Piexif Documentation

Release 1.0.X

hMatoba

Jun 30, 2018

Contents

To simplify exif manipulations with python. Writing, reading, and more... Piexif is pure Python. To everywhere with Python.

CHAPTER 1

About Piexif

1.1 What for?

To simplify exif manipulations with python. Writing, reading, and more...

1.2 How to Use

There are only just five functions.

- *load(filename)* - Get exif data as *dict*.
- *dump(exif_dict)* - Get exif as *bytes* to save with JPEG.
- *insert(exif_bytes, filename)* - Insert exif into JPEG.
- *remove(filename)* - Remove exif from JPEG.
- *transplant(filename, filename)* - Transplant exif from JPEG to JPEG.

1.3 Dependency

Piexif doesn't depend on any third library.

1.4 Environment

Tested on Python 2.7, 3.3, 3.4, 3.5, pypy, and pypy3. Piexif would run even on IronPython. Piexif is OS independent and can run on GoogleAppEngine.

1.5 License

CHAPTER 2

Installation

Note: Piexif supports Python versions 2.7, 3.3, 3.4, Pypy, Pypy3

'easy_install':

```
$ easy_install piexif
```

or 'pip':

```
$ pip install piexif
```

or download .zip, extract it. Put 'piexif' directory into your environment.

CHAPTER 3

Functions

Warning: It could set any value in exif without actual value. For example, actual XResolution is 300, whereas XResolution value in exif is 0. Confliction might happen.

Warning: To edit exif tags and values appropriately, read official document from P167-. http://www.cipa.jp/std/documents/e/DC-008-2012_E.pdf

Note: This document is written for using Piexif on Python 3.x.

3.1 load

piexif.**load**(*filename*, *key_is_name=False*)

Return exif data as dict. Keys(IFD name), be contained, are "0th", "Exif", "GPS", "Interop", "1st", and "thumbnail". Without "thumbnail", the value is dict(tag/value). "thumbnail" value is JPEG as bytes.

Parameters **filename** (*str*) – JPEG or TIFF

Returns Exif data({"0th":dict, "Exif":dict, "GPS":dict, "Interop":dict, "1st":dict, "thumbnail":bytes})

Return type dict

```
exif_dict = piexif.load("foo.jpg")
thumbnail = exif_dict.pop("thumbnail")
if thumbnail is not None:
    with open("thumbnail.jpg", "wb+") as f:
        f.write(thumbnail)
```

(continues on next page)

(continued from previous page)

```
for ifd_name in exif_dict:
    print("\n{0} IFD:".format(ifd_name))
    for key in exif_dict[ifd_name]:
        try:
            print(key, exif_dict[ifd_name][key][:10])
        except:
            print(key, exif_dict[ifd_name][key])
```

piexif.**load**(*data*)

Return exif data as dict. The keys(IFD name), will be contained, are "0th", "Exif", "GPS", "Interop", "1st", and "thumbnail". If there is no data to return, the key won't be contained. Without "thumbnail", the value is dict(tag name/tag value). "thumbnail" value is JPEG as bytes.

Parameters **data** (*bytes*) – JPEG, TIFF, or Exif

Returns Exif data({"0th":dict, "Exif":dict, "GPS":dict, "Interop":dict, "1st":dict, "thumbnail":bytes})

Return type dict

3.2 dump

piexif.**dump**(*exif_dict*)

Return exif as bytes.

Parameters **exif_dict** (*dict*) – Exif data({"0th":0thIFD - dict, "Exif":ExifIFD - dict, "GPS":GPSIFD - dict, "Interop":InteroperabilityIFD - dict, "1st":1stIFD - dict, "thumbnail":JPEG data - bytes})

Returns Exif

Return type bytes

```
import io
from PIL import Image
import piexif

o = io.BytesIO()
thumb_im = Image.open("foo.jpg")
thumb_im.thumbnail((50, 50), Image.ANTIALIAS)
thumb_im.save(o, "jpeg")
thumbnail = o.getvalue()

zeroth_ifd = {piexif.ImageIFD.Make: u"Canon",
              piexif.ImageIFD.XResolution: (96, 1),
              piexif.ImageIFD.YResolution: (96, 1),
              piexif.ImageIFD.Software: u"piexif"
              }
exif_ifd = {piexif.ExifIFD.DateTimeOriginal: u"2099:09:29 10:10:10",
            piexif.ExifIFD.LensMake: u"LensMake",
            piexif.ExifIFD.Sharpness: 65535,
            piexif.ExifIFD.LensSpecification: ((1, 1), (1, 1), (1, 1), (1, 1)),
            }
gps_ifd = {piexif.GPSIFD.GPSVersionID: (2, 0, 0, 0),
           piexif.GPSIFD.GPSAltitudeRef: 1,
           piexif.GPSIFD.GPSDateStamp: u"1999:99:99 99:99:99",
```

(continues on next page)

(continued from previous page)

```
            }
first_ifd = {piexif.ImageIFD.Make: u"Canon",
             piexif.ImageIFD.XResolution: (40, 1),
             piexif.ImageIFD.YResolution: (40, 1),
             piexif.ImageIFD.Software: u"piexif"
             }

exif_dict = {"0th":zeroth_ifd, "Exif":exif_ifd, "GPS":gps_ifd, "1st":first_ifd,
↪"thumbnail":thumbnail}
exif_bytes = piexif.dump(exif_dict)
im = Image.open("foo.jpg")
im.thumbnail((100, 100), Image.ANTIALIAS)
im.save("out.jpg", exif=exif_bytes)
```

Properties of *piexif.ImageIFD* help to make 0thIFD dict and 1stIFD dict. *piexif.ExifIFD* is for ExifIFD dict. *piexif.GPSIFD* is for GPSIFD dict. *piexif.InteropIFD* is for InteroperabilityIFD dict.

Note: ExifTag(34665), GPSTag(34853), and InteroperabilityTag(40965) in 0thIFD automatically are set appropriate value.

Note: JPEGInterchangeFormat(513), and JPEGInterchangeFormatLength(514) in 1stIFD automatically are set appropriate value.

Note: If 'thumbnail' is contained in dict, '1st' must be contained – and vice versa. 1stIFD means thumbnail's information.

3.3 insert

piexif.**insert**(*exif_bytes*, *filename*)

Insert exif into JPEG.

Parameters

- **exif_bytes** (*bytes*) – Exif as bytes
- **filename** (*str*) – JPEG

```
exif_bytes = piexif.dump(exif_dict)
piexif.insert(exif_bytes, "foo.jpg")
```

piexif.**insert**(*exif_bytes*, *data*, *output*)

Insert exif into JPEG.

Parameters

- **exif_bytes** (*bytes*) – Exif as bytes
- **data** (*bytes*) – JPEG data
- **output** (*io.BytesIO*) – ouput data

3.4 remove

piexif.**remove**(*filename*)
Remove exif from JPEG.

Parameters **filename** (*str*) – JPEG

```
piexif.remove("foo.jpg")
```

piexif.**remove**(*data*, *output*)
Remove exif from JPEG.

Parameters

- **data** (*bytes*) – JPEG data
- **output** (*io.BytesIO*) – output data

3.5 transplant

piexif.**transplant**(*filename1*, *filename2*)
Transplant exif from filename1 to filename2.

Parameters

- **filename1** (*str*) – JPEG
- **filename2** (*str*) – JPEG

```
piexif.transplant("exif_src.jpg", "foo.jpg")
```

piexif.**transplant**(*exif_src*, *image_src*, *output*)
Transplant exif from exif_src to image_src.

Parameters

- **exif_src** (*bytes*) – JPEG data
- **image_src** (*bytes*) – JPEG data
- **output** (*io.BytesIO*) – output data

CHAPTER 4

Appendices

4.1 Exif Data in Piexif

Each exif tag has appropriate type of the value. BYTE, ASCII, SHORT, or... See the document of Exif. http://www.cipa.jp/std/documents/e/DC-008-2012_E.pdf

Exif Type	Python Type(3.x)
BYTE	int
ASCII	str
SHORT	int
LONG	int
RATIONAL	(int, int)
UNDEFINED	bytes
SRATIONAL	(int, int)

If value type is number(BYTE, SHORT, LONG, RATIONAL, or SRATIONAL) and value count is two or more number, it is expressed with tuple.

BYTE, SHORT, LONG	(int, int, ...)
RATIONAL, SRATIONAL	((int, int), (int, int), ...)

Note: If value type is number and value count is one, tuple that is length one value(e.g. (int,)) also be accepted.

Exif in piexif example is below.

```
zeroth_ifd = {piexif.ImageIFD.Make: "Canon",  # ASCII, count any
              piexif.ImageIFD.XResolution: (96, 1),  # RATIONAL, count 1
              piexif.ImageIFD.YResolution: (96, 1),  # RATIONAL, count 1
              piexif.ImageIFD.Software: "piexif"  # ASCII, count any
```

(continues on next page)

(continued from previous page)

```
              }
exif_ifd = {piexif.ExifIFD.ExifVersion: b"\x02\x00\x00\x00"  # UNDEFINED, count 4
            piexif.ExifIFD.LensMake: "LensMake",  # ASCII, count any
            piexif.ExifIFD.Sharpness: 65535,  # SHORT, count 1 ... also be accepted
↪'(65535,)'
            piexif.ExifIFD.LensSpecification: ((1, 1), (1, 1), (1, 1), (1, 1)),  #␣
↪Rational, count 4
            }
gps_ifd = {piexif.GPSIFD.GPSVersionID: (2, 0, 0, 0),  # BYTE, count 4
           piexif.GPSIFD.GPSAltitudeRef: 1,  # BYTE, count 1 ... also be accepted '(1,
↪)'
           }
exif_dict = {"0th":zeroth_ifd, "Exif":exif_ifd, "GPS":gps_ifd}
exif_bytes = piexif.dump(exif_dict)

# round trip
piexif.insert(exif_bytes, "foo.jpg")
exif_dict_tripped = piexif.load("foo.jpg")
```

4.2 On GoogleAppEngine

On GoogleAppEngine, it can't save files on disk. Therefore files must be handled on memory.

```
jpg_data = self.request.get("jpeg")
output = io.BytesIO()

# load
exif = piexif.load(jpg_data)

# insert
piexif.insert(exif_bytes, jpg_data, output)

# remove
piexif.remove(jpg_data, output)

# transplant
piexif.transplant(jpg_data1, jpg_data2, output)
```

4.3 Invalid EXIF Thumbnails

EXIF data will sometimes be either corrupted or written by non-compliant software. When this happens, it's possible that the thumbnail stored in EXIF cannot be found when attempting to dump the EXIF dictionary.

A good solution would be to remove the thumbnail from the EXIF dictionary and then re-attempt the dump:

```
try:
    exif_bytes = piexif.dump(exif_dict)
except InvalidImageDataError:
    del exif_dict["1st"]
    del exif_dict["thumbnail"]
    exif_bytes = piexif.dump(exif_dict)
```

CHAPTER 5

Samples

5.1 With PIL(Pillow)

```
from PIL import Image
import piexif

im = Image.open(filename)
exif_dict = piexif.load(im.info["exif"])
# process im and exif_dict...
w, h = im.size
exif_dict["0th"][piexif.ImageIFD.XResolution] = (w, 1)
exif_dict["0th"][piexif.ImageIFD.YResolution] = (h, 1)
exif_bytes = piexif.dump(exif_dict)
im.save(new_file, "jpeg", exif=exif_bytes)
```

5.2 Check Containing Tag

```
from PIL import Image
import piexif

exif_dict = piexif.load(filename)
if piexif.ImageIFD.Orientation in exif_dict["0th"]:
    print("Orientation is ", exif_dict["0th"][piexif.ImageIFD.Orientation])
if piexif.ExifIFD.Gamma in exif_dict["Exif"]:
    print("Gamma is ", exif_dict["Exif"][piexif.ExifIFD.Gamma])
```

5.3 Rotate Image by Exif Orientation

Rotate image by exif orientation tag and remove orientation tag.

```
from PIL import Image
import piexif

def rotate_jpeg(filename):
    img = Image.open(filename)
    if "exif" in img.info:
        exif_dict = piexif.load(img.info["exif"])

        if piexif.ImageIFD.Orientation in exif_dict["0th"]:
            orientation = exif_dict["0th"].pop(piexif.ImageIFD.Orientation)
            exif_bytes = piexif.dump(exif_dict)

            if orientation == 2:
                img = img.transpose(Image.FLIP_LEFT_RIGHT)
            elif orientation == 3:
                img = img.rotate(180)
            elif orientation == 4:
                img = img.rotate(180).transpose(Image.FLIP_LEFT_RIGHT)
            elif orientation == 5:
                img = img.rotate(-90).transpose(Image.FLIP_LEFT_RIGHT)
            elif orientation == 6:
                img = img.rotate(-90)
            elif orientation == 7:
                img = img.rotate(90).transpose(Image.FLIP_LEFT_RIGHT)
            elif orientation == 8:
                img = img.rotate(90)

            img.save(filename, exif=exif_bytes)
```

5.4 Piexif on Server

Piexif loads exif data as dict from JPEG. Python dict is easy to convert to JSON, therefore piexif has a good compatible with AJAX, document oriented DB...

```
"""GoogleAppEngine and Python 2.7"""
import json

import tornado.web
import tornado.wsgi
import piexif


class PostHandler(tornado.web.RequestHandler):
    def post(self):
        jpg_data = self.request.body
        try:
            exif_dict = piexif.load(jpg_data)
        except:
            self.set_status(400)
            return self.write("Wrong jpeg")
        self.add_header("Content-Type", "application/json")
        thumbnail = exif_dict.pop("thumbnail")
        data_d = {}
        for ifd in exif_dict:
            data_d[ifd] = {piexif.TAGS[ifd][tag]["name"]:exif_dict[ifd][tag]
```

(continues on next page)

(continued from previous page)

```
                            for tag in exif_dict[ifd]}
        data_d["thumbnail"] = thumbnail
        data = json.dumps(data_d, encoding="latin1")
        return self.write(data)

application = tornado.web.Application([
    (r"/p", PostHandler),
])

application = tornado.wsgi.WSGIAdapter(application)
```

CHAPTER 6

Changelog

6.1 1.0.12

- Added explicit InvalidImageDataError exception to aid users. Related to https://github.com/hMatoba/Piexif/issues/30.
- Fixed minor issue with tests.
- Removed minor amounts of unused logic.
- Updated .travis.yml for Python and Pillow versions.

6.2 1.0.11

- Add option argument to “load”.

6.3 1.0.10

- Add tags in Exif ver.2.31

6.4 1.0.9

- Performance up “load” jpeg from file.

6.5 1.0.8

- Exclude checking extension in “load”.

6.6 1.0.7

- Fix packaging.

6.7 1.0.6

- Refactoring.

6.8 1.0.5

- Bug fix: https://github.com/hMatoba/Piexif/issues/16

6.9 1.0.4

- Fix APP1 matter.

6.10 1.0.3

- Support SLong type.

6.11 1.0.2

- Add some error detail to 'dump'.

6.12 1.0.1

- Fix bug. 'load' and 'dump' InteroperabilityIFD was wrong.

6.13 1.0.0

- Add handling InteroperabilityIFD, 1stIFD, and thumbnail image.
- *'load'* returns a dict that contains "0th", "Exif", "GPS", "Interop", "1st", and "thumbnail" keys.
- *'dump'* argument is changed from three dicts to a dict.
- *piexif.ZerothIFD* is renamed *piexif.ImageIFD* for 1stIFD support.

6.14 0.7.0c

- Rename project.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

P