exif Documentation

Release 1.3.5

Tyler N. Thieding

Apr 17, 2022

Contents

Read and modify image EXIF metadata using Python without any third-party software dependencies. For example, batch process image metadata using a Python script.

Note: I developed this package in 2018 as a hobby; however, I no longer have the same bandwidth to work on this project. As always, contributions and bug fixes are welcome and appreciated. If this package does not suit your needs in its current form, I encourage you to investigate alternative packages such as piexif, Pillow, or the like.

CHAPTER 1

Quick Start

Open an image with EXIF metadata using the Python `open()` built-in function. Ensure the binary mode flag is set. Pass this image file object into the `exif.Image` class:

```
>>> from exif import Image
>>> with open('grand_canyon.jpg', 'rb') as image_file:
...     my_image = Image(image_file)
...
>>> my_image.has_exif
True
```

List EXIF attributes using the `list_all()` method:

```
>>> my_image.list_all()
['_exif_ifd_pointer', '_gps_ifd_pointer', 'aperture_value', 'brightness_value',
↪'color_space',
 'components_configuration', 'compression', 'datetime', 'datetime_digitized',
↪'datetime_original', 'exif_version',
 'exposure_bias_value', 'exposure_mode', 'exposure_program', 'exposure_time', 'f_
↪number', 'flash',
 'flashpix_version', 'focal_length', 'focal_length_in_35mm_film', 'gps_altitude',
↪'gps_altitude_ref',
 'gps_datestamp', 'gps_dest_bearing', 'gps_dest_bearing_ref', 'gps_horizontal_
↪positioning_error',
 'gps_img_direction', 'gps_img_direction_ref', 'gps_latitude', 'gps_latitude_ref',
↪'gps_longitude',
 'gps_longitude_ref', 'gps_speed', 'gps_speed_ref', 'gps_timestamp', 'jpeg_
↪interchange_format',
 'jpeg_interchange_format_length', 'lens_make', 'lens_model', 'lens_specification',
↪'make', 'maker_note',
 'metering_mode', 'model', 'orientation', 'photographic_sensitivity', 'pixel_x_
↪dimension', 'pixel_y_dimension',
 'resolution_unit', 'scene_capture_type', 'scene_type', 'sensing_method', 'shutter_
↪speed_value', 'software',
 'subject_area', 'subsec_time_digitized', 'subsec_time_original', 'white_balance', 'x_
↪resolution',
```

(continues on next page)

(continued from previous page)

```
 'y_and_c_positioning', 'y_resolution']
```

Access EXIF metadata tags using Python attribute notation:

```
>>> # Read tags with Python "get" notation.
>>> my_image.gps_latitude
(36.0, 3.0, 11.08)
>>> my_image.gps_longitude
(112.0, 5.0, 4.18)
>>> my_image.model
'iPhone 7'
>>>
>>> # Modify tags with Python "set" notation.
>>> my_image.make = "Python"
>>>
>>> # Delete tags with Python "del" notation.
>>> del my_image.gps_latitude
>>> del my_image.gps_longitude
>>>
>>> # Add new tags with Python "set" notation.
>>> from exif import LightSource
>>> my_image.light_source = LightSource.DAYLIGHT
```

Write the image with modified EXIF metadata to an image file using `open()` in binary write mode:

```
>>> with open('modified_image.jpg', 'wb') as new_image_file:
...     new_image_file.write(my_image.get_file())
...
```

Refer to the usage page for information and examples of alternative ways to access EXIF tags (e.g. with index/item syntax or with methods).

1.1 About

1.1.1 Contributors

- Tyler N. Thieding (Primary Author)
- ArgiesDario (`delete_all()` Method)
- Justin Saunders (Support Signed Short Integers)
- RKrahl (`setup.py` Tweaks)
- chbndrhnns (Allow Read File in Instantiation)
- Rune Monzel (Example Code for Use with NumPy and OpenCV)
- Alex Mykyta (Fix Value of `SceneCaptureType.NIGHT_SCENE`)

1.1.2 Development

Repository https://www.gitlab.com/TNThieding/exif

1.1.3 License

```
Copyright 2021 Tyler N. Thieding

Permission is hereby granted, free of charge, to any person obtaining a copy of this
software and associated documentation files (the "Software"), to deal in the Software
without restriction, including without limitation the rights to use, copy, modify,␣
↪merge,
publish, distribute, sublicense, and/or sell copies of the Software, and to permit␣
↪persons
to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies␣
↪or
substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A␣
↪PARTICULAR
PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE␣
↪LIABLE
FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

1.2 API Reference

Read and modify image EXIF metadata using Python.

1.2.1 Classes

Image

class exif.**Image**(*img_file: Union[BinaryIO, bytes, str]*)

Image EXIF metadata interface class.

Parameters **img_file** – image file with EXIF metadata

delete(*attribute: str*) → None

Remove the specified attribute from the image.

Parameters **attribute** – image EXIF attribute name

delete_all() → None

Remove all EXIF tags from the image.

get(*attribute: str*, *default: Any = None*) → Any

Return the value of the specified tag.

If the attribute is not available or set, return the value specified by the `default` keyword argument.

Parameters

- **attribute** – image EXIF attribute name
- **default** – return value if attribute does not exist

Returns tag value if present, `default` otherwise

Return type corresponding Python type

get_all() → Dict[str, Any]

Return dictionary containing all EXIF tag values keyed by tag name.

get_file() → bytes

Generate equivalent binary file contents.

Returns image binary with EXIF metadata

get_thumbnail() → bytes

Extract thumbnail binary contained in EXIF metadata.

Returns thumbnail binary

Raises **RuntimeError** – image does not contain thumbnail

list_all() → List[str]

List all EXIF tags contained in the image.

set(*attribute: str*, *value*) → None

Set the value of the specified attribute.

Parameters

- **attribute** – image EXIF attribute name

- **value** (*corresponding Python type*) – tag value

1.2.2 Data Types

Note: All data types are constructed using the plum (pack/unpack memory) package.

Flash

class exif.**Flash**(*, *flash_fired: Union[int, bool]*, *flash_return: Union[int, exif._datatypes.FlashReturn]*, *flash_mode: Union[int, exif._datatypes.FlashMode]*, *flash_function_not_present: Union[int, bool]*, *red_eye_reduction_supported: Union[int, bool]*, *reserved: int*)

Status of the camera's flash when the image was taken. (Reported by the `flash` tag.)

class exif.**FlashMode**

Flash mode of the camera.

class exif.**FlashReturn**

Flash status of returned light.

1.2.3 Enumerations

ColorSpace

class exif.**ColorSpace**

Color space specifier.

SRGB = 1

sRBG

UNCALIBRATED = 65535

Uncalibrated or Other

ExposureMode

class exif.**ExposureMode**

Exposure mode set when the image was shot.

AUTO_BRACKET = 2

Auto Bracket

AUTO_EXPOSURE = 0

Auto Exposure

MANUAL_EXPOSURE = 1

Manual Exposure

ExposureProgram

class exif.**ExposureProgram**

Class of the program used by the camera to set exposure when the picture is taken.

ACTION_PROGRAM = 6
Action Program (Biased Toward Fast Shutter Speed)

APERTURE_PRIORITY = 3
Aperture Priority

CREATIVE_PROGRAM = 5
Creative Program (Biased Toward Depth of Field)

LANDSCAPE_MODE = 8
Landscape Kode (For Landscape Photos with the Background in Focus)

MANUAL = 1
Manual

NORMAL_PROGRAM = 2
Normal Program

NOT_DEFINED = 0
Not Defined

PORTRAIT_MODE = 7
Portrait Mode (For Closeup Photos with the Background out of Focus)

SHUTTER_PRIORITY = 4
Shutter Priority

GpsAltitudeRef

class exif.**GpsAltitudeRef**
Altitude used as the reference altitude.

ABOVE_SEA_LEVEL = 0
Above Sea Level

BELOW_SEA_LEVEL = 1
Below Sea Level

LightSource

class exif.**LightSource**
Class of the program used by the camera to set exposure when the picture is taken.

CLOUDY_WEATHER = 10
Cloudy Weather

COOL_WHITE_FLUORESCENT = 14
Cool White Fluorescent (W 3900 - 4500K)

D50 = 23
D50

D55 = 20
D55

D65 = 21
D65

D75 = 22
D75

DAYLIGHT = 1
Daylight

DAYLIGHT_FLUORESCENT = 12
Daylight Fluorescent (D 5700 - 7100K)

DAY_WHITE_FLUORESCENT = 13
Day White Fluorescent (N 4600 - 5400K)

FINE_WEATHER = 9
Fine Weather

FLASH = 4
Flash

FLUORESCENT = 2
Fluorescent

ISO_STUDIO_TUNGSTEN = 24
ISO Studio Tungsten

OTHER = 255
Other Light Source

SHADE = 11
Shade

STANDARD_LIGHT_A = 17
Standard Light A

STANDARD_LIGHT_B = 18
Standard Light B

STANDARD_LIGHT_C = 19
Standard Light C

TUNGSTEN = 3
Tungsten (Incandescent Light)

UNKNOWN = 0
Unknown

WHITE_FLUORESCENT = 15
White Fluorescent (WW 3200 - 3700K)

MeteringMode

class exif.**MeteringMode**
Metering mode.

AVERAGE = 1
Average

CENTER_WEIGHTED_AVERAGE = 2
Center Weighted Average

MULTI_SPOT = 4
Multi Spot

OTHER = 255
Other

PARTIAL = 6
Partial

PATTERN = 5
Pattern

SPOT = 3
Spot

UNKNOWN = 0
Unknown

Orientation

class exif.**Orientation**
Image orientation in terms of rows and columns.

BOTTOM_LEFT = 4
The 0th row is at the visual bottom of the image and the 0th column is the visual left-hand side.

BOTTOM_RIGHT = 3
The 0th row is at the visual bottom of the image and the 0th column is the visual right-hand side.

LEFT_BOTTOM = 8
The 0th row is the visual left-hand side of the image and the 0th column is the visual bottom.

LEFT_TOP = 5
The 0th row is the visual left-hand side of the image and the 0th column is the visual top.

RIGHT_BOTTOM = 7
The 0th row is the visual right-hand side of the image and the 0th column is the visual bottom.

RIGHT_TOP = 6
The 0th row is the visual right-hand side of the image and the 0th column is the visual bottom.

TOP_LEFT = 1
The 0th row is at the visual top of the image and the 0th column is the visual left-hand side.

TOP_RIGHT = 2
The 0th row is at the visual top of the image and the 0th column is the visual right-hand side.

ResolutionUnit

class exif.**ResolutionUnit**
Unit for measuring X resolution and Y resolution tags.

CENTIMETERS = 3
Centimeters

INCHES = 2
Inches or Unknown

Saturation

class exif.**Saturation**
Saturation processing applied by camera.

HIGH = 2
High Saturation

LOW = 1
Low Saturation

NORMAL = 0
Normal Saturation

SceneCaptureType

class exif.**SceneCaptureType**
Type of scene that was shot or the mode in which the image was shot.

LANDSCAPE = 1
Landscape

NIGHT_SCENE = 3
Night Scene

PORTRAIT = 2
Portrait

STANDARD = 0
Standard

SensingMethod

class exif.**SensingMethod**
Image sensor type on the camera or input device.

COLOR_SEQUENTIAL_AREA_SENSOR = 5
Color Sequential Area Sensor

COLOR_SEQUENTIAL_LINEAR_SENSOR = 8
Color Sequential Linear Sensor

NOT_DEFINED = 1
Not Defined

ONE_CHIP_COLOR_AREA_SENSOR = 2
One-Chip Color Area Sensor

THREE_CHIP_COLOR_AREA_SENSOR = 4
Three-Chip Color Area Sensor

TRILINEAR_SENSOR = 7
Trilinear Sensor

TWO_CHIP_COLOR_AREA_SENSOR = 3
Two-Chip Color Area Sensor

Sharpness

class exif.**Sharpness**
Sharpness processing applied by camera.

HARD = 2
Hard

NORMAL = 0
Normal

SOFT = 1
Soft

WhiteBalance

class exif.**WhiteBalance**
White balance mode set when the image was shot.

AUTO = 0
Auto White Balance

MANUAL = 1
Manual White Balance

1.2.4 Image Attributes

The `exif.Image` interface provides access to the following EXIF tags as Python attributes:

- acceleration
- aperture_value
- artist
- bits_per_sample
- body_serial_number
- brightness_value
- camera_elevation_angle
- camera_owner_name
- cfa_pattern
- color_space
- components_configuration
- compressed_bits_per_pixel
- compression
- contrast
- copyright
- custom_rendered
- datetime
- datetime_digitized
- datetime_original
- device_setting_description
- digital_zoom_ratio
- exif_version
- exposure_bias_value
- exposure_index

- exposure_mode
- exposure_program
- exposure_time
- f_number
- file_source
- flash
- flash_energy
- flashpix_version
- focal_length
- focal_length_in_35mm_film
- focal_plane_resolution_unit
- focal_plane_x_resolution
- focal_plane_y_resolution
- gain_control
- gamma
- gps_altitude
- gps_altitude_ref
- gps_area_information
- gps_datestamp
- gps_dest_bearing
- gps_dest_bearing_ref
- gps_dest_distance
- gps_dest_distance_ref
- gps_dest_latitude
- gps_dest_latitude_ref
- gps_dest_longitude
- gps_dest_longitude_ref
- gps_differential
- gps_dop
- gps_horizontal_positioning_error
- gps_img_direction
- gps_img_direction_ref
- gps_latitude
- gps_latitude_ref
- gps_longitude
- gps_longitude_ref

- gps_map_datum
- gps_measure_mode
- gps_processing_method
- gps_satellites
- gps_speed
- gps_speed_ref
- gps_status
- gps_timestamp
- gps_track
- gps_track_ref
- gps_version_id
- humidity
- image_description
- image_height
- image_unique_id
- image_width
- iso_speed
- iso_speed_latitude_yyy
- iso_speed_latitude_zzz
- jpeg_interchange_format
- jpeg_interchange_format_length
- lens_make
- lens_model
- lens_serial_number
- lens_specification
- light_source
- make
- maker_note
- matrix_coefficients
- max_aperture_value
- metering_mode
- model
- oecf
- offset_time
- offset_time_digitized
- offset_time_original

- orientation
- photographic_sensitivity
- photometric_interpretation
- pixel_x_dimension
- pixel_y_dimension
- planar_configuration
- pressure
- primary_chromaticities
- recommended_exposure_index
- reference_black_white
- related_sound_file
- resolution_unit
- rows_per_strip
- samples_per_pixel
- saturation
- scene_capture_type
- scene_type
- sensing_method
- sensitivity_type
- sharpness
- shutter_speed_value
- software
- spatial_frequency_response
- spectral_sensitivity
- standard_output_sensitivity
- strip_byte_counts
- strip_offsets
- subject_area
- subject_distance
- subject_distance_range
- subject_location
- subsampling_ratio_of_y_to_c
- subsec_time
- subsec_time_digitized
- subsec_time_original
- temperature

- transfer_function
- user_comment
- water_depth
- white_balance
- white_point
- x_resolution
- xp_author
- xp_comment
- xp_keywords
- xp_subject
- xp_title
- y_and_c_positioning
- y_resolution

1.3 Installation

1.3.1 Requirements

- Python 3.7+

1.3.2 Installation Steps

Install `exif` from the command line using pip:

```
pip install exif
```

1.4 Release Notes

1.4.1 [1.3.5] Support initial stable release of `plum-py`. (2022-04-17)

Update package to support the initial stable release of its `plum-py` dependency.

1.4.2 [1.3.4] Decode Windows XP style tags as UTF-16. (2021-12-09)

Previously, the package decoded ASCII characters within Windows XP style tags. Now, the package decodes Windows XP style tags as UTF-16.

This patch addresses the following GitLab user issue:

- `xp_comment` and other Windows XP tags don't handle Unicode strings correctly. (https://gitlab.com/TNThieding/exif/-/issues/53)

1.4.3 [1.3.3] Omit unknown values from `get_all()` method. (2021-11-07)

Previously, `get_all()` did not catch exceptions when reading each EXIF tag. If a single attribute had an incorrectly-encoded value, the `get_all()` method raised an exception. Now, the `get_all()` method catches exceptions due to unknown or unreadable values and logs them as a warning.

This patch addresses the following GitLab user issue:

- `ValueError: 0 is not a valid Orientation` returned from the `Image.get_all()`. method (https://gitlab.com/TNThieding/exif/-/issues/52)

1.4.4 [1.3.2] Add support for writing various tags. (2021-09-04)

Previously, attempting to add the following tags to an image raised an `AttributeError`:

- Body Serial Number
- ISO Speed
- Lens Specification
- Lens Make
- Lens Model
- Lens Serial Number

This patch addresses the following GitLab user issue:

- Trouble setting tags. (https://gitlab.com/TNThieding/exif/-/issues/48)

1.4.5 [1.3.1] Fix value of `SceneCaptureType.NIGHT_SCENE`. (2021-07-03)

Previously, `SceneCaptureType.NIGHT_SCENE` erroneously had a value of `2`. Now, it has a value of `3` in accordance with the EXIF specification.

This patch contains changes submitted via GitLab merge request by the following user:

- Alex Mykyta (amykyta3)

1.4.6 [1.3.0] Consume latest version of `plum-py`. (2021-06-13)

Overhaul package internals to leverage `plum-py` version `0.5.0` and higher. Since the `plum-py` package only supports Python 3.7 and higher, this release drops support for Python 3.6.

1.4.7 [1.2.2] Late-April 2021 bug fix rollup. (2021-04-23)

This patch addresses the following GitLab user issues:

- Add a workaround for `flash` attribute in Python 3.10 to temporarily address bit field `TypeError`. (Upstream `plum-py` Issue: https://gitlab.com/dangass/plum/-/issues/129)
- `UnpackError` occurs when reading a bad IFD. (https://gitlab.com/TNThieding/exif/-/issues/38)

1.4.8 [1.2.1] Preserve empty IFDs and EXIF version in `delete_all()`. (2021-03-23)

Previously, attempting to re-add EXIF tags to an image after calling `delete_all()` on it raised a `RuntimeError` since it removed the EXIF version tag and the IFD structure. Now, `delete_all()` still removes user-facing tags but preserves the EXIF version number and the empty IFD structures and their pointers. This enables users to add tags back to an image after using `delete_all()`.

This patch addresses the following GitLab user issue:

- `RuntimeError` when adding tags after calling `delete_all()`. (https://gitlab.com/TNThieding/exif/-/issues/33)

1.4.9 [1.2.0] Add `get_all()` and `list_all()` methods. (2021-02-06)

Add `list_all()` method that returns a list of all EXIF tags in an image (without including method names and unknown tags like `dir()` includes). Similarly, add `get_all()` method that generates a dictionary mapping each tag names to its value.

This patch addresses the following GitLab user issue:

- API for retrieving all EXIF tags. (https://gitlab.com/TNThieding/exif/-/issues/32)

1.4.10 [1.1.0] Add type hints to public API. (2021-02-04)

Update `Image` class to include `mypy`-compliant type hints.

1.4.11 [1.0.5] Fix corruption errors when adding tags to previously non-EXIF images. (2021-01-23)

Previously, adding EXIF tags to non-EXIF images resulted in an incorrectly-calculated APP1 segment length value. This resulted in some image tools and libraries reporting that the file was corrupt. Now, the APP1 segment length value is calculated correctly by excluding the APP1 marker length from the segment length.

This patch addresses the following GitLab user issue:

- Corrupt JPEG data error caused by writing EXIF data. (https://gitlab.com/TNThieding/exif/-/issues/30)

1.4.12 [1.0.4] Fix adding focal length and user comment tags to images. (2020-11-28)

Previously, attempting to add either a focal length or user comment tag to an image resulted in an `AttributeError`. In addition, this patch changes attribute getters and setters such that they are not case-sensitive (e.g., `image.Copyright` is treated the same as `image.copyright`).

This patch addresses the following GitLab user issue:

- Cannot add user comments to images without preexisting metadata. (https://gitlab.com/TNThieding/exif/issues/24)

This release includes the following under-the-hood changes:

- Don't distribute unit tests with the packaged source code (e.g., when installing via `pip`).

1.4.13 [1.0.3] Fix `ValueError` when SSHORT are present. (2020-11-15)

Previously, reading signed short integers resulted in a `ValueError`.

This patch addresses the following GitLab user issue:

- Signed short integers in EXIF are not supported. (https://gitlab.com/TNThieding/exif/issues/28)

This patch contains changes submitted via GitLab merge request by the following user:

- Justin Saunders (jumaka1)

1.4.14 [1.0.2] Fix `ZeroDivisionError` when reading lens specification with unknown F number. (2020-10-18)

Previously, reading the lens specification attribute where the F values were unknown resulted in a `ZeroDivisionError` since unknown is encoded as 0/0. Now, the value is returned as `0` and the exception is no longer raised.

This patch addresses the following GitLab user issue:

- `ZeroDivisionError` reported when reading `lens_specification`. (https://gitlab.com/TNThieding/exif/issues/26)

1.4.15 [1.0.1] Fix `UnpackError` when reading ASCII tags with shorter value than expected. (2020-09-03)

Previously, reading an ASCII tag whose value was shorter than the specified size (i.e., with excess trailing null bytes) resulted in a `UnpackError`. Now, the package returns the tag value with excess bytes stripped off. It also issues a `RuntimeWarning` stating the nonconformity to the EXIF standard and how many extra bytes were found.

This patch addresses the following GitLab user issue:

- Cannot read EXIF tag containing excess trailing bytes. (https://gitlab.com/TNThieding/exif/issues/23)

1.4.16 [1.0.0] Support adding tags and adding EXIF to non-EXIF images. (2020-07-11)

Initial release with full support for adding new tags to images. This includes adding EXIF tags to an image without any pre-existing metadata (e.g., a JPEG produced by a scanner).

In addition, SHORT tags could only previously be added if pre-existing tags were deleted to make room. Now, this code dynamically expands and re-packs the EXIF/APP1 metadata section to facilitate adding new tags to images without size limitations. ASCII tags can now be modified to a value longer than their original length too.

Add enumeration for the following tag:

- GPS altitude reference

1.4.17 [0.12.0] Add preliminary support for adding IFD tags to images. (2020-07-05)

Support adding the following tag types:

- SHORT (except for TIFF attributes)

Add data types and enumerations for the following tags:

- Flash
- Light source

This release also addresses the following anomalous behavior:

- Previously, thumbnail IFD tags would overwrite the primary image's. Now, thumbnail IFD tags are only included if they are not included in the primary image IFD (e.g., `jpeg_interchange_format`).
- Include thumbnail tags during deletion with `delete_all()` method.

Note: Refer to the known limitations page for an up-to-date list of stipulations, limitations, and workarounds.

1.4.18 [0.11.2] Overhaul internal bytes processing and drop Python 3.5 support. (2020-07-04)

This under-the-hood change significantly simplifies and improves internal bytes processing by using hte `plum-py` (pack / unpack memory) package instead of a custom hexadecimal string interface like before. This patch also includes minor, benign bug fixes with hexadecimal processing. These changes will facilitate future development (e.g., support for adding new tags to images).

Since the `plum-py` package only supports Python 3.6 and higher, this version drops support for Python 3.5.

1.4.19 [0.11.1] Accept file paths and bytes when instantiating `Image`. (2020-06-30)

In addition to accepting an image file descriptor, also support instantiating `Image` with file paths or bytes (e.g., already-read files).

Part of this release contains changes submitted via GitHub pull request by the following user:

- chbndrhnns

1.4.20 [0.11.0] Add `delete_all()` method. (2020-06-06)

Add a new method called `delete_all()` that deletes all known EXIF tags in an `Image` object.

Add enumeration for the following tag:

- Resolution unit

This minor release addresses the following GitHub user issue:

- Removing all known EXIF values. (https://github.com/TNThieding/exif/issues/29)

This minor release contains changes submitted via GitHub pull request by the following user:

- ArgiesDario

1.4.21 [0.10.0] Add additional tag enumerations. (2020-05-31)

Add enumerations for the following tags:

- Exposure mode
- Exposure program
- Metering mode

- Scene capture type
- Sensing method
- White balance

1.4.22 [0.9.0] Add thumbnail image accessor. (2020-05-30)

Add `get_thumbnail()` method to extract bytes representing a thumbnail JPEG.

This patch addresses the following GitHub user issue:

- Extract thumbnail from the EXIF metadata. (https://github.com/TNThieding/exif/issues/28)

1.4.23 [0.8.6] Make `get()` return default value if tag isn't readable. (2020-05-29)

Previously, using `get()` to read a tag that can't be read by this package raised a `NotImplementedError`. Now, `get()` returns the default value (i.e., `None` if not specified otherwise) if the specified tag cannot be read.

This patch addresses the following GitHub user issue:

- Method `gets()` raises `NotImplementedError`. (https://github.com/TNThieding/exif/issues/30)

1.4.24 [0.8.5] Fix `exif_version` attribute. (2020-05-18)

Add support for reading `exif_version` attribute.

This patch addresses the following GitLab user issue:

- Reading `exif_version` fails with `NotImplementedError`. (https://gitlab.com/TNThieding/exif/issues/20)

1.4.25 [0.8.4] Restore Python 3.5 support. (2020-05-10)

Remove format string usage throughout package to restore Python 3.5 support. Add Python 3.5 testing to CI/CD pipeline.

This patch addresses the following GitHub and GitLab user issues:

- Broken Python 3.5 compatibility with Release 0.8.3. (https://gitlab.com/TNThieding/exif/-/issues/21)
- Dependency on enum34 makes it impossible to build a conda package. (https://github.com/TNThieding/exif/issues/25)

This patch contains changes submitted via GitHub pull request by the following user:

- RKrahl

1.4.26 [0.8.3] Mid-April 2020 bug fix rollup. (2020-04-20)

This patch addresses the following GitHub user issues:

- Fix reading ASCII tags containing 3 characters or less. (See https://github.com/TNThieding/exif/issues/12 for more information.)
- Fix *gps_longitude_ref* and *gps_latitude_ref* decoding. (See https://github.com/TNThieding/exif/issues/24 for more information).

1.4.27 [0.8.2] Early-March 2020 bug fix rollup. (2020-03-10)

This patch addresses the following GitHub user issues:

- Update PyPI classification to more clearly indicate that this package only supports Python 3. (See https://github.com/TNThieding/exif/issues/20 for discussion.)
- Add read-only support for Windows XP style tags. (See https://github.com/TNThieding/exif/issues/22 for more information.)
- Fix a benign cursor increment bug in `_app1_metadata.py`. (See https://github.com/TNThieding/exif/issues/18 for more information.)

This patch also addresses the following issues:

- The `offset_time_digitized` was previously incorrectly mapped to `offset_time_original`.

1.4.28 [0.8.1] Restructure tag type behavior. (2019-07-28)

Replace complex and duplicated `if` statements with polymorphic tag datatypes.

1.4.29 [0.8.0] Add `has_exif` attribute. (2019-07-07)

Previously, instantiating an `Image` with a non-EXIF file raised an `IOError`. Now, `Image` instantiation always succeeds and the `has_exif` attribute reports whether or not the image currently has EXIF metadata.

1.4.30 [0.7.0] Support modifying image rotation. (2019-06-23)

Add support for modifying metadata with the SHORT datatype (e.g., image orientation). Add `Orientation` enumeration to facilitate rotating images.

1.4.31 [0.6.0] Drop Python 2 support. (2019-06-16)

Remove legacy Python 2 syntax from code.

This release includes the following under-the-hood changes:

- Migrate repository from GitHub to GitLab (including CI/CD).
- Pylint cleanup regarding Python 3 syntax.

1.4.32 [0.5.1] Mid-April 2019 bug fix rollup. (2019-04-14)

This patch addresses the following GitHub user issues:

- Previously, instantiating `Image` with an image file without a valid APP1 segment caused an infinite loop if the APP1 segment marker was found in the hexadecimal of the image itself. Now, the package raises an `IOError` indicating that the file isn't properly EXIF-encoded. (See https://github.com/TNThieding/exif/issues/14 for more information.)
- Previously, accessing an image's `user_comment` attribute raised an exception stating the datatype was unknown. Now, the package parses the `user_comment` attribute's special data structure as described in the EXIF specification so that users can access its value. (See https://github.com/TNThieding/exif/issues/15 for more information.)

1.4.33 [0.5.0] Add index/item access support. (2019-04-13)

Support indexed get, set, and delete access of EXIF tags. Also, offer `set()` and `delete()` methods.

This release includes the following under-the-hood changes:

- Add minimum Pylint score check to tox configuration.
- Update usage page to describe workflow and different access paradigms.

See https://github.com/TNThieding/exif/issues/13 for more information.

1.4.34 [0.4.0] Add `get()` method. (2019-03-16)

Previously, this package did not offer a mechanism to return a default value when attempting to access a missing tag, causing users to rely heavily on try-except statements. Now, the `Image` class offers a `get()` method. This method accepts a `default=None` keyword argument specifying the return value if the target attribute does not exist.

See https://github.com/TNThieding/exif/issues/7 for more information.

1.4.35 [0.3.1] Fix little endian support. (2018-02-10)

Previously, this package did not fully support little endian EXIF metadata in images, raising `ValueError` exceptions. Now, reading EXIF hexadecimal strings and values takes endianness into account.

This release includes the following under-the-hood changes:

- Move tag reading and modification functions into the IFD tag class.
- Add enumerations for color space, sharpness, and saturation as a proof-of-concept for leveraging enumerations. (More enumerations coming soon in a future release!)
- Improve test coverage.

See https://github.com/TNThieding/exif/issues/5 for more information.

1.4.36 [0.3.0] Add attribute list support. (2018-12-26)

Implement mechanism for listing EXIF tags in an image using `dir()`.

This release includes the following under-the-hood changes:

- Modularize hexadecimal string interface into an internal class.
- More robust test coverage and verification of hexadecimal data.

1.4.37 [0.2.0] Add tag delete support. (2018-12-25)

Add EXIF tag deletion support via Python delete attribute notation.

1.4.38 [0.1.0] Initial alpha release. (2018-12-23)

Release initial alpha version of `exif` package with the following features:

- Support for reading EXIF tags via Python get attribute notation.
- Support for modifying existing EXIF tags via Python set attribute notation.

1.5 Known Limitations

This package contains the following known limitations:

- Accessing SLONG tags is not supported (since no IFD tags in the EXIF specification are SLONG type).
- EXIF IFDs cannot be added to images that only contain IFD 0 (and/or IFD 1). However, GPS IFDs can be inserted if there's a subsequent IFD 1 segment. When adding metadata to a previously non-APP1 image, this is not a concern since the package adds empty 0, EXIF, and GPS IFDs.
- Modifying Windows XP tags is not supported.

1.6 Usage

Warning: Back up your photos before using this tool! You are responsible for any unexpected data loss that may occur through improper use of this package.

1.6.1 Opening an Image

Open an image with EXIF metadata using the Python `open()` built-in function. Ensure the binary mode flag (i.e. `'rb'`) is set. Pass this image file object into the `exif.Image` class:

```
>>> from exif import Image
>>> with open('grand_canyon.jpg', 'rb') as image_file:
...     my_image = Image(image_file)
...
```

Alternatively, supply a file path or image bytes to the `exif.Image` class:

```
>>> my_image = Image('grand_canyon.jpg')

>>> from exif import Image
>>> with open('grand_canyon.jpg', 'rb') as image_file:
```

(continues on next page)

(continued from previous page)

```
...     image_bytes = image_file.read()
...
>>> my_image = Image(image_bytes)
```

Verify that an image has EXIF metadata by leveraging the `has_exif` attribute:

```
>>> my_image.has_exif
True
```

1.6.2 Accessing Tags

List all tags present in an image with `dir()`:

```
>>> dir(my_image)
['<unknown EXIF tag 59932>', '<unknown EXIF tag 59933>', '_exif_ifd_pointer', '_gps_
↪ifd_pointer', '_segments', 'aperture
_value', 'brightness_value', 'color_space', 'components_configuration', 'compression',
↪ 'datetime', 'datetime_digitized',
'datetime_original', 'exif_version', 'exposure_bias_value', 'exposure_mode',
↪'exposure_program', 'exposure_time', 'f_
number', 'flash', 'flashpix_version', 'focal_length', 'focal_length_in_35mm_film',
↪'get', 'get_file', 'get_thumbnail',
'gps_altitude', 'gps_altitude_ref', 'gps_datestamp', 'gps_dest_bearing', 'gps_dest_
↪bearing_ref', 'gps_horizontal_
positioning_error', 'gps_img_direction', 'gps_img_direction_ref', 'gps_latitude',
↪'gps_latitude_ref', 'gps_longitude',
'gps_longitude_ref', 'gps_speed', 'gps_speed_ref', 'gps_timestamp', 'has_exif', 'jpeg_
↪interchange_format', 'jpeg_
interchange_format_length', 'lens_make', 'lens_model', 'lens_specification', 'make',
↪'maker_note', 'metering_mode',
'model', 'orientation', 'photographic_sensitivity', 'pixel_x_dimension', 'pixel_y_
↪dimension', 'resolution_unit',
'scene_capture_type', 'scene_type', 'sensing_method', 'shutter_speed_value', 'software
↪', 'subject_area', 'subsec_time_
digitized', 'subsec_time_original', 'white_balance', 'x_resolution', 'y_and_c_
↪positioning', 'y_resolution']
```

The `Image` class facilitates three different tag access paradigms. Leverage attribute syntax for an intuitive object-oriented feel. Alternatively, leverage indexed/item syntax of additional methods for more control.

Attribute Syntax

Access EXIF tag values as attributes of the `Image` instance:

```
>>> my_image.gps_latitude
(36.0, 3.0, 11.08)
>>> my_image.gps_longitude
(112.0, 5.0, 4.18)
>>> my_image.make
'Apple'
>>> my_image.model
'iPhone 7'
```

Change the EXIF tag value by modifying the attribute value:

```
>>> my_image.make = "Python"
```

Set new attribute values to add EXIF tags to an image:

```
>>> from exif import LightSource
>>> my_image.light_source = LightSource.DAYLIGHT
```

Use `del` notation to remove EXIF tags from the image:

```
>>> del my_image.gps_latitude
>>> del my_image.gps_longitude
```

Indexed/Item Syntax

Alternatively, use indexed/item syntax to read, modify, add, and remove attribute tags:

```
>>> my_image["orientation"]
1
>>> my_image["software"] = "Python Script"
>>> del my_image["maker_note"]
```

Methods

Leverage the dictionary-style `get()` method to gracefully handle cases where attributes do not exist:

```
>>> my_image.get("color_space")
<ColorSpace.UNCALIBRATED: 65535>
>>> my_image.get("nonexistent_tag")
None
```

Call `set()` with a tag name and value to add or modify it:

```
>>> self.image.set("model", "EXIF Package")
```

Call `delete()` with a tag name to remove it from the image:

```
>>> self.image.delete("datetime_original")
```

Erase all EXIF tags in an image using the `delete_all()` method:

```
>>> my_image.delete_all()
```

1.6.3 Writing/Saving the Image

Write the image with modified EXIF metadata to an image file using `open()` in binary write (i.e. `'wb'`) mode:

```
>>> with open('modified_image.jpg', 'wb') as new_image_file:
...     new_image_file.write(my_image.get_file())
...
```

Extract the thumbnail embedded within the EXIF data by using `get_thumbnail()` instead of `get_file()`.

1.6.4 Cookbook

Add Geolocation

Add geolocation metadata to an image by providing tuples of degrees, minutes, and decimal seconds:

```
>>> from exif import Image
>>> image = Image("cleveland_public_square.jpg")
>>>
>>> image.gps_latitude = (41.0, 29.0, 57.48)
>>> image.gps_latitude_ref = "N"
>>> image.gps_longitude = (81.0, 41.0, 39.84)
>>> image.gps_longitude_ref = "W"
>>> image.gps_altitude = 199.034  # in meters
>>> image.gps_altitude_ref = GpsAltitudeRef.ABOVE_SEA_LEVEL
>>>
>>> # Then, save image to desired location using code discussed above.
```

Add Timestamps

Use `datetime_original` and `datetime_digitized` to add timestamps to an image (e.g., from a scanner):

```
>>> from exif import Image, DATETIME_STR_FORMAT
>>> from datetime import datetime
>>> datetime_taken = datetime(year=1999, month=12, day=31, hour=23, minute=49,␣
↪second=12)
>>> datetime_scanned = datetime(year=2020, month=7, day=11, hour=10, minute=11,␣
↪second=37)
>>>
>>> image = Image("my_scanned_photo.jpg")
>>> image.datetime_original = datetime_taken.strftime(DATETIME_STR_FORMAT)
>>> image.datetime_digitized = datetime_scanned.strftime(DATETIME_STR_FORMAT)
>>> # Then, save image to desired location using code discussed above.
```

Use with NumPy and OpenCV Image Encoder

This sample script was provided by Rune Monzel.

It demonstrates how to use this package with NumPy and an image encoder, specifically OpenCV in this case:

```
import exif
import cv2
import numpy as np

# Create a random 2D array within range [0 255]
image = (np.random.rand(800, 1200) * 255).astype(np.uint8)

# decode to the appropriate format
# jpg -> compressed with information loss)
status, image_jpg_coded = cv2.imencode('.jpg', image)
print('successful jpg encoding: %s' % status)
# tif -> no compression, no information loss
status, image_tif_coded = cv2.imencode('.jpg', image)
print('successful tif encoding: %s' % status)
```

(continues on next page)

(continued from previous page)

```
# to a byte string
image_jpg_coded_bytes = image_jpg_coded.tobytes()
image_tif_coded_bytes = image_tif_coded.tobytes()

# using the exif format to add information
exif_jpg = exif.Image(image_jpg_coded_bytes)
exif_tif = exif.Image(image_tif_coded_bytes)

# providing some information
user_comment = "random image"
software = "created in python with numpy"
author = "Rune Monzel"

# adding information to exif files:
exif_jpg["software"] = exif_tif["software"] = software
exif_jpg["user_comment"] = exif_tif["user_comment"] = user_comment

# show existing tags
print(exif_jpg.list_all())

# save image
with open(r'random.tif', 'wb') as new_image_file:
    new_image_file.write(exif_tif.get_file())
with open(r'random.jpg', 'wb') as new_image_file:
    new_image_file.write(exif_jpg.get_file())
```

Python Module Index

e

Index