Piexif Documentation

Release 1.1.x

hMatoba

Nov 15, 2018

Contents

Piexif simplifies interacting with EXIF data in Python. It includes the tools necessary for extracting, creating, manipulating, converting and writing EXIF data to JPEG, WebP and TIFF files.

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 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, 3.6, 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: Any value can be set as an exif value without it matching the actual value. For example, the XResolution value of 0 can be written while the actual XResolution is 300. This can cause conflicts.

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*)

Returns exif data as a dictionary with the following keys: "0th", "Exif", "GPS", "Interop", "1st", and "thumbnail". All values are dictionaries except for "thumbnail" which has the value of either a JPEG as bytes or None if no thumbnail is stored in the exif data.

Parameters **filename** (*str*) – JPEG, WebP, 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)
```

```
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*)

Returns exif data as a dictionary with the following keys unless its value does not exist in the file: "0th", "Exif", "GPS", "Interop", "1st", and "thumbnail". All values are dictionaries except for "thumbnail" which has the value of either a JPEG as bytes or None if no thumbnail is stored in the exif data.

Parameters **data** (*bytes*) – JPEG, WebP, 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*)

Returns exif data 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",
           }
```

```
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)
```

The 0thIFD and 1stIFD dictionaries should be constructed using the properties of *piexif.ImageIFD*. Use the properties of *piexif.ExifIFD* for the ExifIFD dictionary, *piexif.GPSIFD* for the GPSIFD dictionary, and *piexif.InteropIFD* for the InteroperabilityIFD dictionary.

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

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

Note: If the value of key 'thumbnail' is a dictionary, then the value for key '1st' must also be a dictionary and vice versa. This is because 1stIFD stores the thumbnail's information.

3.3 insert

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

Inserts exif into JPEG or WebP.

Parameters

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

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

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

Inserts exif into JPEG or WebP.

Parameters

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

3.4 remove

piexif.**remove**(*filename*)
Removes exif data from JPEG or WebP.

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

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

piexif.**remove**(*data*, *output*)
Removes exif data from JPEG or WebP.

Parameters

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

3.5 transplant

piexif.**transplant**(*filename1*, *filename2*)
Copies exif data 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

Helper Functions

4.1 UserComment

piexif.helper.UserComment.**load**(*data*)

Convert “UserComment” value in exif format to str.

Parameters **data** (*bytes*) – “UserComment” value from exif

Returns u”foobar”

Return type str(Unicode)

```
import piexif
import piexif.helper
exif_dict = piexif.load("foo.jpg")
user_comment = piexif.helper.UserComment.load(exif_dict["Exif"][piexif.ExifIFD.
↪UserComment])
```

piexif.helper.UserComment.**dump**(*data*, *encoding="ascii"*)

Convert str to appropriate format for “UserComment”.

Parameters

- **data** – Like u”foobar”
- **encoding** (*str*) – “ascii”, “jis”, or “unicode”

Returns b”ASCIIx00x00x00foobar”

Return type bytes

```
import piexif
import piexif.helper
user_comment = piexif.helper.UserComment.dump(u"Edit now.")
exif_dict = piexif.load("foo.jpg")
exif_dict["Exif"][piexif.ExifIFD.UserComment] = user_comment
exif_bytes = piexif.dump(exif_dict)
```

CHAPTER 5

Appendices

5.1 Exif Data in Piexif

Each exif tag has an appropriate data type (BYTE, ASCII, SHORT, etc.). Please see the official Exif standards for full documentation: http://www.cipa.jp/std/documents/e/DC-008-2012_E.pdf

Exif Type	**Python Type(3.x)**
BYTE	int
SIGNED BYTE	int
ASCII	str
SHORT	int
SIGNED SHORT	int
LONG	int
RATIONAL	(int, int)
UNDEFINED	bytes
SRATIONAL	(int, int)
FLOAT	float
DOUBLE	float

Values that are numerical (BYTE, SHORT, LONG, RATIONAL, or SRATIONAL) and that require two or more values should be expressed with a tuple.

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

Note: If the value type is numerical but only one value is required, a tuple of length 1 (e.g. (int,)) is also acceptable.

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
              }
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")
```

5.2 On GoogleAppEngine

Files cannot be saved to disk when using GoogleAppEngine. Therefore, files must be handled in 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)
```

5.3 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 6

Samples

6.1 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)
```

6.2 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])
```

6.3 Rotate Image by Exif Orientation

Example) rotate the image by its exif orientation tag value and remove the orientation tag from the image's exif data:

```
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, expand=True).transpose(Image.FLIP_LEFT_RIGHT)
            elif orientation == 6:
                img = img.rotate(-90, expand=True)
            elif orientation == 7:
                img = img.rotate(90, expand=True).transpose(Image.FLIP_LEFT_RIGHT)
            elif orientation == 8:
                img = img.rotate(90, expand=True)

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

6.4 Piexif on Server

Piexif stores the image's exif data as a dictionary. This dictionary is easy to convert to JSON for use with AJAX or for use with document oriented databases.

```
"""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]
```

```
                            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 7

Changelog

7.1 1.1.2

- Resolve issue. https://github.com/hMatoba/Piexif/issues/64

7.2 1.1.1

- Ignore XMP segment. Related to https://github.com/hMatoba/Piexif/pull/74.

7.3 1.1.0b

- “load”, “insert”, and “remove” support WebP format.

7.4 1.0.13

- Added helper function to read and write “UserComment”.
- Added to support for SignedByte, SigendShort, Float, and Double.

7.5 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.

7.6 1.0.11

- Add option argument to “load”.

7.7 1.0.10

- Add tags in Exif ver.2.31

7.8 1.0.9

- Performance up “load” jpeg from file.

7.9 1.0.8

- Exclude checking extension in “load”.

7.10 1.0.7

- Fix packaging.

7.11 1.0.6

- Refactoring.

7.12 1.0.5

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

7.13 1.0.4

- Fix APP1 matter.

7.14 1.0.3

- Support SLong type.

7.15 1.0.2

- Add some error detail to 'dump'.

7.16 1.0.1

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

7.17 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.

7.18 0.7.0c

- Rename project.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

P