py3exiv2 Documentation

Release 0.3.0

Vincent Vande Vyvre

Apr 15, 2019

Contents

py3exiv2 is a Python 3 binding to exiv2, the C++ library for manipulation of EXIF (EXchangeable Image File), IPTC (International Press Telecommunications Council) and XMP (eXtensible Metadata Platform) image metadata. It is a python 3 module that allows your scripts to read and write metadata (EXIF, IPTC, XMP, thumbnails) embedded in image files (JPEG, TIFF, ...).

It is designed as a high-level interface to the functionalities offered by libexiv2. Using python's built-in data types and standard modules, it provides easy manipulation of image metadata.

py3exiv2 is distributed under the GPL version 3 license.

The main content of the code was initially written by Olivier Tilloy for Python 2 under the name pyexiv2.

Differences between py3exiv2 (Python 3) and pyexiv2 (Python 2)

The module's name and the syntax are unchanged, your code written previously for Python 2 may run with py3exiv2, however there's three thinks that you should be care.

- The deprecated `IptcTag.raw_values` was removed in py3exiv2, use `IptcTag.raw_value` instead.
- The `pyexiv2.preview.Preview.data` is not implemented, use `pyexiv2.exif.ExifThumbnail.data` instead.
- All the string returned by any `Tag.value` are unicode, but you don't need to convert yours strings in bytes to set a value for a tag which only accept ASCII characters, this is the job of py3exiv2.

Contents:

CHAPTER 1

API documentation

1.1 pyexiv2

The top-level module `pyexiv2`.

Attributes

- *version_info*
- *__version__*
- *exiv2_version_info*
- *__exiv2_version__*

Description

Top-level module. All other modules are imported from `pyexiv2`.

Documentation

Attributes

`version_info`
A tuple containing the three components of the version number: major, minor, micro.

`__version__`
The version of the module as a string (major.minor.micro).

`exiv2_version_info`
A tuple containing the three components of the version number of libexiv2: major, minor, micro.

`__exiv2_version__`
The version of libexiv2 as a string (major.minor.micro).

1.2 pyexiv2.metadata

class pyexiv2.metadata.**ImageMetadata**

Instance Attributes

- *buffer*
- *comment*
- *dimensions*
- *exif_keys*
- *iptc_charset*
- *iptc_keys*
- *mime_type*
- *previews*
- *xmp_keys*

Instance Methods

- *copy(other, exif=True, iptc=True, xmp=True, comment=True)*
- *__delitem__(key)*
- *get_aperture(self)*
- *get_exposure_data(self, float_=False)*
- *get_focal_length(self)*
- *get_iso(self)*
- *__getitem__(key)*
- *get_orientation(self)*
- *get_rights_data(self)*
- *get_shutter_speed(self, float_=False)*
- *read()*
- *__setitem__(key)*
- *write(preserve_timestamps=False)*

Description

The *pyexiv2.metadata.ImageMetadata* is a container for all the metadata embedded in an image.

It provides convenient methods for the manipulation of EXIF, IPTC and XMP metadata embedded in image files such as JPEG and TIFF files, using Python types. It also provides access to the previews embedded in an image.

Documentation

Instanciation

class pyexiv2.metadata.**ImageMetadata**(*filename*)
Inherits: MutableMapping

Argument:

- *filename* str(path of an image file)

See *read()*

Attributes

buffer
Return the image data as bytes. This is useful to reduce disk access, the data can be send to an image library.

Example with Pillow:

```
>>> from PIL import Image
>>> import io
>>> import pyexiv2
>>> meta = pyexiv2.ImageMetadata("lena.jpg")
>>> meta.read()
>>> byteio = io.BytesIO(meta.buffer)
>>> img = Image.open(byteio)
>>> img.show()
```

comment
The image comment.

dimensions
A tuple containing the width and height of the image, expressed in pixels.

exif_keys
List of the keys of the available EXIF tags.

iptc_charset
An optional character set the IPTC data is encoded in.

iptc_keys
List of the keys of the available IPTC tags.

mime_type
The mime type of the image, as a string.

previews
List of the previews available in the image, sorted by increasing size.

xmp_keys
List of the keys of the available XMP tags.

Methods

copy(*other*, *exif=True*, *iptc=True*, *xmp=True*, *comment=True*)
Copy the metadata to another image. The metadata in the destination is overridden. In particular, if the destination contains e.g. EXIF data and the source doesn't, it will be erased in the destination, unless explicitly omitted.

Arguments:

- *other* An instance of :class:pyexiv2.metadata.ImageMetadata, the destination metadata to copy to (it must have been read() beforehand)
- *exif* (boolean) – Whether to copy the EXIF metadata
- *iptc* (boolean) – Whether to copy the IPTC metadata
- *xmp* (boolean) – Whether to copy the XMP metadata
- *comment* (boolean) – Whether to copy the image comment

__delitem__(*key*)
Delete a metadata tag for a given key.

Argument:

- *key* Metadata key in the dotted form *familyName.groupName.tagName* where *familyName* may be one of *exif*, *iptc* or *xmp*.

Raises KeyError if the tag with the given key doesn't exist

get_aperture (*self*)
Returns the fNumber as float.

get_exposure_data (*self*, *float_=False*)
Returns the exposure parameters of the image.

The values are returned as a dict which contains:

- *"iso"*: the ISO value
- *"speed"*: the exposure time
- *"focal"*: the focal length
- *"aperture"*: the fNumber
- *"orientation"*: the orientation of the image

When a tag is not set, the value will be None.

Argument:

- *float_* If False, default, the value of the exposure time is returned as rational otherwise a float is returned.

get_focal_length (*self*)
Returns the focal length as float.

get_iso (*self*)
Returns the ISO value as integer.

__getitem__ (*key*)
Get a metadata tag for a given key.

Argument:

- *key* Metadata key in the dotted form familyName.groupName.tagName where familyName may be one of exif, iptc or xmp.

Raises KeyError if the tag doesn't exist

get_orientation (*self*)
Returns the orientation of the image as integer.

If the tag is not set, the value 1 is returned.

get_rights_data (*self*)
Returns the author and copyright info.

The values are returned as a dict which contains:

- *"creator"*: the value of Xmp.dc.creator
- *"artist"*: the value of Exif.Image.Artist
- *"rights"*: the value of Xmp.dc.rights
- *"copyright"*: the value of Exif.Image.Copyright
- *"marked"*: the value of Xmp.xmpRights.Marked
- *"usage"*: the value of Xmp.xmpRights.UsageTerms

When a tag is not set, the value will be None.

get_shutter_speed(*self*, *float_=False*)
Returns the exposure time as rational or float or None if the tag is not set.

Argument:

- *float_* If False, default, the value is returned as rational otherwise a float is returned

read()
Read the metadata embedded in the associated image. It is necessary to call this method once before attempting to access the metadata (an exception will be raised if trying to access metadata before calling this method).

__setitem__(*key*, *tag_or_value*)
Set a metadata tag for a given key. If the tag was previously set, it is overwritten. As a handy shortcut, a value may be passed instead of a fully formed tag. The corresponding tag object will be instantiated.

Arguments:

- *key* Metadata key in the dotted form familyName.groupName.tagName where familyName may be one of exif, iptc or xmp.
- *tag_or_value* (pyexiv2.exif.ExifTag or pyexiv2.iptc.IptcTag or pyexiv2.xmp.XmpTag or any valid value type) – An instance of the corresponding family of metadata tag, or a value

Raises KeyError if the tag doesn't exist

write(*preserve_timestamps=False*)
Write the metadata back to the image.

Argument:

- *preserve_timestamps* (boolean) – Whether to preserve the file's original timestamps (access time and modification time)

1.3 pyexiv2.exif

This module provides the classes `ExifTag`, `ExifValueError` and `ExifThumbnail`.

class pyexiv2.exif.**ExifTag**

Instance Attributes

- *description*
- *human_value*
- *key*
- *label*
- *name*
- *raw_value*
- *section_description*
- *section_name*
- *type*
- *value*

Description

The `ExifTag` define an EXIF tag.

Documentation

Instanciation

class pyexiv2.exif.**ExifTag**(*key*, *value=None*, *_tag=None*)

An EXIF tag.

Arguments:

- *key* The key of the tag in the dotted form *familyName.groupName.tagName* where *familyName* = *exif*.
- *value* The value of the tag.

Here is a correspondance table between the EXIF types and the possible python types the value of a tag may take:

- Ascii: datetime.datetime(), datetime.date(), str()
- Byte, SByte: str()
- Comment: str()
- Long, SLong: [list of] int
- Short, SShort: [list of] int
- Rational, SRational: [list of] fractions.Fraction
- Undefined: str()

Attributes

description
The description of the tag.

human_value
A (read-only) human-readable representation of the value of the tag.

key
The key of the tag in the dotted form familyName.groupName.tagName where familyName = exif.

label
The title (label) of the tag.

name
The name of the tag (this is also the third part of the key).

raw_value
The raw value of the tag as a string.

section_description
The description of the tag's section.

section_name
The name of the tag's section.

type
The EXIF type of the tag (one of Ascii, Byte, SByte, Comment, Short, SShort, Long, SLong, Rational, SRational, Undefined).

value
The value of the tag as a python object.

class pyexiv2.exif.**ExifValueError**(*value*, *type*)
Exception raised when failing to parse the value of an EXIF tag.

Arguments:

- *value* The value that fails to be parsed
- *type* The EXIF type of the tag

class pyexiv2.exif.**ExifThumbnail**

Instance Attributes

- *extension*
- *mime_type*
- *data*

Instance Method

- *erase()*
- *set_from_file(path)*
- *write_to_file(path)*

Description

A thumbnail image optionally embedded in the IFD1 segment of the EXIF data.

The image is either a TIFF or a JPEG image.

Documentation

Instanciation

class pyexiv2.exif.ExifThumbnail(_metadata)

Argument:

- *_metadata* The `ImageMetadata` instance

Attributes

extension
The file extension of the preview image with a leading dot (e.g. .jpg).

mime_type
The mime type of the preview image (e.g. image/jpeg).

data
The preview data as a Python list. The data can be send to an image library.

Example with Pillow:

```
>>> from PIL import Image
>>> import io
>>> from pyexiv2 import ImageMetadata, exif
>>> meta = ImageMetadata("lena.jpg")
>>> meta.read()
>>> thumb = exif.ExifThumbnail(meta)
>>> byteio = io.BytesIO(bytes(thumb.data))
>>> img = Image.open(byteio)
>>> img.show()
```

Methods

erase()
Delete the thumbnail from the EXIF data. Removes all Exif.Thumbnail.*, i.e. Exif IFD1 tags.

set_from_file(*path*)
Set the EXIF thumbnail to the JPEG image path. This sets only the Compression, JPEGInterchangeFormat and JPEGInterchangeFormatLength tags, which is not all the thumbnail EXIF information mandatory according to the EXIF standard (but it is enough to work with the thumbnail).

Argument:

- *path* str(Path to a JPEG file).

write_to_file(*path*)
Write the thumbnail image to a file on disk. The file extension will be automatically appended to the path.

Argument:

- *path* str(Path to write the thumbnail to) - without an extension.

1.4 pyexiv2.iptc

This module provides the classes `IptcTag` and `IptcValueError`.

class pyexiv2.iptc.**IptcTag**

Instance Attributes

- *description*
- *key*
- *name*
- *photoshop_name*
- *raw_value*
- *record_description*
- *record_name*
- *repeatable*
- *title*
- *type*
- *value*

Description

The `IptcTag` define an IPTC tag.

Documentation

Instanciation

class pyexiv2.iptc.**IptcTag**(*key*, *value=None*, *_tag=None*)
An IPTC tag.

Arguments:

- *key* The key of the tag in the dotted form *familyName.groupName.tagName* where *familyName* = *iptc*.
- *value* The value of the tag.

This tag can have several values (tags that have the *repeatable* property).

Here is a correspondance table between the IPTC types and the possible python types the value of a tag may take:

- Short: int
- String: string
- Date: `datetime.date`
- Time: `datetime.time`
- Undefined: string

Attributes

description
The description of the tag.

key
The key of the tag in the dotted form familyName.groupName.tagName where familyName = iptc.

name
The name of the tag (this is also the third part of the key).

photoshop_name
The Photoshop name of the tag

raw_value
The raw values of the tag as a list of strings.

record_description
The description of the tag's record.

record_name
The name of the tag's record.

repeatable
Whether the tag is repeatable (accepts several values).

title
The title (label) of the tag.

type
The IPTC type of the tag (one of Short, String, Date, Time, Undefined).

value
The values of the tag as a list of python objects.

class `pyexiv2.iptc.`**IptcValueError**(*ValueError*)
Exception raised when failing to parse the value of an IPTC tag.

Attributes

value
The value that fails to be parsed

type
The IPTC type of the tag

1.5 pyexiv2.xmp

This module provides the classes `XmpTag` and `XmpValueError` and the following five functions to handle the XMP parser and name spaces.

`pyexiv2.xmp.`**`initialiseXmpParser`**`()`
Initialise the xmp parser.

Calling this method is usually not needed, as encode() and decode() will initialize the XMP Toolkit if necessary.

This function itself still is not thread-safe and needs to be called in a thread-safe manner (e.g., on program startup).

`pyexiv2.xmp.`**`closeXmpParser`**`()`
Close the xmp parser.

Terminate the XMP Toolkit and unregister custom namespaces.

Call this method when the XmpParser is no longer needed to allow the XMP Toolkit to cleanly shutdown.

`pyexiv2.xmp.`**`register_namespace`**`(`*name*, *prefix*`)`
Register a custom XMP namespace.

Overriding the prefix of a known or previously registered namespace is not allowed.

Arguments:

- *name* str() The name of the custom namespace (ending with a /), typically a URL (e.g. http://purl.org/dc/elements/1.1/)
- *prefix* str() The prefix for the custom namespace (keys in this namespace will be in the form Xmp.{prefix}.{something})

Raises:

- *ValueError* – if the name doesn't end with a /
- *KeyError* – if a namespace already exist with this prefix

`pyexiv2.xmp.`**`unregister_namespace`**`(`*name*`)`
Unregister a custom XMP namespace.

A custom namespace is identified by its name, not by its prefix.

Attempting to unregister an unknown namespace raises an error, as does attempting to unregister a builtin namespace.

Arguments:

- *name* str() The name of the custom namespace (ending with a /), typically a URL (e.g. http://purl.org/dc/elements/1.1/)

Raises:

- *ValueError* – if the name doesn't end with a /
- *KeyError* – if the namespace is unknown or a builtin namespace

`pyexiv2.xmp.`**`unregister_namespaces`**`()`
Unregister all custom XMP namespaces.

Builtin namespaces are not unregistered.

This function always succeeds.

`class` `pyexiv2.xmp.`**`XmpTag`**

Instance Attributes

- *description*
- *key*
- *name*
- *raw_value*
- *title*
- *type*
- *value*

Description

The `XmpTag` define an XMP tag.

Documentation

Instanciation

class `pyexiv2.xmp.`**XmpTag**(*key*, *value=None*, *_tag=None*)

An XMP tag.

Arguments:

- *key* The key of the tag in the dotted form *familyName.groupName.tagName* where *familyName* = *xmp*.
- *value* The value of the tag.

Here is a correspondance table between the XMP types and the possible python types the value of a tag may take:

- alt, bag, seq: list of the contained simple type
- lang alt: dict of (language-code: value)
- Boolean: boolean
- Colorant: *[not implemented yet]*
- Date: `datetime.date`, `datetime.datetime`
- Dimensions: *[not implemented yet]*
- Font: *[not implemented yet]*
- GPSCoordinate: `pyexiv2.utils.GPSCoordinate`
- Integer: int
- Locale: *[not implemented yet]*
- MIMEType: 2-tuple of strings
- Rational: `fractions.Fraction`
- Real: *[not implemented yet]*
- AgentName, ProperName, Text: unicode string
- Thumbnail: *[not implemented yet]*
- URI, URL: string
- XPath: *[not implemented yet]*

Attributes

description
The description of the tag.

key
The key of the tag in the dotted form familyName.groupName.tagName where familyName = xmp.

name
The name of the tag (this is also the third part of the key).

raw_value
The raw value of the tag as a [list of] string(s).

title
The title (label) of the tag.

type
The XMP type of the tag.

value
The value of the tag as a [list of] python object(s).

class pyexiv2.xmp.**XmpValueError**(*ValueError*)
Exception raised when failing to parse the value of an XMP tag.

Attributes

value
The value that fails to be parsed

type
The XMP type of the tag

1.6 pyexiv2.preview

class pyexiv2.preview.**Preview**

Instance Attributes

- *dimensions*
- *extension*
- *mime_type*
- *size*
- *data*

Instance Method

- *write_to_file(path)*

Description

The `Preview` define a preview image (properties and data buffer) embedded in image metadata.

Documentation

Instanciation

class pyexiv2.preview.**Preview**(*preview*)
A preview image embedded in image metadata.

Attributes

dimensions
A tuple containing the width and height of the preview image in pixels.

extension
The file extension of the preview image with a leading dot (e.g. .jpg).

mime_type
The mime type of the preview image (e.g. image/jpeg).

size
The size of the preview image in bytes.

data
The preview data as a Python list. The data can be send to an image library.

New in version 0.6.0

Example with Pillow:

```
>>> from PIL import Image
>>> import io
>>> from pyexiv2 import ImageMetadata, exif
>>> meta = ImageMetadata("lena.jpg")
>>> meta.read()
>>> # try with the first one
>>> preview = meta.previews[0]
>>> byteio = io.BytesIO(preview.data)
>>> img = Image.open(byteio)
>>> img.show()
```

Method

write_to_file(*path*)
Write the preview image to a file on disk. The file extension will be automatically appended to the path.

Argument:

- *path* str(path) The file path to write the preview to (without an extension)

CHAPTER 2

Tutorial

This tutorial is meant to give you a quick overview of what py3exiv2 allows you to do. You can just read it through or follow it interactively, in which case you will need to have py3exiv2 installed. It doesn't cover all the possibilities offered by py3exiv2, only a basic subset of them. For complete reference, see the *API documentation*.

Let's get started!

Remember, the lib is named py3exiv2 but the top-level module, for compatibility reasons, is named pyexiv2. So, we import the pyexiv2 module:

```
>>> import pyexiv2
```

We then load an image file and read its metadata:

```
>>> metadata = pyexiv2.ImageMetadata('test.jpg')
>>> metadata.read()
```

2.1 Reading and writing EXIF tags

Let's retrieve a list of all the available EXIF tags available in the image:

```
>>> metadata.exif_keys
['Exif.Image.ImageDescription',
 'Exif.Image.XResolution',
 'Exif.Image.YResolution',
 'Exif.Image.ResolutionUnit',
 'Exif.Image.Software',
 'Exif.Image.DateTime',
 'Exif.Image.Artist',
 'Exif.Image.Copyright',
 'Exif.Image.ExifTag',
 'Exif.Photo.Flash',
 'Exif.Photo.PixelXDimension',
 'Exif.Photo.PixelYDimension']
```

Each of those tags can be accessed with the `[]` operator on the metadata, much like a python dictionary:

```
>>> tag = metadata['Exif.Image.DateTime']
```

The value of an `ExifTag` object can be accessed in two different ways: with the *`raw_value`* and with the `value` attributes:

```
>>> tag.raw_value
'2004-07-13T21:23:44Z'

>>> tag.value
datetime.datetime(2004, 7, 13, 21, 23, 44)
```

The raw value is always a byte string, this is how the value is stored in the file. The value is lazily computed from the raw value depending on the EXIF type of the tag, and is represented as a convenient python object to allow easy manipulation.

Note that querying the value of a tag may raise an `ExifValueError` if the format of the raw value is invalid according to the EXIF specification (may happen if it was written by other software that implements the specification in a broken manner), or if pyexiv2 doesn't know how to convert it to a convenient python object.

Accessing the value of a tag as a python object allows easy manipulation and formatting:

```
>>> tag.value.strftime('%A %d %B %Y, %H:%M:%S')
'Tuesday 13 July 2004, 21:23:44'
```

Now let's modify the value of the tag and write it back to the file:

```
>>> import datetime
>>> tag.value = datetime.datetime.today()

>>> metadata.write()
```

Similarly to reading the value of a tag, one can set either the *`raw_value`* or the `value` (which will be automatically converted to a correctly formatted byte string by pyexiv2).

You can also add new tags to the metadata by providing a valid key and value pair (see exiv2's documentation for a list of valid EXIF tags):

```
>>> key = 'Exif.Photo.UserComment'
>>> value = 'This is a useful comment.'
>>> metadata[key] = pyexiv2.ExifTag(key, value)
```

As a handy shortcut, you can always assign a value for a given key regardless of whether it's already present in the metadata. If a tag was present, its value is overwritten. If the tag was not present, one is created and its value is set:

```
>>> metadata[key] = value
```

The EXIF data may optionally embed a thumbnail in the JPEG or TIFF format. The thumbnail can be accessed, set from a JPEG file or buffer, saved to disk and erased:

```
>>> thumb = metadata.exif_thumbnail
>>> thumb.set_from_file('/tmp/thumbnail.jpg')
>>> thumb.write_to_file('/tmp/copy')
>>> thumb.erase()
>>> metadata.write()
```

2.2 Reading and writing IPTC tags

Reading and writing IPTC tags works pretty much the same way as with EXIF tags. Let's retrieve the list of all available IPTC tags in the image:

```
>>> metadata.iptc_keys
['Iptc.Application2.Caption',
 'Iptc.Application2.Writer',
 'Iptc.Application2.Byline',
 'Iptc.Application2.ObjectName',
 'Iptc.Application2.DateCreated',
 'Iptc.Application2.City',
 'Iptc.Application2.ProvinceState',
 'Iptc.Application2.CountryName',
 'Iptc.Application2.Category',
 'Iptc.Application2.Keywords',
 'Iptc.Application2.Copyright']
```

Each of those tags can be accessed with the `[]` operator on the metadata:

```
>>> tag = metadata['Iptc.Application2.DateCreated']
```

An IPTC tag always has a list of values rather than a single value. This is because some tags have a repeatable character. Tags that are not repeatable only hold one value in their list of values.

Check the `repeatable` attribute to know whether a tag can hold more than one value:

```
>>> tag.repeatable
False
```

As with EXIF tags, the values of an `IptcTag` object can be accessed in two different ways: with the *`raw_value`* and with the `value` attributes:

```
>>> tag.raw_value
['2004-07-13']

>>> tag.value
[datetime.date(2004, 7, 13)]
```

Note that querying the values of a tag may raise an `IptcValueError` if the format of the raw values is invalid according to the IPTC specification (may happen if it was written by other software that implements the specification in a broken manner), or if pyexiv2 doesn't know how to convert it to a convenient python object.

Now let's modify the values of the tag and write it back to the file:

```
>>> tag.value = [datetime.date.today()]

>>> metadata.write()
```

Similarly to reading the values of a tag, one can set either the *`raw_value`* or the `value` (which will be automatically converted to correctly formatted byte strings by pyexiv2).

You can also add new tags to the metadata by providing a valid key and values pair (see exiv2's documentation for a list of valid IPTC tags):

```
>>> key = 'Iptc.Application2.Contact'
>>> values = ['John', 'Paul', 'Ringo', 'George']
>>> metadata[key] = pyexiv2.IptcTag(key, values)
```

As a handy shortcut, you can always assign values for a given key regardless of whether it's already present in the metadata. If a tag was present, its values are overwritten. If the tag was not present, one is created and its values are set:

```
>>> metadata[key] = values
```

The IPTC metadata in an image may embed an optional character set for its encoding. This is defined by the `Iptc.Envelope.CharacterSet` tag. The `ImageMetadata` class has an *`iptc_charset`* property that allows to easily get, set and delete this value:

```
>>> metadata.iptc_charset
'ascii'

>>> metadata.iptc_charset = 'utf-8'

>>> del metadata.iptc_charset
```

Note that at the moment, the only supported charset that can be assigned to the property is `utf-8`. Also note that even if the charset is not explicitly set, its value may be inferred from the contents of the image. If not, it will be `None`.

2.3 Reading and writing XMP tags

Reading and writing XMP tags works pretty much the same way as with EXIF tags. Let's retrieve the list of all available XMP tags in the image:

```
>>> metadata.xmp_keys
['Xmp.dc.creator',
 'Xmp.dc.description',
 'Xmp.dc.rights',
 'Xmp.dc.source',
 'Xmp.dc.subject',
 'Xmp.dc.title',
 'Xmp.xmp.CreateDate',
 'Xmp.xmp.ModifyDate']
```

Each of those tags can be accessed with the `[]` operator on the metadata:

```
>>> tag = metadata['Xmp.xmp.ModifyDate']
```

As with EXIF tags, the value of an `XmpTag` object can be accessed in two different ways: with the *`raw_value`* and with the `value` attributes:

```
>>> tag.raw_value
'2002-07-19T13:28:10'

>>> tag.value
datetime.datetime(2002, 7, 19, 13, 28, 10)
```

Note that querying the value of a tag may raise an `XmpValueError` if the format of the raw value is invalid according to the XMP specification (may happen if it was written by other software that implements the specification in a broken manner), or if pyexiv2 doesn't know how to convert it to a convenient python object.

Now let's modify the value of the tag and write it back to the file:

```
>>> tag.value = datetime.datetime.today()

>>> metadata.write()
```

Similarly to reading the value of a tag, one can set either the *raw_value* or the `value` (which will be automatically converted to a correctly formatted byte string by pyexiv2).

You can also add new tags to the metadata by providing a valid key and value pair (see exiv2's documentation for a list of valid XMP tags):

```
>>> key = 'Xmp.xmp.Label'
>>> value = 'A beautiful picture.'
>>> metadata[key] = pyexiv2.XmpTag(key, value)
```

As a handy shortcut, you can always assign a value for a given key regardless of whether it's already present in the metadata. If a tag was present, its value is overwritten. If the tag was not present, one is created and its value is set:

```
>>> metadata[key] = value
```

If you need to write custom metadata, you can register a custom XMP namespace:

```
>>> pyexiv2.xmp.register_namespace('http://example.org/foo/', 'foo')
>>> metadata['Xmp.foo.bar'] = 'baz'
```

Note that a limitation of the current implementation is that only simple text values can be written to tags in a custom namespace.

A custom namespace can be unregistered. This has the effect of invalidating all tags in this namespace for images that have not been written back yet:

```
>>> pyexiv2.xmp.unregister_namespace('http://example.org/foo/')
```

2.4 Accessing embedded previews

Images may embed previews (also called thumbnails) of various sizes in their metadata. pyexiv2 allows to easily access them:

```
>>> previews = metadata.previews

>>> len(previews)
2
```

They are sorted by increasing size. Let's play with the largest one:

```
>>> largest = previews[-1]

>>> largest.dimensions
(320, 240)

>>> largest.mime_type
'image/jpeg'

>>> largest.write_to_file('largest')
```

CHAPTER 3

Developers

If you are a developer and use py3exiv2 in your project, you will find here useful information.

3.1 Getting the code

py3exiv2's source code is versioned with bazaar, and the main development focus (sometimes referred to as *trunk*), is hosted on Launchpad.

To get a working copy of the latest revision of the development branch, just issue the following command in a terminal:

```
bzr branch lp:py3exiv2
```

3.2 Dependencies

To use py3exiv2:

- Python 3.2
- boost.python 1.46
- libexiv2 0.20

To build py3exiv2:

- *python-all-dev* (3.2)
- *libexiv2-dev* (0.20)
- *libboost-python-dev* (1.45)
- *g++*

Some unit tests have a dependency on python-tz. This dependency is optional: the corresponding tests will be skipped if it is not present on the system.

Additionally, if you want to cross-compile py3exiv2 for Windows and generate a Windows installer, you will need the following dependencies:

- MinGW
- 7-Zip
- NSIS

3.3 Building and installing

3.3.1 Linux

Open a terminal into the top-level directory (where is the file *configure.py*):

```
$ python3 configure.py
```

The configure script try to find the exact name of *libboost_python3* wich is depending on your environment. If it can't find the lib, give it the full path of this lib with the option *–libboost*. Example on Debian with Python-3.4:

```
$ python3 configure.py --libboost=/usr/lib/x86_64-linux-gnu/libboost_python-py34.so
```

Build the lib:

```
$ ./build.sh
```

The result of the build process is a shared library, `libexiv2python.so`, in the build directory:

```
$ ls build/
$ exiv2wrapper.os  exiv2wrapper_python.os  libexiv2python.so
```

And, if no error, install all the files:

```
$ ./build.sh -i
```

You will most likely need administrative privileges to the last step.

3.4 Documentation

The present documentation is generated using Sphinx from reStructuredText sources found in the doc/ directory. Invoke `make html` to (re)build the HTML documentation.

The index of the documentation will then be found under doc/_build/html/index.html.

3.5 Unit tests

py3exiv2's source comes with a battery of unit tests, in the test/ directory. To run them, invoke `python3 TestsRunner.py`.

3.6 Contributing

py3exiv2 is Free Software, meaning that you are encouraged to use it, modify it to suit your needs, contribute back improvements, and redistribute it.

Bugs are tracked on Launchpad. There is a team called py3exiv2-developers open to anyone interested in following development on py3exiv2. Don't hesitate to subscribe to the team (you don't need to actually contribute!) and to the associated mailing list.

There are several ways in which you can contribute to improve py3exiv2:

- Use it;
- Give your feedback and discuss issues and feature requests on the mailing list;
- Report bugs, write patches;
- Package it for your favorite distribution/OS.

When reporting a bug, don't forget to include the following information in the report:

- version of py3exiv2
- version of libexiv2 it was compiled against
- a minimal script that reliably reproduces the issue
- a sample image file with which the bug can reliably be reproduced

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

R

S

T

V

W

X