

Unittest for media

August 29, 2020

Contents

1	Test Information	3
1.1	Test Candidate Information	3
1.2	Unittest Information	3
1.3	Test System Information	3
2	Statistic	3
2.1	Test-Statistic for testrun with python 3.8.2 (final)	3
2.2	Coverage Statistic	4
3	Tested Requirements	5
3.1	Metadata	5
3.1.1	Method to get Metadata	5
3.2	Image	6
3.2.1	Load from File	6
3.2.2	Save	7
3.2.3	Image data	7
3.2.4	Resize	7
3.2.5	Rotate	8
3.2.6	Join	8
A	Trace for testrun with python 3.8.2 (final)	10
A.1	Tests with status Info (7)	10
A.1.1	Method to get Metadata	10
A.1.2	Load from File	15
A.1.3	Save	16
A.1.4	Image data	17
A.1.5	Resize	17
A.1.6	Rotate	18
A.1.7	Join	20

B Test-Coverage	22
B.1 media	22
B.1.1 media.__init__.py	22
B.1.2 media.common.py	26
B.1.3 media.convert.py	27
B.1.4 media.metadata.py	27

1 Test Information

1.1 Test Candidate Information

The Module `media` is designed to help on all issues with media files, like tags (e.g. `exif`, `id3`) and transformations. For more Information read the documentation.

Library Information	
Name	media
State	Released
Supported Interpreters	python3
Version	2f1c613a705625b8766637c1c130b6ab

Dependencies	
--------------	--

1.2 Unittest Information

Unittest Information	
Version	f6d7d5abd9b54bbc40f0db5d65c56893
Testruns with	python 3.8.2 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	Linux Mint 20 ulyana
Hostname	ahorn
Kernel	5.4.0-42-generic (#46-Ubuntu SMP Fri Jul 10 00:24:02 UTC 2020)
Machine	x86_64
Path	/user_data/data/dirk/prj/unittest/media/unittest
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun with python 3.8.2 (final)

Number of tests	7
Number of successfull tests	7
Number of possibly failed tests	0
Number of failed tests	0

Executionlevel	Full Test (all defined tests)
Time consumption	5.087s

2.2 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
media	97.3%	94.8%
media.__init__.py	99.3%	
media.common.py	100.0%	
media.convert.py	86.7%	
media.metadata.py	97.1%	

3 Tested Requirements

3.1 Metadata

3.1.1 Method to get Metadata

Description

A Method shall return the metadata for a given media filename.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.1!

Testrun:	python 3.8.2 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/_init_.py (27)
Start-Time:	2020-08-29 20:16:23,740
Finished-Time:	2020-08-29 20:16:24,269
Time-Consumption	0.529s

Testsummary:

- Success** Media data for unknown.txt is correct (Content None and Type is <class 'NoneType'>).
- Success** Media data for audio.mp3 is correct (Content {'duration': 236.094694, 'bitrate': 290743, 'artist': 'Kaleo', 'title': 'No Good', 'album': 'A/B', 'track': 1, 'genre': 'Rock', 'year': 2016, 'size': 8580366, 'time': 1451606398, 'tm_is_subst': True} and Type is <class 'dict'>).
- Success** Media data for audio_fail_conv.mp3 is correct (Content {'duration': 281.991837, 'bitrate': 228298, 'title': 'Video Games (Album Version Remastered)', 'artist': 'Lana Del Rey', 'album': 'Born To Die', 'genre': 'Pop', 'track': 4, 'year': 2012, 'size': 8047290, 'time': 1325375995, 'tm_is_subst': True} and Type is <class 'dict'>).
- Success** Media data for audio_year_0.mp3 is correct (Content {'duration': 120.476735, 'bitrate': 240202, 'title': 'Was bringt der Dezember', 'artist': 'Rolf und seine Freunde', 'album': 'Wir warten auf Weihnachten', 'year': 0, 'track': 9, 'genre': 'Other', 'size': 3617354} and Type is <class 'dict'>).
- Success** Media data for image_exif_gps.jpg is correct (Content {'time': 1560083621, 'exposure_program': 'Program Normal', 'exposure_time': 0.007633587786259542, 'flash': 'Off', 'aperture': 2.2, 'focal_length': 3.463, 'gps': {'lon': 11.574697, 'lat': 52.993599}, 'height': 3120, 'iso': 100, 'orientation': 6, 'width': 4160, 'size': 4524705, 'camera': 'motorola: motorola one'} and Type is <class 'dict'>).
- Success** Media data for image_exif_no_gps.jpg is correct (Content {'time': 1515143529, 'exposure_program': 'Program Normal', 'exposure_time': 0.03, 'flash': 'Fired', 'aperture': 2.2, 'focal_length': 4.5, 'height': 3968, 'iso': 160, 'orientation': 0, 'width': 2976, 'size': 2837285, 'camera': 'HUAWAI: EVA-L09'} and Type is <class 'dict'>).
- Success** Media data for image_non_exif.jpg is correct (Content {'size': 1139092, 'time': 1449870515, 'tm_is_subst': True} and Type is <class 'dict'>).
- Success** Media data for image_extraction_failed.jpg is correct (Content {'time': 1226149915, 'exposure_program': 'Program Normal', 'exposure_time': 0.008, 'flash': 'Fill Fired', 'aperture': 7.1, 'focal_length': 170.0, 'height': 2592, 'iso': 400, 'orientation': 1, 'width': 3888, 'size': 1301272, 'camera': 'Canon: Canon EOS 40D'} and Type is <class 'dict'>).
- Success** Media data for faulty_gps_data.jpg is correct (Content {'time': 1590940859, 'exposure_program': 'Program Normal', 'exposure_time': 0.01, 'flash': 'Off', 'aperture': 2.0, 'focal_length': 3.463, 'height': 3120, 'iso': 124, 'orientation': 6, 'width': 4160, 'size': 3500036, 'camera': 'motorola: motorola one'} and Type is <class 'dict'>).

Success Media data for video.3gp is correct (Content {'width': 800, 'height': 480, 'ratio': 1.6666666666666667, 'duration': 3.964, 'bitrate': 2341765, 'time': 1414948303, 'size': 1160345} and Type is <class 'dict'>).

Success Media data for video.mp4 is correct (Content {'width': 1920, 'height': 1080, 'ratio': 1.7777777777777777, 'duration': 12.453, 'bitrate': 17883888, 'time': 1503125482, 'size': 27838508} and Type is <class 'dict'>).

Success Media data for video_special_time.avi is correct (Content {'width': 320, 'height': 240, 'duration': 26.531264, 'bitrate': 840554, 'time': 1086778620, 'size': 2787622} and Type is <class 'dict'>).

Success Media data for video_no_date.avi is correct (Content {'width': 640, 'height': 480, 'ratio': 1.3333333333333333, 'duration': 11.016, 'bitrate': 2153411, 'size': 2965248, 'time': 1158528375, 'tm_is_subst': True} and Type is <class 'dict'>).

3.2 Image

The library `media` shall have a class `image`. This class shall be able to read from image or video files, pil image instances or from `media.image` instances itself. The class shall help by some common tasks like rotating, resizing, ...

3.2.1 Load from File

Description

The class `image` shall have a method `load_from_file`, which creates a copy of an image to the instance. Load from file can handle a filename, but also pil images and media images. The method returns `True` on success and `False` on failures.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.2!

Testrun:	python 3.8.2 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/_init....py (31)
Start-Time:	2020-08-29 20:16:24,270
Finished-Time:	2020-08-29 20:16:24,688
Time-Consumption	0.419s

Testsummary:

Success Type of image stored in instance, if no parameter is given is correct (Content <class 'NoneType'> and Type is <class 'type'>).

Success Type of image stored in instance, if a unsupported parameter is given is correct (Content <class 'NoneType'> and Type is <class 'type'>).

Success Type of image stored in instance, if an unknown file is given is correct (Content <class 'NoneType'> and Type is <class 'type'>).

Success Type of image stored in instance, if a image file is given is correct (Content <class 'PIL.Image.Image'> and Type is <class 'type'>).

Success Type of image stored in instance, if a video file is given is correct (Content <class 'PIL.Image.Image'> and Type is <class 'type'>).

3.2.2 Save

Description

The class `image` shall have a method `save`, which stores the modified image to a given filename. The method returns `True` on success and `False` on failures.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.3!

Testrun:	python 3.8.2 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/_init_.py (32)
Start-Time:	2020-08-29 20:16:24,688
Finished-Time:	2020-08-29 20:16:24,990
Time-Consumption	0.302s

Testsummary:

Success	Returnvalue of failed save method is correct (Content False and Type is <class 'bool'>).
Success	Existence of saved file is correct (Content False and Type is <class 'bool'>).
Success	Returnvalue of successful save method is correct (Content True and Type is <class 'bool'>).
Success	Existence of saved file is correct (Content True and Type is <class 'bool'>).

3.2.3 Image data

Description

The class `image` shall have a method `image_data`, which returns the raw data of the modified image.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.4!

Testrun:	python 3.8.2 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/_init_.py (33)
Start-Time:	2020-08-29 20:16:24,991
Finished-Time:	2020-08-29 20:16:25,179
Time-Consumption	0.189s

Testsummary:

Success	Filecompare for image_data.jpg is correct (Content True and Type is <class 'bool'>).
----------------	--

3.2.4 Resize

Description

The class `image` shall have a method `resize`, which resizes the image. The method returns `True` on success and `False` on failures.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.5!

Testrun:	python 3.8.2 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/_init_.py (34)
Start-Time:	2020-08-29 20:16:25,183
Finished-Time:	2020-08-29 20:16:25,320
Time-Consumption	0.137s

Testsummary:

- Success** Returnvalue of successful resize method is correct (Content True and Type is <class 'bool'>).
 - Success** Resolution of resized image is correct (Content 300 and Type is <class 'int'>).
 - Success** Returnvalue of failed resize method is correct (Content False and Type is <class 'bool'>).
-

3.2.5 Rotate

Description

The class `image` shall have a method `rotate_by_orientation`, which rotates the image by an exif orientation. If no parameter is given, the orientation will be taken out of the loaded image. The method returns `True` on success and `False` on failures.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.6!

Testrun:	python 3.8.2 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/_init_.py (35)
Start-Time:	2020-08-29 20:16:25,320
Finished-Time:	2020-08-29 20:16:26,494
Time-Consumption	1.173s

Testsummary:

- Success** Returnvalue of rotate method without loading an image is correct (Content False and Type is <class 'bool'>).
 - Success** Returnvalue of rotate method with invalid orientation is correct (Content False and Type is <class 'bool'>).
 - Success** Returnvalue of rotate method with no orientation in method call and exif is correct (Content False and Type is <class 'bool'>).
 - Success** Filecompare for rotated_image_none.jpg is correct (Content True and Type is <class 'bool'>).
 - Success** Filecompare for rotated_image_6.jpg is correct (Content True and Type is <class 'bool'>).
 - Success** Filecompare for rotated_image_8.jpg is correct (Content True and Type is <class 'bool'>).
 - Success** Filecompare for rotated_image_3.jpg is correct (Content True and Type is <class 'bool'>).
-

3.2.6 Join

Description

The class `image` shall have a method `join`, which joins an image to the loaded image. The method returns `True` on success and `False` on failures.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.7!

Testrun:	python 3.8.2 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/_init_.py (36)
Start-Time:	2020-08-29 20:16:26,496
Finished-Time:	2020-08-29 20:16:28,834
Time-Consumption	2.338s

Testsummary:

Success	Returnvalue of join method without loading an image is correct (Content False and Type is <class 'bool'>).
Success	Returnvalue of join method with invalid join position is correct (Content False and Type is <class 'bool'>).
Success	Returnvalue of join method with unknown join file is correct (Content False and Type is <class 'bool'>).
Success	Filecompare for joined_image_3.jpg is correct (Content True and Type is <class 'bool'>).
Success	Filecompare for joined_image_4.jpg is correct (Content True and Type is <class 'bool'>).
Success	Filecompare for joined_image_5.jpg is correct (Content True and Type is <class 'bool'>).
Success	Filecompare for joined_image_1.jpg is correct (Content True and Type is <class 'bool'>).
Success	Filecompare for joined_image_2.jpg is correct (Content True and Type is <class 'bool'>).

A Trace for testrun with python 3.8.2 (final)

A.1 Tests with status Info (7)

A.1.1 Method to get Metadata

Description

A Method shall return the metadata for a given media filename.

Testresult

This test was passed with the state: **Success**.

Success Media data for unknown.txt is correct (Content None and Type is <class 'NoneType'>).

Filetype not known: /user_data/data/dirk/prj/unittest/media/unittest/input_data/unknown.txt

Result (Media data for unknown.txt): None (<class 'NoneType'>)

Expectation (Media data for unknown.txt): result = None (<class 'NoneType'>)

Success Media data for audio.mp3 is correct (Content {'duration': 236.094694, 'bitrate': 290743, 'artist': 'Kaleo', 'title': 'No Good', 'album': 'A/B', 'track': 1, 'genre': 'Rock', 'year': 2016, 'size': 8580366, 'time': 1451606398, 'tm_is_subst': True} and Type is <class 'dict'>).

Result (Media data for audio.mp3): { 'duration': 236.094694, 'bitrate': 290743, 'artist':
 ↳ 'Kaleo', 'title': 'No Good', 'album': 'A/B', 'track': 1, 'genre': 'Rock', 'year': 2016,
 ↳ 'size': 8580366, 'time': 1451606398, 'tm_is_subst': True } (<class 'dict'>)

Expectation (Media data for audio.mp3): result = { 'duration': 236.094694, 'bitrate': 290743,
 ↳ 'artist': 'Kaleo', 'title': 'No Good', 'album': 'A/B', 'track': 1, 'genre': 'Rock',
 ↳ 'year': 2016, 'time': 1451606398, 'tm_is_subst': True, 'size': 8580366 } (<class 'dict'>)

Success Media data for audio_fail_conv.mp3 is correct (Content {'duration': 281.991837, 'bitrate': 228298, 'title': 'Video Games (Album Version Remastered)', 'artist': 'Lana Del Rey', 'album': 'Born To Die', 'genre': 'Pop', 'track': 4, 'year': 2012, 'size': 8047290, 'time': 1325375995, 'tm_is_subst': True} and Type is <class 'dict'>).

Result (Media data for audio_fail_conv.mp3): { 'duration': 281.991837, 'bitrate': 228298,
 ↳ 'title': 'Video Games (Album Version Remastered)', 'artist': 'Lana Del Rey', 'album':
 ↳ 'Born To Die', 'genre': 'Pop', 'track': 4, 'year': 2012, 'size': 8047290, 'time':
 ↳ 1325375995, 'tm_is_subst': True } (<class 'dict'>)

Expectation (Media data for audio_fail_conv.mp3): result = { 'duration': 281.991837,
 ↳ 'bitrate': 228298, 'artist': 'Lana Del Rey', 'title': 'Video Games (Album Version
 ↳ Remastered)', 'album': 'Born To Die', 'track': 4, 'genre': 'Pop', 'year': 2012, 'time':
 ↳ 1325375995, 'tm_is_subst': True, 'size': 8047290 } (<class 'dict'>)

Success Media data for audio_year_0.mp3 is correct (Content {'duration': 120.476735, 'bitrate': 240202, 'title': 'Was bringt der Dezember', 'artist': 'Rolf und seine Freunde', 'album': 'Wir warten auf Weihnachten', 'year': 0, 'track': 9, 'genre': 'Other', 'size': 3617354} and Type is <class 'dict'>).

```
Result (Media data for audio_year_0.mp3): { 'duration': 120.476735, 'bitrate': 240202,
↳ 'title': 'Was bringt der Dezember', 'artist': 'Rolf und seine Freunde', 'album': 'Wir
↳ warten auf Weihnachten', 'year': 0, 'track': 9, 'genre': 'Other', 'size': 3617354 }
↳ (<class 'dict'>)
```

```
Expectation (Media data for audio_year_0.mp3): result = { 'duration': 120.476735, 'bitrate':
↳ 240202, 'artist': 'Rolf und seine Freunde', 'title': 'Was bringt der Dezember', 'album':
↳ 'Wir warten auf Weihnachten', 'track': 9, 'genre': 'Other', 'year': 0, 'size': 3617354 }
↳ (<class 'dict'>)
```

Success Media data for image_exif_gps.jpg is correct (Content {'time': 1560083621, 'exposure_program': 'Program Normal', 'exposure_time': 0.007633587786259542, 'flash': 'Off', 'aperture': 2.2, 'focal_length': 3.463, 'gps': {'lon': 11.574697, 'lat': 52.993599}, 'height': 3120, 'iso': 100, 'orientation': 6, 'width': 4160, 'size': 4524705, 'camera': 'motorola: motorola one'} and Type is <class 'dict'>).

Converting time out of '2019:06:09 14:33:41'

Converting exposure_program out of 2

Converting exposure_time out of (1, 131)

Converting flash out of 16

Converting aperture out of (22, 10)

Converting focal_length out of (3463, 1000)

Converting gps out of {0: b'\x02\x02\x00\x00', 1: 'N', 2: ((52, 1), (59, 1), (369564, 10000)), 3: 'E', 4: ((11, 1), (34, 1), (289092, 10000)), 5: b'\x01', 6: (49610, 1000), 7: ((12, 1), (33, 1), (39, 1)), 18: 'WGS-84', 27: 'ASCII\x00\x00\x00GPS', 29: '2019:06:09'}

Converting height out of 3120

Converting iso out of 100

Converting camera_vendor out of 'motorola'

Converting camera_model out of 'motorola one'

Converting orientation out of 6

Converting width out of 4160

```
Result (Media data for image_exif_gps.jpg): { 'time': 1560083621, 'exposure_program':
↳ 'Program Normal', 'exposure_time': 0.007633587786259542, 'flash': 'Off', 'aperture': 2.2,
↳ 'focal_length': 3.463, 'gps': { 'lon': 11.574697, 'lat': 52.993599 }, 'height': 3120,
↳ 'iso': 100, 'orientation': 6, 'width': 4160, 'size': 4524705, 'camera': 'motorola:
↳ motorola one' } (<class 'dict'>)
```

```
Expectation (Media data for image_exif_gps.jpg): result = { 'time': 1560083621,
↳ 'exposure_program': 'Program Normal', 'exposure_time': 0.007633587786259542, 'flash':
↳ 'Off', 'aperture': 2.2, 'focal_length': 3.463, 'gps': { 'lon': 11.574697, 'lat':
↳ 52.993599 }, 'height': 3120, 'iso': 100, 'orientation': 6, 'width': 4160, 'camera':
↳ 'motorola: motorola one', 'size': 4524705 } (<class 'dict'>)
```

Success Media data for image_exif_no_gps.jpg is correct (Content {'time': 1515143529, 'exposure_program': 'Program Normal', 'exposure_time': 0.03, 'flash': 'Fired', 'aperture': 2.2, 'focal_length': 4.5, 'height': 3968, 'iso': 160, 'orientation': 0, 'width': 2976, 'size': 2837285, 'camera': 'HUAWEI: EVA-L09'} and Type is <class 'dict'>).

Converting time out of '2018:01:05 10:12:09'

Converting exposure_program out of 2

Converting exposure_time out of (30000000, 1000000000)

Converting flash out of 1

Converting aperture out of (220, 100)

Converting focal_length out of (4500, 1000)

Converting height out of 3968

Converting iso out of 160

Converting camera_vendor out of 'HUAWEI'

Converting camera_model out of 'EVA-L09'

Converting orientation out of 0

Converting width out of 2976

Result (Media data for image_exif_no_gps.jpg): { 'time': 1515143529, 'exposure_program':
 ↪ 'Program Normal', 'exposure_time': 0.03, 'flash': 'Fired', 'aperture': 2.2,
 ↪ 'focal_length': 4.5, 'height': 3968, 'iso': 160, 'orientation': 0, 'width': 2976, 'size':
 ↪ 2837285, 'camera': 'HUAWEI: EVA-L09' } (<class 'dict'>)

Expectation (Media data for image_exif_no_gps.jpg): result = { 'time': 1515143529,
 ↪ 'exposure_program': 'Program Normal', 'exposure_time': 0.03, 'flash': 'Fired',
 ↪ 'aperture': 2.2, 'focal_length': 4.5, 'height': 3968, 'iso': 160, 'orientation': 0,
 ↪ 'width': 2976, 'camera': 'HUAWEI: EVA-L09', 'size': 2837285 } (<class 'dict'>)

Success Media data for image_non_exif.jpg is correct (Content {'size': 1139092, 'time': 1449870515, 'tm_is_subst': True} and Type is <class 'dict'>).

/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_non_exif.jpg does not have
 ↪ any exif information

Result (Media data for image_non_exif.jpg): { 'size': 1139092, 'time': 1449870515,
 ↪ 'tm_is_subst': True } (<class 'dict'>)

Expectation (Media data for image_non_exif.jpg): result = { 'time': 1449870515,
 ↪ 'tm_is_subst': True, 'size': 1139092 } (<class 'dict'>)

Success Media data for image_extraction_failed.jpg is correct (Content {'time': 1226149915, 'exposure_program': 'Program Normal', 'exposure_time': 0.008, 'flash': 'Fill Fired', 'aperture': 7.1, 'focal_length': 170.0, 'height': 2592, 'iso': 400, 'orientation': 1, 'width': 3888, 'size': 1301272, 'camera': 'Canon: Canon EOS 40D'} and Type is <class 'dict'>).

Converting time out of '2008:11:08 14:11:55'

Converting exposure_program out of 2

Converting exposure_time out of (1, 125)

Converting flash out of 9

Converting aperture out of (71, 10)

Converting focal_length out of (170, 1)

Converting gps out of {0: b'\x02\x02\x00\x00'}

GPS data extraction failed for {0: b'\x02\x02\x00\x00'}

Converting height out of 2592

Converting iso out of 400

Converting camera_vendor out of 'Canon'

Converting camera_model out of 'Canon EOS 40D'

Converting orientation out of 1

Converting width out of 3888

Result (Media data for image_extraction_failed.jpg): { 'time': 1226149915,
↪ 'exposure_program': 'Program Normal', 'exposure_time': 0.008, 'flash': 'Fill Fired',
↪ 'aperture': 7.1, 'focal_length': 170.0, 'height': 2592, 'iso': 400, 'orientation': 1,
↪ 'width': 3888, 'size': 1301272, 'camera': 'Canon: Canon EOS 40D' } (<class 'dict'>)

Expectation (Media data for image_extraction_failed.jpg): result = { 'time': 1226149915,
↪ 'exposure_program': 'Program Normal', 'exposure_time': 0.008, 'flash': 'Fill Fired',
↪ 'aperture': 7.1, 'focal_length': 170.0, 'height': 2592, 'iso': 400, 'orientation': 1,
↪ 'width': 3888, 'camera': 'Canon: Canon EOS 40D', 'size': 1301272 } (<class 'dict'>)

Success Media data for faulty_gps_data.jpg is correct (Content {'time': 1590940859, 'exposure_program': 'Program Normal', 'exposure_time': 0.01, 'flash': 'Off', 'aperture': 2.0, 'focal_length': 3.463, 'height': 3120, 'iso': 124, 'orientation': 6, 'width': 4160, 'size': 3500036, 'camera': 'motorola: motorola one'} and Type is <class 'dict'>).

```

Converting time out of '2020:05:31 18:00:59'
Converting exposure_program out of 2
Converting exposure_time out of (1, 100)
Converting flash out of 16
Converting aperture out of (20, 10)
Converting focal_length out of (3463, 1000)
Converting gps out of {0: b'\x00\x00\x00\x00', 1: '\x00', 2: ((0, 0), (0, 0), (0, 0)), 3:
↪ '\x00', 4: ((0, 0), (0, 0), (0, 0)), 5: b'\x00', 6: (0, 0), 7: ((0, 0), (0, 0), (0, 0)),
↪ 18: '\x00\x00\x00\x00\x00\x00', 27: '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00', 29:
↪ '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'}
Converting height out of 3120
Converting iso out of 124
Converting camera_vendor out of 'motorola'
Converting camera_model out of 'motorola one'
Converting orientation out of 6
Converting width out of 4160
Result (Media data for faulty_gps_data.jpg): { 'time': 1590940859, 'exposure_program':
↪ 'Program Normal', 'exposure_time': 0.01, 'flash': 'Off', 'aperture': 2.0, 'focal_length':
↪ 3.463, 'height': 3120, 'iso': 124, 'orientation': 6, 'width': 4160, 'size': 3500036,
↪ 'camera': 'motorola: motorola one' } (<class 'dict'>)
Expectation (Media data for faulty_gps_data.jpg): result = { 'time': 1590940859,
↪ 'exposure_program': 'Program Normal', 'exposure_time': 0.01, 'flash': 'Off', 'aperture':
↪ 2.0, 'focal_length': 3.463, 'height': 3120, 'iso': 124, 'orientation': 6, 'width': 4160,
↪ 'camera': 'motorola: motorola one', 'size': 3500036 } (<class 'dict'>)

```

Success Media data for video.3gp is correct (Content {'width': 800, 'height': 480, 'ratio': 1.6666666666666667, 'duration': 3.964, 'bitrate': 2341765, 'time': 1414948303, 'size': 1160345} and Type is <class 'dict'>).

```

Result (Media data for video.3gp): { 'width': 800, 'height': 480, 'ratio':
↪ 1.6666666666666667, 'duration': 3.964, 'bitrate': 2341765, 'time': 1414948303, 'size':
↪ 1160345 } (<class 'dict'>)
Expectation (Media data for video.3gp): result = { 'width': 800, 'height': 480, 'ratio':
↪ 1.6666666666666667, 'duration': 3.964, 'bitrate': 2341765, 'time': 1414948303, 'size':
↪ 1160345 } (<class 'dict'>)

```

Success Media data for video.mp4 is correct (Content {'width': 1920, 'height': 1080, 'ratio': 1.7777777777777777, 'duration': 12.453, 'bitrate': 17883888, 'time': 1503125482, 'size': 27838508} and Type is <class 'dict'>).

```

Result (Media data for video.mp4): { 'width': 1920, 'height': 1080, 'ratio':
↪ 1.7777777777777777, 'duration': 12.453, 'bitrate': 17883888, 'time': 1503125482, 'size':
↪ 27838508 } (<class 'dict'>)
Expectation (Media data for video.mp4): result = { 'width': 1920, 'height': 1080, 'ratio':
↪ 1.7777777777777777, 'duration': 12.453, 'bitrate': 17883888, 'time': 1503125482, 'size':
↪ 27838508 } (<class 'dict'>)

```

Success Media data for video.special_time.avi is correct (Content {'width': 320, 'height': 240, 'duration': 26.531264, 'bitrate': 840554, 'time': 1086778620, 'size': 2787622} and Type is <class 'dict'>).

Can't convert 'N/A' (ratio) for ratio

Can't convert 'N/A' (duration) for duration

Result (Media data for video_special_time.avi): { 'width': 320, 'height': 240, 'duration': 26.531264, 'bitrate': 840554, 'time': 1086778620, 'size': 2787622 } (<class 'dict'>)

Expectation (Media data for video_special_time.avi): result = { 'width': 320, 'height': 240, 'duration': 26.531264, 'bitrate': 840554, 'time': 1086778620, 'size': 2787622 } (<class 'dict'>)

Success Media data for video_no_date.avi is correct (Content {'width': 640, 'height': 480, 'ratio': 1.3333333333333333, 'duration': 11.016, 'bitrate': 2153411, 'size': 2965248, 'time': 1158528375, 'tm_is_subst': True} and Type is <class 'dict'>).

Result (Media data for video_no_date.avi): { 'width': 640, 'height': 480, 'ratio': 1.3333333333333333, 'duration': 11.016, 'bitrate': 2153411, 'size': 2965248, 'time': 1158528375, 'tm_is_subst': True } (<class 'dict'>)

Expectation (Media data for video_no_date.avi): result = { 'width': 640, 'height': 480, 'ratio': 1.3333333333333333, 'duration': 11.016, 'bitrate': 2153411, 'time': 1158528375, 'tm_is_subst': True, 'size': 2965248 } (<class 'dict'>)

A.1.2 Load from File

Description

The class `Image` shall have a method `load_from_file`, which creates a copy of an image to the instance. Load from file can handle a filename, but also pil images and media images. The method returns `True` on success and `False` on failures.

Testresult

This test was passed with the state: **Success**.

Success Type of image stored in instance, if no parameter is given is correct (Content <class 'NoneType'> and Type is <class 'type'>).

Result (Type of image stored in instance, if no parameter is given): <class 'NoneType'>
↳ (<class 'type'>)

Expectation (Type of image stored in instance, if no parameter is given): result = <class 'NoneType'> (<class 'type'>)

Success Type of image stored in instance, if a unsupported parameter is given is correct (Content <class 'NoneType'> and Type is <class 'type'>).

Instance type is not supported: <class 'int'>

Result (Type of image stored in instance, if a unsupported parameter is given): <class 'NoneType'> (<class 'type'>)

Expectation (Type of image stored in instance, if a unsupported parameter is given): result = <class 'NoneType'> (<class 'type'>)

Success Type of image stored in instance, if an unknown file is given is correct (Content <class 'NoneType'> and Type is <class 'type'>).

Filetype is not supported

↪ (/user_data/data/dirk/prj/unittest/media/unittest/input_data/unknown.txt)

Result (Type of image stored in instance, if an unknown file is given): <class 'NoneType'>

↪ (<class 'type'>)

Expectation (Type of image stored in instance, if an unknown file is given): result = <class

↪ 'NoneType'> (<class 'type'>)

Success Type of image stored in instance, if a image file is given is correct (Content <class 'PIL.Image.Image'> and Type is <class 'type'>).

loading image from

↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'

Result (Type of image stored in instance, if a image file is given): <class

↪ 'PIL.Image.Image'> (<class 'type'>)

Expectation (Type of image stored in instance, if a image file is given): result = <class

↪ 'PIL.Image.Image'> (<class 'type'>)

Success Type of image stored in instance, if a video file is given is correct (Content <class 'PIL.Image.Image'> and Type is <class 'type'>).

loading image from '/user_data/data/dirk/prj/unittest/media/unittest/input_data/video.mp4'

Result (Type of image stored in instance, if a video file is given): <class

↪ 'PIL.Image.Image'> (<class 'type'>)

Expectation (Type of image stored in instance, if a video file is given): result = <class

↪ 'PIL.Image.Image'> (<class 'type'>)

A.1.3 Save

Description

The class image shall have a method save, which stores the modified image to a given filename. The method returns True on success and False on failures.

Testresult

This test was passed with the state: **Success**.

Success Returnvalue of failed save method is correct (Content False and Type is <class 'bool'>).

No image available to be saved

↪ ('/user_data/data/dirk/prj/unittest/media/unittest/output_data/saved_image.jpg')

Result (Returnvalue of failed save method): False (<class 'bool'>)

Expectation (Returnvalue of failed save method): result = False (<class 'bool'>)

Success Existence of saved file is correct (Content False and Type is <class 'bool'>).

```
Result (Existance of saved file): False (<class 'bool'>)
```

```
Expectation (Existance of saved file): result = False (<class 'bool'>)
```

Success Returnvalue of successful save method is correct (Content True and Type is <class 'bool'>).

```
loading image from '/user_data/data/dirk/prj/unittest/media/unittest/input_data/video.mp4'
```

```
Saving image to '/user_data/data/dirk/prj/unittest/media/unittest/output_data/saved_image.jpg'
```

```
Result (Returnvalue of successful save method): True (<class 'bool'>)
```

```
Expectation (Returnvalue of successful save method): result = True (<class 'bool'>)
```

Success Existance of saved file is correct (Content True and Type is <class 'bool'>).

```
Result (Existance of saved file): True (<class 'bool'>)
```

```
Expectation (Existance of saved file): result = True (<class 'bool'>)
```

A.1.4 Image data

Description

The class image shall have a method image_data, which returns the raw data of the modified image.

Testresult

This test was passed with the state: **Success**.

Success Filecompare for image_data.jpg is correct (Content True and Type is <class 'bool'>).

```
loading image from
```

```
↳ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_no_gps.jpg'
```

```
Result (Filecompare for image_data.jpg): True (<class 'bool'>)
```

```
Expectation (Filecompare for image_data.jpg): result = True (<class 'bool'>)
```

A.1.5 Resize

Description

The class image shall have a method resize, which resizes the image. The method returns True on success and False on failures.

Testresult

This test was passed with the state: **Success**.

Success Returnvalue of successful resize method is correct (Content True and Type is <class 'bool'>).

```
loading image from
↳ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
Resizing picture to max 300 pixel in whatever direction
Result (Returnvalue of successful resize method): True (<class 'bool'>)
Expectation (Returnvalue of successful resize method): result = True (<class 'bool'>)
```

Success Resolution of resized image is correct (Content 300 and Type is <class 'int'>).

```
Saving image to
↳ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/resized_image.jpg'
Result (Resolution of resized image): 300 (<class 'int'>)
Expectation (Resolution of resized image): result = 300 (<class 'int'>)
```

Success Returnvalue of failed resize method is correct (Content False and Type is <class 'bool'>).

```
No image available to be resized
Result (Returnvalue of failed resize method): False (<class 'bool'>)
Expectation (Returnvalue of failed resize method): result = False (<class 'bool'>)
```

A.1.6 Rotate

Description

The class `image` shall have a method `rotate_by_orientation`, which rotates the image by an exif orientation. If no parameter is given, the orientation will be taken out of the loaded image. The method returns `True` on success and `False` on failures.

Testresult

This test was passed with the state: **Success**.

Success Returnvalue of rotate method without loading an image is correct (Content False and Type is <class 'bool'>).

```
No image available, rotation not possible
Result (Returnvalue of rotate method without loading an image): False (<class 'bool'>)
Expectation (Returnvalue of rotate method without loading an image): result = False (<class 'bool'>)
↳ 'bool'>)
```

Success Returnvalue of rotate method with invalid orientation is correct (Content False and Type is <class 'bool'>).

```
loading image from
↳ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
Orientation 17 unknown for rotation
Result (Returnvalue of rotate method with invalid orientation): False (<class 'bool'>)
Expectation (Returnvalue of rotate method with invalid orientation): result = False (<class 'bool'>)
↳ 'bool'>)
```

Success Returnvalue of rotate method with no orientation in method call and exif is correct (Content False and Type is <class 'bool'>).

```
loading image from
↳ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_non_exif.jpg'
Result (Returnvalue of rotate method with no orientation in method call and exif): False
↳ (<class 'bool'>)
Expectation (Returnvalue of rotate method with no orientation in method call and exif):
↳ result = False (<class 'bool'>)
```

Success Filecompare for rotated_image_none.jpg is correct (Content True and Type is <class 'bool'>).

```
Rotate with orientation None
loading image from
↳ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
No orientation given, orientation 6 extract from exif data
Rotating picture by 270 (deg)
Saving image to
↳ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/rotated_image_none.jpg'
Result (Filecompare for rotated_image_none.jpg): True (<class 'bool'>)
Expectation (Filecompare for rotated_image_none.jpg): result = True (<class 'bool'>)
```

Success Filecompare for rotated_image_6.jpg is correct (Content True and Type is <class 'bool'>).

```
Rotate with orientation 6
loading image from
↳ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
Rotating picture by 270 (deg)
Saving image to
↳ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/rotated_image_6.jpg'
Result (Filecompare for rotated_image_6.jpg): True (<class 'bool'>)
Expectation (Filecompare for rotated_image_6.jpg): result = True (<class 'bool'>)
```

Success Filecompare for rotated_image_8.jpg is correct (Content True and Type is <class 'bool'>).

```
Rotate with orientation 8
loading image from
↳ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
Rotating picture by 90 (deg)
Saving image to
↳ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/rotated_image_8.jpg'
Result (Filecompare for rotated_image_8.jpg): True (<class 'bool'>)
Expectation (Filecompare for rotated_image_8.jpg): result = True (<class 'bool'>)
```

Success Filecompare for rotated_image_3.jpg is correct (Content True and Type is <class 'bool'>).

```

Rotate with orientation 3
loading image from
↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
Rotating picture by 180 (deg)
Saving image to
↪ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/rotated_image_3.jpg'
Result (Filecompare for rotated_image_3.jpg): True (<class 'bool'>)
Expectation (Filecompare for rotated_image_3.jpg): result = True (<class 'bool'>)
    
```

A.1.7 Join

Description

The class image shall have a method join, which joins an image to the loaded image. The method returns True on success and False on failures.

Testresult

This test was passed with the state: **Success**.

Success Returnvalue of join method without loading an image is correct (Content False and Type is <class 'bool'>).

No image available, joining not possible

```

Result (Returnvalue of join method without loading an image): False (<class 'bool'>)
Expectation (Returnvalue of join method without loading an image): result = False (<class
↪ 'bool'>)
    
```

Success Returnvalue of join method with invalid join position is correct (Content False and Type is <class 'bool'>).

```

loading image from
↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
Join position value 17 is not supported
Result (Returnvalue of join method with invalid join position): False (<class 'bool'>)
Expectation (Returnvalue of join method with invalid join position): result = False (<class
↪ 'bool'>)
    
```

Success Returnvalue of join method with unknown join file is correct (Content False and Type is <class 'bool'>).

```

loading image from
↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
Instance type is not supported: <class 'int'>
Image to be joined is not supported None
Result (Returnvalue of join method with unknown join file): False (<class 'bool'>)
Expectation (Returnvalue of join method with unknown join file): result = False (<class
↪ 'bool'>)
    
```

Success Filecompare for joined_image_3.jpg is correct (Content True and Type is <class 'bool'>).

Join with position 3

loading image from

↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'

loading image from

↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_no_gps.jpg'

Resizing picture to max 300 pixel in whatever direction

Joining two images

Saving image to

↪ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/joined_image_3.jpg'

Result (Filecompare for joined_image_3.jpg): True (<class 'bool'>)

Expectation (Filecompare for joined_image_3.jpg): result = True (<class 'bool'>)

Success Filecompare for joined_image_4.jpg is correct (Content True and Type is <class 'bool'>).

Join with position 4

loading image from

↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'

loading image from

↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_no_gps.jpg'

Resizing picture to max 300 pixel in whatever direction

Joining two images

Saving image to

↪ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/joined_image_4.jpg'

Result (Filecompare for joined_image_4.jpg): True (<class 'bool'>)

Expectation (Filecompare for joined_image_4.jpg): result = True (<class 'bool'>)

Success Filecompare for joined_image_5.jpg is correct (Content True and Type is <class 'bool'>).

Join with position 5

loading image from

↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'

loading image from

↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_no_gps.jpg'

Resizing picture to max 300 pixel in whatever direction

Joining two images

Saving image to

↪ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/joined_image_5.jpg'

Result (Filecompare for joined_image_5.jpg): True (<class 'bool'>)

Expectation (Filecompare for joined_image_5.jpg): result = True (<class 'bool'>)

Success Filecompare for joined_image_1.jpg is correct (Content True and Type is <class 'bool'>).

```

Join with position 1
loading image from
↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
loading image from
↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_no_gps.jpg'
Resizing picture to max 300 pixel in whatever direction
Joining two images
Saving image to
↪ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/joined_image_1.jpg'
Result (Filecompare for joined_image_1.jpg): True (<class 'bool'>)
Expectation (Filecompare for joined_image_1.jpg): result = True (<class 'bool'>)

```

Success Filecompare for joined_image_2.jpg is correct (Content True and Type is <class 'bool'>).

```

Join with position 2
loading image from
↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_gps.jpg'
loading image from
↪ '/user_data/data/dirk/prj/unittest/media/unittest/input_data/image_exif_no_gps.jpg'
Resizing picture to max 300 pixel in whatever direction
Joining two images
Saving image to
↪ '/user_data/data/dirk/prj/unittest/media/unittest/output_data/joined_image_2.jpg'
Result (Filecompare for joined_image_2.jpg): True (<class 'bool'>)
Expectation (Filecompare for joined_image_2.jpg): result = True (<class 'bool'>)

```

B Test-Coverage

B.1 media

The line coverage for media was 97.3%

The branch coverage for media was 94.8%

B.1.1 media.__init__.py

The line coverage for media.__init__.py was 99.3%

The branch coverage for media.__init__.py was 94.8%

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3 #
4 """

```

```

5 media (Media Tools)
6 =====
7
8 **Author:**
9
10 * Dirk Alders <sudo-dirk@mount-mockery.de>
11
12 **Description:**
13
14     This module helps on all issues with media files , like tags (e.g. exif, id3) and
15     transformations.
16
17 **Submodules:**
18
19 * :func:`media.get_media_data`
20 * :class:`media.image`
21
22 **Unittest:**
23
24     See also the :download:`unittest <../../media/_testresults_/unittest.pdf>` documentation.
25 """
26
27 __DEPENDENCIES__ = []
28
29 import io
30 import logging
31 from PIL import Image, ImageEnhance, ExifTags
32
33 logger_name = 'MEDIA'
34 logger = logging.getLogger(logger_name)
35
36 __DESCRIPTION__ = """The Module {\\tt %s} is designed to help on all issues with media files ,
37     like tags (e.g. exif, id3) and transformations.
38 For more Information read the documentation.""" % __name__.replace('-', '\\-')
39 """The Module Description"""
40 __INTERPRETER__ = (3, )
41 """The Tested Interpreter-Versions"""
42
43
44 KEY_ALBUM = 'album'
45 KEY_APERTURE = 'aperture'
46 KEY_ARTIST = 'artist'
47 KEY_BITRATE = 'bitrate'
48 KEY_CAMERA = 'camera'
49 KEY_DURATION = 'duration'
50 KEY_EXPOSURE_PROGRAM = 'exposure_program'
51 KEY_EXPOSURE_TIME = 'exposure_time'
52 KEY_FLASH = 'flash'
53 KEY_FOCAL_LENGTH = 'focal_length'
54 KEY_GENRE = 'genre'
55 KEY_GPS = 'gps'
56 KEY_HEIGHT = 'height'
57 KEY_ISO = 'iso'
58 KEY_ORIENTATION = 'orientation'
59 KEY_RATIO = 'ratio'
60 KEY_SIZE = 'size'
61 KEY_TIME = 'time' # USE time.localtime(value) or datetime.fromtimestamp(value) to convert the
62     timestamp
63 KEY_TIME.IS.SUBSTITUTION = 'tm_is_subst'
64 KEY_TITLE = 'title'
65 KEY_TRACK = 'track'
66 KEY_WIDTH = 'width'
67 KEY_YEAR = 'year'

```

Unittest for media

```
65
66
67 def get_media_data(full_path):
68     from media.metadata import get_audio_data, get_image_data, get_video_data
69     from media.common import get_filetype, FILETYPE_AUDIO, FILETYPE_IMAGE, FILETYPE_VIDEO
70     #
71     ft = get_filetype(full_path)
72     #
73     if ft == FILETYPE_AUDIO:
74         return get_audio_data(full_path)
75     elif ft == FILETYPE_IMAGE:
76         return get_image_data(full_path)
77     elif ft == FILETYPE_VIDEO:
78         return get_video_data(full_path)
79     else:
80         logger.warning('Filetype not known: %s', full_path)
81
82
83 ORIENTATION_NORMAL = 1
84 ORIENTATION_VERTICAL_MIRRORED = 2
85 ORIENTATION_HALF_ROTATED = 3
86 ORIENTATION_HORIZONTAL_MIRRORED = 4
87 ORIENTATION_LEFT_ROTATED = 6
88 ORIENTATION_RIGHT_ROTATED = 8
89
90 JOIN_TOP_LEFT = 1
91 JOIN_TOP_RIGHT = 2
92 JOIN_BOT_LEFT = 3
93 JOIN_BOT_RIGHT = 4
94 JOIN_CENTER = 5
95
96
97 class image(object):
98     def __init__(self, media_instance=None):
99         if media_instance is not None:
100             self.load_from_file(media_instance)
101         else:
102             self._im = None
103
104     def load_from_file(self, media_instance):
105         from media.convert import get_pil_image
106         #
107         self._im = get_pil_image(media_instance)
108         if self._im is None:
109             return False
110         try:
111             self._exif = dict(self._im._getexif().items())
112         except AttributeError:
113             self._exif = {}
114         if type(self._im) is not Image.Image:
115             self._im = self._im.copy()
116         logger.debug('loading image from %s', repr(media_instance))
117         return True
118
119     def save(self, full_path):
120         if self._im is None:
121             logger.warning('No image available to be saved (%s)', repr(full_path))
122             return False
123         else:
124             logger.debug('Saving image to %s', repr(full_path))
125             with open(full_path, 'w') as fh:
126                 im = self._im.convert('RGB')
127                 im.save(fh, 'JPEG')
128         return True
```

```

129
130 def image_data(self):
131     im = self._im.copy().convert('RGB')
132     output = io.BytesIO()
133     im.save(output, format='JPEG')
134     return output.getvalue()
135
136 def resize(self, max_size):
137     if self._im is None:
138         logger.warning('No image available to be resized')
139         return False
140     else:
141         logger.debug('Resizing picture to max %d pixel in whatever direction', max_size)
142         x, y = self._im.size
143         xy_max = max(x, y)
144         self._im = self._im.resize((int(x * float(max_size) / xy_max), int(y * float(max_size) / xy_max)), Image.NEAREST).rotate(0)
145         return True
146
147 def rotate_by_orientation(self, orientation=None):
148     if self._im is None:
149         logger.warning('No image available, rotation not possible')
150         return False
151
152     if orientation is None:
153         exif_tags = dict((v, k) for k, v in ExifTags.TAGS.items())
154         try:
155             orientation = self._exif[exif_tags['Orientation']]
156             logger.debug("No orientation given, orientation %s extract from exif data", repr(orientation))
157         except KeyError:
158             return False
159
160     if orientation == ORIENTATION_HALF_ROTATED:
161         angle = 180
162     elif orientation == ORIENTATION_LEFT_ROTATED:
163         angle = 270
164     elif orientation == ORIENTATION_RIGHT_ROTATED:
165         angle = 90
166     else:
167         if type(orientation) == int and orientation > 8:
168             logger.warning('Orientation %s unknown for rotation', repr(orientation))
169             return False
170         logger.debug('Rotating picture by %d (deg)', angle)
171         self._im = self._im.rotate(angle, expand=True)
172         return True
173
174 def join(self, join_image, join_pos=JOIN_TOP_RIGHT, opacity=0.7):
175     from media.convert import get_pil_image
176
177     def rgba_copy(im):
178         if im.mode != 'RGBA':
179             return im.convert('RGBA')
180         else:
181             return im.copy()
182
183     if self._im is None:
184         logger.warning('No image available, joining not possible')
185         return False
186
187     # ensure type of join_image is PIL.Image

```

```

188     join_image = get_pil_image(join_image)
189     if join_image is None:
190         logger.warning('Image to be joined is not supported %s', repr(join_image))
191         return False
192
193     im2 = rgba_copy(join_image)
194     # change opacity of im2
195     alpha = im2.split()[3]
196     alpha = ImageEnhance.Brightness(alpha).enhance(opacity)
197     im2.putalpha(alpha)
198
199     self._im = rgba_copy(self._im)
200
201     # create a transparent layer
202     layer = Image.new('RGBA', self._im.size, (0, 0, 0, 0))
203     # draw im2 in layer
204     if join_pos == JOIN_TOP_LEFT:
205         layer.paste(im2, (0, 0))
206     elif join_pos == JOIN_TOP_RIGHT:
207         layer.paste(im2, ((self._im.size[0] - im2.size[0]), 0))
208     elif join_pos == JOIN_BOT_LEFT:
209         layer.paste(im2, (0, (self._im.size[1] - im2.size[1])))
210     elif join_pos == JOIN_BOT_RIGHT:
211         layer.paste(im2, ((self._im.size[0] - im2.size[0]), (self._im.size[1] - im2.size[1])))
212     )
213     elif join_pos == JOIN_CENTER:
214         layer.paste(im2, (int((self._im.size[0] - im2.size[0]) / 2), int((self._im.size[1] -
215         im2.size[1]) / 2)))
216     else:
217         logger.warning("Join position value %s is not supported", join_pos)
218         return False
219
220     logger.debug('Joining two images')
221     self._im = Image.composite(layer, self._im, layer)
222
223     return True

```

B.1.2 media.common.py

The line coverage for media.common.py was 100.0%

The branch coverage for media.common.py was 94.8%

```

1 import os
2
3 FILETYPE_AUDIO = 'audio'
4 FILETYPE_IMAGE = 'image'
5 FILETYPE_VIDEO = 'video'
6
7 EXTENTIONS_AUDIO = ['.mp3', ]
8 EXTENTIONS_IMAGE = ['.jpg', '.jpeg', '.jpe', '.png', '.tif', '.tiff', '.gif', ]
9 EXTENTIONS_VIDEO = ['.avi', '.mpg', '.mpeg', '.mpe', '.mov', '.qt', '.mp4', '.webm', '.ogv', '.
10     flv', '.3gp', ]
11
12 def get_filetype(full_path):
13     ext = os.path.splitext(full_path.lower())[1]
14     if ext in EXTENTIONS_AUDIO:
15         return FILETYPE_AUDIO
16     elif ext in EXTENTIONS_IMAGE:
17         return FILETYPE_IMAGE
18     elif ext in EXTENTIONS_VIDEO:
19         return FILETYPE_VIDEO

```

B.1.3 media.convert.py

The line coverage for media.convert.py was 86.7%

The branch coverage for media.convert.py was 94.8%

```

1 import io
2 from media import common, logger
3 from PIL import Image
4 import subprocess
5 import platform
6
7
8 def get_pil_image(media_instance):
9     try:
10         media_instance = media_instance._im
11     except AttributeError:
12         pass
13
14     #
15     if type(media_instance) == str:
16         ft = common.get_filetype(media_instance)
17         if ft == common.FILETYPE_IMAGE:
18             return Image.open(media_instance)
19         elif ft == common.FILETYPE_VIDEO:
20             if platform.system() == 'Linux':
21                 cmd = 'ffmpeg -ss 0.5 -i "' + media_instance + '" -vframes 1 -f image2pipe pipe:1
22                 2> /dev/null '
23             else:
24                 cmd = 'ffmpeg -ss 0.5 -i "' + media_instance + '" -vframes 1 -f image2pipe pipe:1
25                 2> NULL '
26             try:
27                 data = subprocess.check_output(cmd, shell=True)
28             except subprocess.CalledProcessError:
29                 logger.warning('ffmpeg seems to be not installed')
30                 return None
31             ffmpeg_handle = io.BytesIO(data)
32             im = Image.open(ffmpeg_handle)
33             return im.copy()
34             logger.warning('Filetype is not supported (%s)', media_instance)
35         elif type(media_instance) == Image.Image:
36             return media_instance.copy()
37         else:
38             logger.warning('Instance type is not supported: %s' % type(media_instance))

```

B.1.4 media.metadata.py

The line coverage for media.metadata.py was 97.1%

The branch coverage for media.metadata.py was 94.8%

```

1 import logging
2 import os
3 from PIL import Image
4 import math
5 import media
6 import subprocess
7 import time
8
9
10 logger = media.logger
11
12

```

```

13 __KEY_CAMERA_VENDOR__ = 'camera_vendor'
14 __KEY_CAMERA_MODEL__ = 'camera_model'
15
16
17 def get_audio_data(full_path):
18     conv_key_dict = {}
19     conv_key_dict['album'] = (str, media.KEY_ALBUM)
20     conv_key_dict['TAG:album'] = (str, media.KEY_ALBUM)
21     conv_key_dict['TAG:artist'] = (str, media.KEY_ARTIST)
22     conv_key_dict['artist'] = (str, media.KEY_ARTIST)
23     conv_key_dict['bit_rate'] = (__int_conv__, media.KEY_BITRATE)
24     conv_key_dict['duration'] = (float, media.KEY_DURATION)
25     conv_key_dict['TAG:genre'] = (str, media.KEY_GENRE)
26     conv_key_dict['genre'] = (str, media.KEY_GENRE)
27     conv_key_dict['TAG:title'] = (str, media.KEY_TITLE)
28     conv_key_dict['title'] = (str, media.KEY_TITLE)
29     conv_key_dict['TAG:track'] = (__int_conv__, media.KEY_TRACK)
30     conv_key_dict['track'] = (__int_conv__, media.KEY_TRACK)
31     conv_key_dict['TAG:date'] = (__int_conv__, media.KEY_YEAR)
32     conv_key_dict['date'] = (__int_conv__, media.KEY_YEAR)
33     return __adapt__data__(__get_xprobe_data__(full_path, conv_key_dict), full_path)
34
35
36 def get_video_data(full_path):
37     conv_key_dict = {}
38     conv_key_dict['creation_time'] = (__vid_datetime_conv__, media.KEY_TIME)
39     conv_key_dict['TAG:creation_time'] = (__vid_datetime_conv__, media.KEY_TIME)
40     conv_key_dict['bit_rate'] = (__int_conv__, media.KEY_BITRATE)
41     conv_key_dict['duration'] = (float, media.KEY_DURATION)
42     conv_key_dict['height'] = (__int_conv__, media.KEY_HEIGHT)
43     conv_key_dict['width'] = (__int_conv__, media.KEY_WIDTH)
44     conv_key_dict['display_aspect_ratio'] = (__ratio_conv__, media.KEY_RATIO)
45     return __adapt__data__(__get_xprobe_data__(full_path, conv_key_dict), full_path)
46
47
48 def get_image_data(full_path):
49     return __adapt__data__(__get_exif_data__(full_path), full_path)
50
51
52 def __adapt__data__(data, full_path):
53     data[media.KEY_SIZE] = os.path.getsize(full_path)
54     # Join Camera Vendor and Camera Model
55     if __KEY_CAMERA_MODEL__ in data and __KEY_CAMERA_VENDOR__ in data:
56         model = data.pop(__KEY_CAMERA_MODEL__)
57         vendor = data.pop(__KEY_CAMERA_VENDOR__)
58         data[media.KEY_CAMERA] = '%s: %s' % (vendor, model)
59     # Add time if not exists
60     if media.KEY_TIME not in data:
61         if media.KEY_YEAR in data and media.KEY_TRACK in data:
62             if data[media.KEY_YEAR] != 0: # ignore year 0 - must be wrong
63                 # Use a date where track 1 is the newest in the given year
64                 minute = int(data[media.KEY_TRACK] / 60)
65                 second = (data[media.KEY_TRACK] - 60 * minute) % 60
66                 #
67                 data[media.KEY_TIME] = int(time.mktime((data[media.KEY_YEAR], 1, 1, 0, 59 -
68                 minute, 59 - second, 0, 0, 0)))
69                 data[media.KEY_TIME_IS_SUBSTITUTION] = True
70         else:
71             data[media.KEY_TIME] = int(os.path.getmtime(full_path))
72             data[media.KEY_TIME_IS_SUBSTITUTION] = True
73     return data

```

```

73
74
75 def __get_xxprobe_data__(full_path, conv_key_dict):
76     def _ffprobe_command(full_path):
77         return ['ffprobe', '-v', 'quiet', '-show_format', '-show_streams', full_path]
78
79     def _avprobe_command(full_path):
80         return ['avprobe', '-v', 'quiet', '-show_format', '-show_streams', full_path]
81
82     try:
83         xxprobe_text = subprocess.check_output(_avprobe_command(full_path))
84     except FileNotFoundError:
85         try:
86             xxprobe_text = subprocess.check_output(_ffprobe_command(full_path))
87         except FileNotFoundError:
88             logger.warning('ffprobe and avprobe seem to be not installed')
89             return {}
90
91     #
92     rv = {}
93     for line in xxprobe_text.decode('utf-8').splitlines():
94         try:
95             key, val = [snippet.strip() for snippet in line.split('=')]
96         except ValueError:
97             continue
98         else:
99             if key in conv_key_dict:
100                 tp, name = conv_key_dict[key]
101                 try:
102                     rv[name] = tp(val)
103                 except ValueError:
104                     logger.log(logging.WARNING if val else logging.INFO, 'Can\'t convert %s (%s)
105                     for %s', repr(val), name, name)
106             return rv
107
108 def __get_exif_data__(full_path):
109     rv = {}
110     im = Image.open(full_path)
111     try:
112         exif = dict(im._getexif().items())
113     except AttributeError:
114         logger.debug('%s does not have any exif information', full_path)
115     else:
116         conv_key_dict = {}
117         # IMAGE
118         conv_key_dict[0x9003] = (__datetime_conv__, media.KEY_TIME)
119         conv_key_dict[0x8822] = (__exposure_program_conv__, media.KEY_EXPOSURE_PROGRAM)
120         conv_key_dict[0x829A] = (__num_denum_conv__, media.KEY_EXPOSURE_TIME)
121         conv_key_dict[0x9209] = (__flash_conv__, media.KEY_FLASH)
122         conv_key_dict[0x829D] = (__num_denum_conv__, media.KEY_APERTURE)
123         conv_key_dict[0x920A] = (__num_denum_conv__, media.KEY_FOCAL_LENGTH)
124         conv_key_dict[0x8825] = (__gps_conv__, media.KEY_GPS)
125         conv_key_dict[0xA003] = (__int_conv__, media.KEY_HEIGHT)
126         conv_key_dict[0x8827] = (__int_conv__, media.KEY_ISO)
127         conv_key_dict[0x010F] = (str, __KEY_CAMERA_VENDOR__)
128         conv_key_dict[0x0110] = (str, __KEY_CAMERA_MODEL__)
129         conv_key_dict[0x0112] = (__int_conv__, media.KEY_ORIENTATION)
130         conv_key_dict[0xA002] = (__int_conv__, media.KEY_WIDTH)
131     for key in conv_key_dict:
132         if key in exif:
133             tp, name = conv_key_dict[key]
134             raw_value = exif[key]

```

Unittest for media

```

134         logger.debug(" Converting %s out of %s", name, repr(raw_value))
135         value = tp(raw_value)
136         if value is not None:
137             rv[name] = value
138     return rv
139
140
141 # TODO: Join datetime converter __datetime_conv__ and __vid_datetime_conv__
142 def __datetime_conv__(dt):
143     format_string = "%Y:%m:%d %H:%M:%S"
144     return int(time.mktime(time.strptime(dt, format_string)))
145
146
147 def __vid_datetime_conv__(dt):
148     try:
149         dt = dt[:dt.index('.')]
150     except ValueError:
151         pass # time string seems to have no '.'
152     dt = dt.replace('T', ' ').replace('/', ' ').replace('\\', '\\')
153     if len(dt) == 16:
154         dt += ':00'
155     format_string = '%Y-%m-%d %H:%M:%S'
156     return int(time.mktime(time.strptime(dt, format_string)))
157
158
159 def __exposure_program_conv__(n):
160     return {
161         0: 'Unidentified',
162         1: 'Manual',
163         2: 'Program Normal',
164         3: 'Aperture Priority',
165         4: 'Shutter Priority',
166         5: 'Program Creative',
167         6: 'Program Action',
168         7: 'Portrait Mode',
169         8: 'Landscape Mode'
170     }.get(n, None)
171
172
173 def __flash_conv__(n):
174     return {
175         0: 'No',
176         1: 'Fired',
177         5: 'Fired (?)', # no return sensed
178         7: 'Fired (!)', # return sensed
179         9: 'Fill Fired',
180         13: 'Fill Fired (?)',
181         15: 'Fill Fired (!)',
182         16: 'Off',
183         24: 'Auto Off',
184         25: 'Auto Fired',
185         29: 'Auto Fired (?)',
186         31: 'Auto Fired (!)',
187         32: 'Not Available'
188     }.get(n, None)
189
190
191 def __int_conv__(value):
192     try:
193         return int(value)
194     except ValueError:

```

```

195     for c in ['. ', '/', '-']:
196         p = value.find(c)
197         if p >= 0:
198             value = value[:p]
199     try:
200         return int(value)
201     except ValueError:
202         return None
203
204
205 def __num_denum_conv__(data):
206     try:
207         return float(data)
208     except TypeError:
209         num, denum = data
210         return num / denum
211
212
213 def __gps_conv__(data):
214     def lat_lon_cal(lon_or_lat):
215         lon_lat = 0.
216         fac = 1.
217         for data in lon_or_lat:
218             try:
219                 lon_lat += float(data[0]) / float(data[1]) * fac
220             except TypeError:
221                 lon_lat += data * fac
222             except ZeroDivisionError:
223                 return 0.
224             fac *= 1. / 60.
225         if math.isnan(lon_lat):
226             return 0.
227         return lon_lat
228     try:
229         lon = lat_lon_cal(data[0x0004])
230         lat = lat_lon_cal(data[0x0002])
231         if lon != 0 or lat != 0: # do not use lon and lat equal 0, caused by motorola gps
232             return {'lon': lon, 'lat': lat}
233     except KeyError:
234         logger.warning('GPS data extraction failed for %s', repr(data))
235
236
237 def __ratio_conv__(ratio):
238     ratio = ratio.replace('\\', '')
239     num, denum = ratio.split(':')
240     return float(num) / float(denum)

```