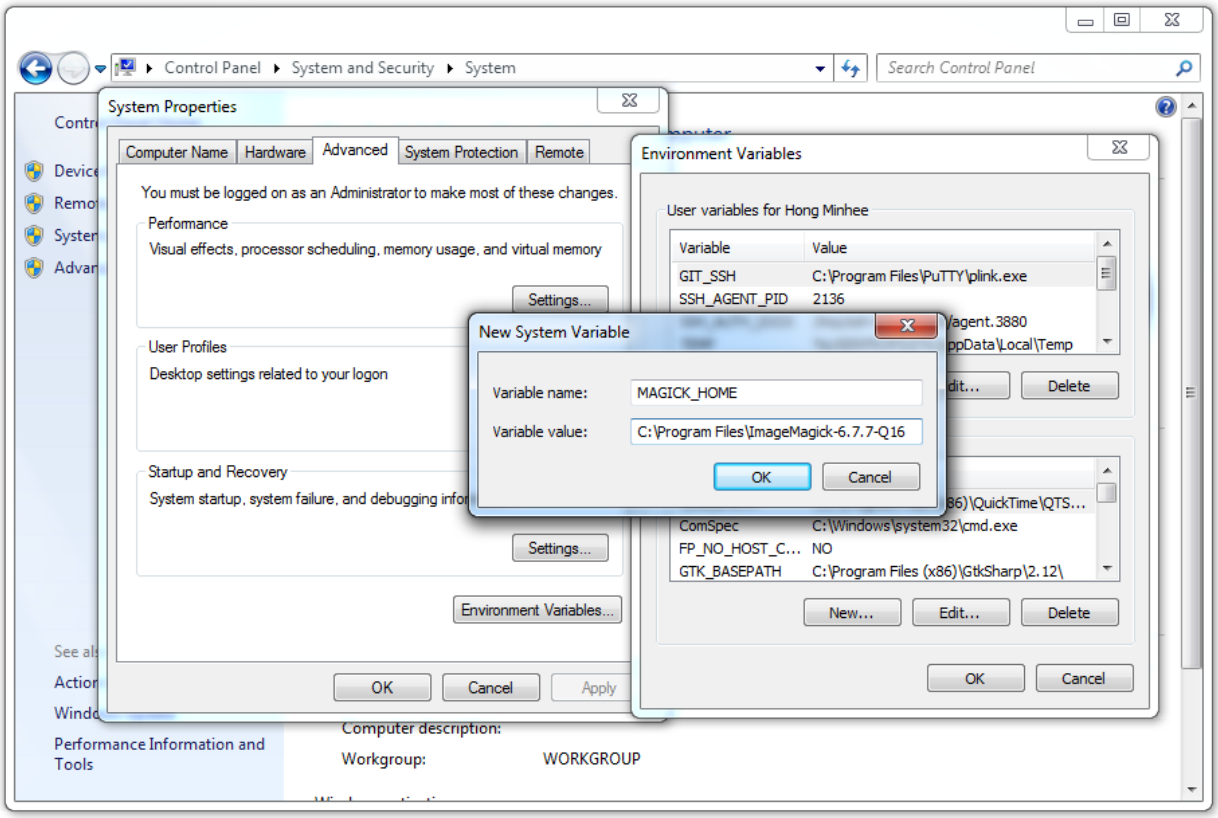
Choose a binary for your architecture:

Windows 32-bit ImageMagick-6.7.7-6-Q16-windows-dll.exe

Windows 64-bit ImageMagick-6.7.7-6-Q16-windows-x64-dll.exe



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



Lastly you have to set `MAGICK_HOME` environment variable to the path of ImageMagick (e.g. `C:\Program Files\ImageMagick-6.7.7-Q16`). You can set it in *Computer* → *Properties* → *Advanced system settings* → *Advanced* → *Environment Variables...*.

3.1.5 Install Wand on FreeBSD

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

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

3.2 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*

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

3.2.1 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 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).

3.2.2 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 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.

3.2.3 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 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.

3.2.4 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...
```

3.2.5 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 Writing images

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

3.3.1 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
```

3.3.2 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')
```

3.3.3 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()
```

3.3.4 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')
```

3.4 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 urllib2 import urlopen
>>> from wand.image import Image
>>> f = urlopen('http://api.twitter.com/1/users/profile_image/hongminhee')
>>> with Image(file=f) as img:
...     width = img.width
...     height = img.height
...
>>> f.close()
>>> width
48
>>> height
48
```

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

Note: These three properties are all readonly.

3.4.1 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)
```

3.4.2 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)
```

3.5 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.

3.6 Running tests

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

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

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

3.6.1 Skipping tests

There are some time-consuming tests. If `WANDTESTS_SKIP` environment variable it skips specified modules:

```
$ WANDTESTS_SKIP="color image" python setup.py test
```

Or you can test only specified modules using `WANDTESTS_ONLY` environment variable:

```
$ WANDTESTS_ONLY="color resource" python setup.py test
```

3.6.2 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 **easy_install** or **pip**:

```
$ easy_install tox
```

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

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

3.6.3 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:

http://travis-ci.org/dahlia/wand

3.7 Roadmap

3.7.1 Version 0.3

Python 3 compatibility Wand 0.3 will be the first version that supports Python 3.

The branch name for it will be python3.

Jython compatibility (#9) Wand 0.3 will support Jython 2.7+. Jython 2.7 is (June 2012) currently under alpha release, and Wand has been tested on it and fixed incompatible things.

It has been developed in the branch jython.

EXIF (#25) ImageMagick itself can read/write EXIF through `GetMagickInfo()` function. Wand 0.3 will make a binding for it.

Its branch name will be exif.

Image layers (#22) Wand 0.3 will be able to deal with layers of an image.

Its branch name will be layer.

3.7.2 Very future versions

Animations (#1) Wand will finally support animations like GIF and SWF in the future.

Its branch name will be animation.

3.8 Wand Changelog

3.8.1 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]

3.8.2 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 occured 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.

3.8.3 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]

3.8.4 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()`)

3.8.5 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`.

3.8.6 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]

3.8.7 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.

3.8.8 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.

3.8.9 Version 0.1.7

Released on November 10, 2011. Still alpha version.

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

3.8.10 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.

3.8.11 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.

3.8.12 Version 0.1.4

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

3.8.13 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.

3.8.14 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.

3.8.15 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).

3.8.16 Version 0.1.0

Released on October 1, 2011. Very alpha version.

CHAPTER

FOUR

REFERENCES

4.1 wand — Simple MagickWand API binding for Python

4.1.1 wand.image — Image objects

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

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

wand.image.**ALPHA_CHANNEL_TYPES = ('undefined', 'activate', 'background', 'copy', 'deactivate', 'extract', 'opaque', 'res**
(`tuple`) The list of alpha channel types

- `'undefined'`
- `'activate'`
- `'background'`
- `'copy'`
- `'deactivate'`
- `'extract'`
- `'opaque'`
- `'reset'`
- `'set'`
- `'shape'`
- `'transparent'`
- `'flatten'`
- `'remove'`

See Also:

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

wand.image.**CHANNELS = {'opacity': 8, 'true_alpha': 64, 'gray': 1, 'rgb_channels': 128, 'yellow': 4, 'sync_channels': 256, 'd**
(`dict`) The dictionary of channel types.

- 'undefined'
- 'red'
- 'gray'
- 'cyan'
- 'green'
- 'magenta'
- 'blue'
- 'yellow'
- 'alpha'
- 'opacity'
- 'black'
- 'index'
- 'composite_channels'
- 'all_channels'
- 'true_alpha'
- 'rgb_channels'
- 'gray_channels'
- 'sync_channels'
- 'default_channels'

See Also:

ImageMagick Color Channels Lists the various channel types with descriptions of each

wand.image.**COMPOSITE_OPS = ('undefined', 'no', 'add', 'atop', 'blend', 'bumpmap', 'change_mask', 'clear', 'color_burn',**
(tuple) The list of composition operators

- 'undefined'
- 'no'
- 'add'
- 'atop'
- 'blend'
- 'bumpmap'
- 'change_mask'
- 'clear'
- 'color_burn'
- 'color_dodge'
- 'colorize'
- 'copy_black'
- 'copy_blue'

- 'copy'
- 'copy_cyan'
- 'copy_green'
- 'copy_magenta'
- 'copy_opacity'
- 'copy_red'
- 'copy_yellow'
- 'darken'
- 'dst_atop'
- 'dst'
- 'dst_in'
- 'dst_out'
- 'dst_over'
- 'difference'
- 'displace'
- 'dissolve'
- 'exclusion'
- 'hard_light'
- 'hue'
- 'in'
- 'lighten'
- 'linear_light'
- 'luminize'
- 'minus'
- 'modulate'
- 'multiply'
- 'out'
- 'over'
- 'overlay'
- 'plus'
- 'replace'
- 'saturate'
- 'screen'
- 'soft_light'
- 'src_atop'
- 'src'

- 'src_in'
- 'src_out'
- 'src_over'
- 'subtract'
- 'threshold'
- 'xor'
- 'divide'

See Also:

ImageMagick Composition Operators Demonstrates the results of applying the various composition composition operators.

wand.image.**EVALUATE_OPS = ('undefined', 'add', 'and', 'divide', 'leftshift', 'max', 'min', 'multiply', 'or', 'rightshift', 'set',**
(tuple) The list of evaluation operators

- 'undefined'
- 'add'
- 'and'
- 'divide'
- 'leftshift'
- 'max'
- 'min'
- 'multiply'
- 'or'
- 'rightshift'
- 'set'
- 'subtract'
- 'xor'
- 'pow'
- 'log'
- 'threshold'
- 'thresholdblack'
- 'thresholdwhite'
- 'gaussiannoise'
- 'impulsenoise'
- 'laplaciannoise'
- 'multiplicativenoise'
- 'poissonnoise'
- 'uniformnoise'

- 'cosine'
- 'sine'
- 'addmodulus'
- 'mean'
- 'abs'
- 'exponential'
- 'median'
- 'sum'

See Also:

ImageMagick Image Evaluation Operators Describes the MagickEvaluateImageChannel method and lists the various evaluations operators

wand.image.**FILTER_TYPES = ('undefined', 'point', 'box', 'triangle', 'hermite', 'hanning', 'hamming', 'blackman', 'gaussia**
(tuple) The list of filter types.

- 'undefined'
- 'point'
- 'box'
- 'triangle'
- 'hermite'
- 'hanning'
- 'hamming'
- 'blackman'
- 'gaussian'
- 'quadratic'
- 'cubic'
- 'catrom'
- 'mitchell'
- 'jinc'
- 'sinc'
- 'sincfast'
- 'kaiser'
- 'welsh'
- 'parzen'
- 'bohman'
- 'bartlett'
- 'lagrange'
- 'lanczos'

- 'lanczossharp'
- 'lanczos2'
- 'lanczos2sharp'
- 'robidoux'
- 'robidouxsharp'
- 'cosine'
- 'spline'
- 'sentinel'

See Also:

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

wand.image.**IMAGE_TYPES = ('undefined', 'bilevel', 'grayscale', 'grayscalematte', 'palette', 'palettematte', 'truecolor', 'truec**
(tuple) The list of image types

- 'undefined'
- 'bilevel'
- 'grayscale'
- 'grayscalematte'
- 'palette'
- 'palettematte'
- 'truecolor'
- 'truecolormatte'
- 'colorseparation'
- 'colorseparationmatte'
- 'optimize'
- 'palettebilevelmatte'

See Also:

ImageMagick Image Types Describes the MagickSetImageType method which can be used to set the type of an image

wand.image.**UNIT_TYPES = ('undefined', 'pixelsperinch', 'pixelspercentimeter')**
(tuple) The list of resolution unit types

- 'undefined'
- 'pixelsperinch'
- 'pixelspercentimeter'

See Also:

ImageMagick Image Units Describes the MagickSetImageUnits method which can be used to set image units of resolution

exception wand.image.**ClosedImageError**
An error that rises when some code tries access to an already closed image.

class wand.image.**Image**(*image=None*, *blob=None*, *file=None*, *filename=None*, *format=None*, *width=None*, *height=None*, *background=None*)
An image object.

Parameters

- **image** (Image) – makes an exact copy of the image
- **blob** (str) – opens an image of the blob byte array
- **file** (*file object*) – opens an image of the file object
- **filename** (basestring) – opens an image of the filename string
- **format** (basestring) – forces filename to buffer.‘‘format‘‘ to help imagemagick detect the file format. Used only in blob or file cases
- **width** (numbers.Integral) – the width of new blank image.
- **height** (numbers.Integral) – the height of new blank imgage.
- **background** (wand.color.Color) – an optional background color. default is transparent

New in version 0.1.5: The file parameter.New in version 0.1.1: The blob parameter.New in version 0.2.1: The format parameter.New in version 0.2.2: The width, height, background parameters.

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

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

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

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

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

```
img[-70:-50, -20:-10]
#--> equivalent to img[width-70:width-50, height-20:height-10]
```

Returns the cropped image

Rtype Image

New in version 0.1.2.

alpha_channel
(bool) Get state of image alpha channel. It can also be used to enable/disable alpha channel. New in version 0.2.1.

background_color
(wand.color.Color) The image background color. It can also be set to change the background color. New in version 0.1.9.

clone()
Clones the image. It is equivalent to call Image with image parameter.

```
with img.clone() as cloned:
    # manipulate the cloned image
    pass
```

Returns the cloned new image

Return type Image

New in version 0.1.1.

close()

Closes the image explicitly. If you use the image object in `with` statement, it was called implicitly so don't have to call it.

Note: It has the same functionality of `destroy()` method.

---

composite(*image*, *left*, *top*)

Places the supplied `image` over the current image, with the top left corner of `image` at coordinates `left`, `top` of the current image. The dimensions of the current image are not changed.

Parameters

- **image** (`wand.image.Image`) – the image placed over the current image
- **left** (`numbers.Integral`) – the x-coordinate where *image* will be placed
- **top** (`numbers.Integral`) – the y-coordinate where *image* will be placed

New in version 0.2.0.

compression_quality

(`numbers.Integral`) Compression quality of this image. New in version 0.2.0.

convert(*format*)

Converts the image format with the original image maintained. It returns a converted image instance which is new.

```
with img.convert('png') as converted:
    converted.save(filename='converted.png')
```

Parameters **format** (`basestring`) – image format to convert to

Returns a converted image

Return type Image

Raises `ValueError` when the given `format` is unsupported

New in version 0.1.6.

crop(*left=0*, *top=0*, *right=None*, *bottom=None*, *width=None*, *height=None*, *reset_coords=True*)

Crops the image in-place.

```
+-------------------------------------------------+
|               ^                         ^       |
|               |                         |       |
|              top                        |       |
|               |                         |       |
|               v                         |       |
| <-- left --> +-------------------+  bottom       |
```

```
|               |              ^      |      |          |
|               | <-- width --|--->  |      |          |
|               |            height  |      |          |
|               |              |     |      |          |
|               |              v     |      |          |
|               +--------------------+      v          |
| <---------------- right ---------->                   |
+-------------------------------------------------------+
```

Parameters

- **left** (`numbers.Integral`) – x-offset of the cropped image. default is 0
- **top** (`numbers.Integral`) – y-offset of the cropped image. default is 0
- **right** (`numbers.Integral`) – second x-offset of the cropped image. default is the `width` of the image. this parameter and `width` parameter are exclusive each other
- **bottom** (`numbers.Integral`) – second y-offset of the cropped image. default is the `height` of the image. this parameter and `height` parameter are exclusive each other
- **width** (`numbers.Integral`) – the `width` of the cropped image. default is the `width` of the image. this parameter and `right` parameter are exclusive each other
- **height** (`numbers.Integral`) – the `height` of the cropped image. default is the `height` of the image. this parameter and `bottom` parameter are exclusive each other
- **reset_coords** (`bool`) – optional flag. If set, after the rotation, the coordinate frame will be relocated to the upper-left corner of the new image. By default is *True*.

Raises **exceptions.ValueError** when one or more arguments are invalid

---

Note: If you want to crop the image but not in-place, use slicing operator.

Changed in version 0.1.8: Made to raise `ValueError` instead of `IndexError` for invalid `width`/`height` arguments.New in version 0.1.7.

depth

(`numbers.Integral`) The depth of this image. New in version 0.2.1.

format

(`basestring`) The image format.

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

```
assert isinstance(img, wand.image.Image)
img.format = 'png'
```

It may raise `ValueError` when the format is unsupported.

See Also:

ImageMagick Image Formats ImageMagick uses an ASCII string known as *magick* (e.g. `GIF`) to identify file formats, algorithms acting as formats, built-in patterns, and embedded profile types.

New in version 0.1.6.

height

(`numbers.Integral`) The height of this image.

make_blob(*format=None*)
Makes the binary string of the image.

Parameters **format** (basestring) – the image format to write e.g. 'png', 'jpeg'. it is omittable

Returns a blob (bytes) string

Return type str

Raises ValueError when format is invalid

Changed in version 0.1.6: Removed a side effect that changes the image format silently.New in version 0.1.5: The format parameter became optional.New in version 0.1.1.

mimetype
(basestring) The MIME type of the image e.g. 'image/jpeg', 'image/png'. New in version 0.1.7.

quantum_range
(int) The maxumim value of a color channel that is supported by the imagemgick library. New in version 0.2.0.

reset_coords()
Reset the coordinate frame of the image so to the upper-left corner is (0, 0) again (crop and rotate operations change it). New in version 0.2.0.

resize(*width=None*, *height=None*, *filter='undefined'*, *blur=1*)
Resizes the image.

Parameters

- **width** (numbers.Integral) – the width in the scaled image. default is the original width
- **height** (numbers.Integral) – the height in the scaled image. default is the original height
- **filter** (basestring, numbers.Integral) – a filter type to use for resizing. choose one in FILTER_TYPES. default is 'undefined' which means IM will try to guess best one to use
- **blur** (numbers.Real) – the blur factor where > 1 is blurry, < 1 is sharp. default is 1

Changed in version 0.2.1: The default value of filter has changed from 'triangle' to 'undefined' instead.Changed in version 0.1.8: The blur parameter changed to take numbers.Real instead of numbers.Rational.New in version 0.1.1.

rotate(*degree*, *background=None*, *reset_coords=True*)
Rotates the image right. It takes a background color for degree that isn't a multiple of 90.

Parameters

- **degree** (numbers.Real) – a degree to rotate. multiples of 360 affect nothing
- **background** (wand.color.Color) – an optional background color. default is transparent
- **reset_coords** (bool) – optional flag. If set, after the rotation, the coordinate frame will be relocated to the upper-left corner of the new image. By default is *True*.

New in version 0.2.0: The reset_coords parameter.New in version 0.1.8.

save(*file=None*, *filename=None*)
Saves the image into the file or filename. It takes only one argument at a time.

Parameters

- **file** (*file object*) – a file object to write to
- **filename** (`basename`) – a filename string to write to

New in version 0.1.5: The `file` parameter.New in version 0.1.1.

signature
(`str`) The SHA-256 message digest for the image pixel stream. New in version 0.1.9.

size
(`tuple`) The pair of (`width`, `height`).

strip()
Strips an image of all profiles and comments. New in version 0.2.0.

transform(*crop=''*, *resize=''*)
Transforms the image using `MagickTransformImage()`, which is a convenience function accepting geometry strings to perform cropping and resizing. Cropping is performed first, followed by resizing. Either or both arguments may be omitted or given an empty string, in which case the corresponding action will not be performed. Geometry specification strings are defined as follows:

A geometry string consists of a size followed by an optional offset. The size is specified by one of the options below, where **bold** terms are replaced with appropriate integer values:

scale% Height and width both scaled by specified percentage

scale-x%xscale-y% Height and width individually scaled by specified percentages. Only one % symbol is needed.

width Width given, height automagically selected to preserve aspect ratio.

xheight Height given, width automagically selected to preserve aspect ratio.

widthxheight Maximum values of width and height given; aspect ratio preserved.

widthxheight! Width and height emphatically given; original aspect ratio ignored.

widthxheight> Shrinks images with dimension(s) larger than the corresponding width and/or height dimension(s).

widthxheight< Enlarges images with dimensions smaller than the corresponding width and/or height dimension(s).

area@ Resize image to have the specified area in pixels. Aspect ratio is preserved.

The offset, which only applies to the cropping geometry string, is given by `{+-}`**x**`{+-}`**y**, that is, one plus or minus sign followed by an **x** offset, followed by another plus or minus sign, followed by a **y** offset. Offsets are in pixels from the upper left corner of the image. Negative offsets will cause the corresponding number of pixels to be removed from the right or bottom edge of the image, meaning the cropped size will be the computed size minus the absolute value of the offset.

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

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

This method is a fairly thing wrapper for the C API, and does not perform any additional checking of the parameters except insofar as verifying that they are of the correct type. Thus, like the C API function, the method is very permissive in terms of what it accepts for geometry strings; unrecognized strings and trailing characters will be ignored rather than raising an error.

Parameters

- **crop** (basestring) – A geometry string defining a subregion of the image to crop to
- **resize** (basestring) – A geometry string defining the final size of the image

---

Note: #104 reported that `transform()` leaks memory under version 0.3.0, so be careful when you use it on long-running process like web server.

See Also:

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

New in version 0.2.2.

transparentize(*transparency*)
Makes the image transparent by subtracting some percentage of the black color channel. The `transparency` parameter specifies the percentage.

Parameters **transparency** (numbers.Real) – the percentage fade that should be performed on the image, from 0.0 to 1.0

New in version 0.2.0.

trim()
Remove solid border from image. Uses top left pixel as a guide. New in version 0.2.1.

type
(basestring) The image type.

Defines image type as in wand.image.IMAGE_TYPES enumeration.

It may raise `ValueError` when the type is unknown. New in version 0.2.2.

units
(basestring) The resolution units of this image.

wand
Internal pointer to the MagickWand instance. It may raise `ClosedImageError` when the instance has destroyed already.

watermark(*image*, *transparency=0.0*, *left=0*, *top=0*)
Transparentized the supplied `image` and places it over the current image, with the top left corner of `image` at coordinates `left`, `top` of the current image. The dimensions of the current image are not changed.

Parameters

- **image** (wand.image.Image) – the image placed over the current image
- **transparency** (numbers.Real) – the percentage fade that should be performed on the image, from 0.0 to 1.0
- **left** (numbers.Integral) – the x-coordinate where *image* will be placed
- **top** (numbers.Integral) – the y-coordinate where *image* will be placed

New in version 0.2.0.

width
(numbers.Integral) The width of this image.

class wand.image.**Iterator**(*image=None*, *iterator=None*)
Row iterator for Image. It shouldn't be instantiated directly; instead, it can be acquired through Image instance:

```
assert isinstance(image, wand.image.Image)
iterator = iter(image)
```

It doesn't iterate every pixel, but rows. For example:

```
for row in image:
    for col in row:
        assert isinstance(col, wand.color.Color)
        print col
```

Every row is a collections.Sequence which consists of one or more wand.color.Color values.

Parameters **image** (Image) – the image to get an iterator

New in version 0.1.3.

clone()
Clones the same iterator.

4.1.2 wand.color — Colors

New in version 0.1.2.

class wand.color.**Color**(*string=None*, *raw=None*)
Color value.

Unlike any other objects in Wand, its resource management can be implicit when it used outside of with block. In these case, its resource are allocated for every operation which requires a resource and destroyed immediately. Of course it is inefficient when the operations are much, so to avoid it, you should use color objects inside of with block explicitly e.g.:

```
red_count = 0
with Color('#f00') as red:
    with Image(filename='image.png') as img:
        for row in img:
            for col in row:
                if col == red:
                    red_count += 1
```

Parameters **string** (basestring) – a color namel string e.g. 'rgb(255, 255, 255)', '#fff', 'white'. see ImageMagick Color Names doc also

See Also:

ImageMagick Color Names The color can then be given as a color name (there is a limited but large set of these; see below) or it can be given as a set of numbers (in decimal or hexadecimal), each corresponding to a channel in an RGB or RGBA color model. HSL, HSLA, HSB, HSBA, CMYK, or CMYKA color models may also be specified. These topics are briefly described in the sections below.

== (other)
Equality operator.

Param other a color another one

Type color Color

Returns `True` only if two images equal.

Rtype `bool`

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

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

static c_equals(*a*, *b*)
Raw level version of equality test function for two pixels.

Parameters

- **a** (`ctypes.c_void_p`) – a pointer to PixelWand to compare
- **b** (`ctypes.c_void_p`) – a pointer to PixelWand to compare

Returns `True` only if two pixels equal

Return type `bool`

Note: It's only for internal use. Don't use it directly. Use == operator of `Color` instead.

---

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

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

string
(`basestring`) The string representation of the color.

4.1.3 wand.resource — Global resource management

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

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

> **Warning:** Don't call this function directly. Use `increment_refcount()` and `decrement_refcount()` functions instead.

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

> **Warning:** Don't call this function directly. Use `increment_refcount()` and `decrement_refcount()` functions instead.

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

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

class wand.resource.**Resource**

Abstract base class for MagickWand object that requires resource management. Its all subclasses manage the resource semiautomatically and support `with` statement as well:

```
with Resource() as resource:
    # use the resource...
    pass
```

It doesn't implement constructor by itself, so subclasses should implement it. Every constructor should assign the pointer of its resource data into `resource` attribute inside of `with allocate()` context. For example:

```
class Pizza(Resource):
    '''My pizza yummy.'''

    def __init__(self):
        with self.allocate():
            self.resource = library.NewPizza()
```

New in version 0.1.2.

allocate(**args*, ***kwds*)

Allocates the memory for the resource explicitly. Its subclasses should assign the created resource into `resource` attribute inside of this context. For example:

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

c_clear_exception = NotImplemented

(`ctypes.CFUNCTYPE`) The `ctypes` function that clears an exception of the `resource`.

Note: It is an abstract attribute that has to be implemented in the subclass.

---

c_destroy_resource = NotImplemented

(`ctypes.CFUNCTYPE`) The `ctypes` function that destroys the `resource`.

Note: It is an abstract attribute that has to be implemented in the subclass.

---

c_get_exception = NotImplemented

(`ctypes.CFUNCTYPE`) The `ctypes` function that gets an exception from the `resource`.

Note: It is an abstract attribute that has to be implemented in the subclass.

---

c_is_resource = NotImplemented

(`ctypes.CFUNCTYPE`) The `ctypes` predicate function that returns whether the given pointer (that contains a resource data usuaully) is a valid resource.

Note: It is an abstract attribute that has to be implemented in the subclass.

---

destroy()

Cleans up the resource explicitly. If you use the resource in `with` statement, it was called implicitly so have not to call it.

get_exception()

Gets a current exception instance.

Returns a current exception. it can be `None` as well if any errors aren't occurred

Return type `wand.exceptions.WandException`

raise_exception(*stacklevel=1*)
Raises an exception or warning if it has occurred.

resource
Internal pointer to the resource instance. It may raise `DestroyedResourceError` when the resource has destroyed already.

exception `wand.resource.`**DestroyedResourceError**
An error that rises when some code tries access to an already destroyed resource.

4.1.4 `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.`**BlobError**
Bases: `wand.exceptions.WandError`, `exceptions.IOError`

A binary large object could not be allocated, read, or written.

exception `wand.exceptions.`**BlobFatalError**
Bases: `wand.exceptions.WandFatalError`, `exceptions.IOError`

A binary large object could not be allocated, read, or written.

exception `wand.exceptions.`**BlobWarning**
Bases: `wand.exceptions.WandWarning`, `exceptions.IOError`

A binary large object could not be allocated, read, or written.

`wand.exceptions.`**CODE_MAP = [(<class 'wand.exceptions.WandWarning'>, 'Warning'), (<class 'wand.exceptions.WandErr**
(`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.WandError`

Pixels could not be read or written to the pixel cache.

exception `wand.exceptions.`**CacheFatalError**
Bases: `wand.exceptions.WandFatalError`

Pixels could not be read or written to the pixel cache.

exception `wand.exceptions.`**CacheWarning**
Bases: `wand.exceptions.WandWarning`

Pixels could not be read or written to the pixel cache.

exception `wand.exceptions.`**CoderError**
Bases: `wand.exceptions.WandError`

There was a problem with an image coder.

exception wand.exceptions.**CoderFatalError**
Bases: wand.exceptions.WandFatalError

There was a problem with an image coder.

exception wand.exceptions.**CoderWarning**
Bases: wand.exceptions.WandWarning

There was a problem with an image coder.

exception wand.exceptions.**ConfigureError**
Bases: wand.exceptions.WandError

There was a problem getting a configuration file.

exception wand.exceptions.**ConfigureFatalError**
Bases: wand.exceptions.WandFatalError

There was a problem getting a configuration file.

exception wand.exceptions.**ConfigureWarning**
Bases: wand.exceptions.WandWarning

There was a problem getting a configuration file.

exception wand.exceptions.**CorruptImageError**
Bases: wand.exceptions.WandError, exceptions.ValueError

The image file may be corrupt.

exception wand.exceptions.**CorruptImageFatalError**
Bases: wand.exceptions.WandFatalError, exceptions.ValueError

The image file may be corrupt.

exception wand.exceptions.**CorruptImageWarning**
Bases: wand.exceptions.WandWarning, exceptions.ValueError

The image file may be corrupt.

wand.exceptions.**DOMAIN_MAP = [('ResourceLimit', 'A program resource is exhausted e.g. not enough memory.', (<type '**
(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.WandError

An ImageMagick delegate failed to complete.

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

An ImageMagick delegate failed to complete.

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

An ImageMagick delegate failed to complete.

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

A drawing operation failed.

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

A drawing operation failed.

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

A drawing operation failed.

exception wand.exceptions.**FileOpenError**
Bases: wand.exceptions.WandError, exceptions.IOError

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

exception wand.exceptions.**FileOpenFatalError**
Bases: wand.exceptions.WandFatalError, exceptions.IOError

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

exception wand.exceptions.**FileOpenWarning**
Bases: wand.exceptions.WandWarning, exceptions.IOError

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

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

The operation could not complete due to an incompatible image.

exception wand.exceptions.**ImageFatalError**
Bases: wand.exceptions.WandFatalError

The operation could not complete due to an incompatible image.

exception wand.exceptions.**ImageWarning**
Bases: wand.exceptions.WandWarning

The operation could not complete due to an incompatible image.

exception wand.exceptions.**MissingDelegateError**
Bases: wand.exceptions.WandError, exceptions.ImportError

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

exception wand.exceptions.**MissingDelegateFatalError**
Bases: wand.exceptions.WandFatalError, exceptions.ImportError

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

exception wand.exceptions.**MissingDelegateWarning**
Bases: wand.exceptions.WandWarning, exceptions.ImportError

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

exception wand.exceptions.**ModuleError**
Bases: wand.exceptions.WandError

There was a problem with an image module.

exception wand.exceptions.**ModuleFatalError**
Bases: wand.exceptions.WandFatalError

There was a problem with an image module.

exception wand.exceptions.**ModuleWarning**
Bases: wand.exceptions.WandWarning

There was a problem with an image module.

exception wand.exceptions.**MonitorError**
Bases: wand.exceptions.WandError

There was a problem activating the progress monitor.

exception wand.exceptions.**MonitorFatalError**
Bases: wand.exceptions.WandFatalError

There was a problem activating the progress monitor.

exception wand.exceptions.**MonitorWarning**
Bases: wand.exceptions.WandWarning

There was a problem activating the progress monitor.

exception wand.exceptions.**OptionError**
Bases: wand.exceptions.WandError

A command-line option was malformed.

exception wand.exceptions.**OptionFatalError**
Bases: wand.exceptions.WandFatalError

A command-line option was malformed.

exception wand.exceptions.**OptionWarning**
Bases: wand.exceptions.WandWarning

A command-line option was malformed.

exception wand.exceptions.**PolicyError**
Bases: wand.exceptions.WandError

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

exception wand.exceptions.**PolicyFatalError**
Bases: wand.exceptions.WandFatalError

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

exception wand.exceptions.**PolicyWarning**
Bases: wand.exceptions.WandWarning

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

exception wand.exceptions.**RandomError**
Bases: wand.exceptions.WandError

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

exception wand.exceptions.**RandomFatalError**
Bases: wand.exceptions.WandFatalError

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

exception wand.exceptions.**RandomWarning**
Bases: wand.exceptions.WandWarning

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

exception wand.exceptions.**RegistryError**
Bases: wand.exceptions.WandError

There was a problem getting or setting the registry.

exception wand.exceptions.**RegistryFatalError**
Bases: wand.exceptions.WandFatalError

There was a problem getting or setting the registry.

exception wand.exceptions.**RegistryWarning**
Bases: wand.exceptions.WandWarning

There was a problem getting or setting the registry.

exception wand.exceptions.**ResourceLimitError**
Bases: wand.exceptions.WandError, exceptions.MemoryError

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

exception wand.exceptions.**ResourceLimitFatalError**
Bases: wand.exceptions.WandFatalError, exceptions.MemoryError

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

exception wand.exceptions.**ResourceLimitWarning**
Bases: wand.exceptions.WandWarning, exceptions.MemoryError

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

exception wand.exceptions.**StreamError**
Bases: wand.exceptions.WandError, exceptions.IOError

There was a problem reading or writing from a stream.

exception wand.exceptions.**StreamFatalError**
Bases: wand.exceptions.WandFatalError, exceptions.IOError

There was a problem reading or writing from a stream.

exception wand.exceptions.**StreamWarning**
Bases: wand.exceptions.WandWarning, exceptions.IOError

There was a problem reading or writing from a stream.

wand.exceptions.**TYPE_MAP = {385: <class 'wand.exceptions.MonitorWarning'>, 770: <class 'wand.exceptions.WandFata**
(dict) The dictionary of (code, exc_type).

exception wand.exceptions.**TypeError**
Bases: wand.exceptions.WandError

A font is unavailable; a substitution may have occurred.

exception wand.exceptions.**TypeFatalError**
Bases: wand.exceptions.WandFatalError

A font is unavailable; a substitution may have occurred.

exception wand.exceptions.**TypeWarning**
Bases: wand.exceptions.WandWarning

A font is unavailable; a substitution may have occurred.

exception wand.exceptions.**WandError**
Bases: wand.exceptions.WandError

There was a problem specific to the MagickWand API.

exception wand.exceptions.**WandException**
Bases: exceptions.Exception

All Wand-related exceptions are derived from this class.

exception wand.exceptions.**WandFatalError**
Bases: wand.exceptions.WandFatalError

There was a problem specific to the MagickWand API.

exception wand.exceptions.**WandWarning**
Bases: wand.exceptions.WandWarning

There was a problem specific to the MagickWand API.

exception wand.exceptions.**XServerError**
Bases: wand.exceptions.WandError

An X resource is unavailable.

exception wand.exceptions.**XServerFatalError**
Bases: wand.exceptions.WandFatalError

An X resource is unavailable.

exception wand.exceptions.**XServerWarning**
Bases: wand.exceptions.WandWarning

An X resource is unavailable.

4.1.5 wand.api — Low-level interfaces

Changed in version 0.1.10: Changed to throw ImportError instead of AttributeError when the shared library fails to load.

wand.api.**load_library**()
Loads the MagickWand library.

Returns the MagickWand library and the ImageMagick library

Return type ctypes.CDLL

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

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

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

4.1.6 wand.display — Displaying images

The display() functions shows you the image. It is useful for debugging.

If you are in Mac, the image will be opened by your default image application (**Preview.app** usually).

If you are in Windows, the image will be opened by **imdisplay.exe**, or your default image application (**Windows Photo Viewer** usually) if **imdisplay.exe** is unavailable.

You can use it from CLI also. Execute wand.display module through *python -m* option:

```
$ python -m wand.display wandtests/assets/mona-lisa.jpg
```

New in version 0.1.9.

wand.display.**display**(*image*, *server_name=':0'*)
Displays the passed `image`.

Parameters

- **image** (`Image`) – an image to display
- **server_name** (`str`) – X11 server name to use. it is ignored and not used for Mac. default is `':0'`

4.1.7 `wand.version` — Version data

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

```
$ python -m wand.version
0.2.4
$ python -m wand.version --verbose
Wand 0.2.4
ImageMagick 6.7.7-6 2012-06-03 Q16 http://www.imagemagick.org
```

New in version 0.2.0: The command line interface.New in version 0.2.4: The `--verbose`/`-v` option which also prints ImageMagick library version for CLI.

wand.version.**VERSION = '0.2.4'**
(`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, 2, 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_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**
(`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_RELEASE_DATE_STRING = None**
(`datetime.date`) The release date of the linked ImageMagick library. The same to the result of `GetMagickReleaseDate()` function. New in version 0.2.1.

CHAPTER

FIVE

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.

CHAPTER

SIX

OPEN SOURCE

Wand is an open source software written by Hong Minhee (initially written for StyleShare). 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/dahlia/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

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

W