Unittest for `media`

February 1, 2020

Contents

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	d4bf62e70e70b47431f7471f25637ecf
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	1bd602a73ceaead82d56c196c3e6a973
Testruns with	python 3.6.9 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	LinuxMint 19.3 tricia
Hostname	ahorn
Kernel	5.3.0-28-generic (#30 18.04.1-Ubuntu SMP Fri Jan 17 06:14:09 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 3.6.9 (final)

Number of tests	**6**
Number of successfull tests	**6**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	4.128s

2.2 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
media	97.6%	96.4%
media.__init__.py	99.2%	
media.common.py	100.0%	
media.convert.py	86.7%	
media.metadata.py	98.1%	

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 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (26)
Start-Time:	2020-02-01 20:09:30,073
Finished-Time:	2020-02-01 20:09:30,583
Time-Consumption	0.510s

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': 1518783213, 'exposure_program': 'Program Normal', 'exposure_time': 0.000535, 'flash': 'Auto Off', 'aperture': 2.2, 'focal_length': 4.5, 'gps': {'lon': 12.140646934444444, 'lat': 53.68635940527778}, 'height': 2240, 'iso': 50, 'orientation': 0, 'width': 3968, 'size': 4342955, 'camera': 'HUAWEI: EVA-L09'} 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': 'HUAWEI: 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 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, 'ratio': 0.0, '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'>).

4 Testcases with no corresponding Requirement

4.1 Summary for testrun with python 3.6.9 (final)

4.1.1 Initialisation

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (30)
Start-Time:	2020-02-01 20:09:30,583
Finished-Time:	2020-02-01 20:09:30,980
Time-Consumption	0.397s

Testsummary:

Success	Type of image stored in instance is correct (Content <class 'NoneType'> and Type is <class 'type'>).
Success	Type of image stored in instance is correct (Content <class 'NoneType'> and Type is <class 'type'>).
Success	Type of image stored in instance is correct (Content <class 'NoneType'> and Type is <class 'type'>).
Success	Type of image stored in instance is correct (Content <class 'PIL.Image.Image'> and Type is <class 'type'>).
Success	Type of image stored in instance is correct (Content <class 'PIL.Image.Image'> and Type is <class 'type'>).

4.1.2 Join

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (34)
Start-Time:	2020-02-01 20:09:32,061
Finished-Time:	2020-02-01 20:09:34,202
Time-Consumption	2.140s

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

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

4.1.3 Resize

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (32)
Start-Time:	2020-02-01 20:09:31,275
Finished-Time:	2020-02-01 20:09:31,383
Time-Consumption	0.108s
Testsummary:	
Success	Returnvalue of successful resize method is correct (Content True and Type is <class 'bool'>).
Success	Resulution 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'>).

4.1.4 Rotate

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (33)
Start-Time:	2020-02-01 20:09:31,383
Finished-Time:	2020-02-01 20:09:32,061
Time-Consumption	0.678s
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	Filecompare is correct (Content True and Type is <class 'bool'>).
Success	Filecompare is correct (Content True and Type is <class 'bool'>).
Success	Filecompare is correct (Content True and Type is <class 'bool'>).

4.1.5 Save

Testresult

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

Testrun:	python 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (31)
Start-Time:	2020-02-01 20:09:30,981
Finished-Time:	2020-02-01 20:09:31,275

Time-Consumption	0.294s
Testsummary:	
Success	Returnvalue of failed save method is correct (Content False and Type is <class 'bool'>).
Success	Existance 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	Existance of saved file is correct (Content True and Type is <class 'bool'>).

A Trace for testrun with python 3.6.9 (final)

A.1 Tests with status Info (6)

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'>).

```
Can't convert 'N/A' (bitrate) for bitrate
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': 1518783213, 'exposure_program': 'Program Normal', 'exposure_time': 0.000535, 'flash': 'Auto Off', 'aperture': 2.2, 'focal_length': 4.5, 'gps': {'lon': 12.140646934444444, 'lat': 53.68635940527778}, 'height': 2240, 'iso': 50, 'orientation': 0, 'width': 3968, 'size': 4342955, 'camera': 'HUAWEI: EVA-L09'} and Type is <class 'dict'>).

```
Result (Media data for image_exif_gps.jpg): { 'time': 1518783213, 'exposure_program':
↪  'Program Normal', 'exposure_time': 0.000535, 'flash': 'Auto Off', 'aperture': 2.2,
↪  'focal_length': 4.5, 'gps': { 'lon': 12.140646934444444, 'lat': 53.68635940527778 },
↪  'height': 2240, 'iso': 50, 'orientation': 0, 'width': 3968, 'size': 4342955, 'camera':
↪  'HUAWEI: EVA-L09' } (<class 'dict'>)
```

```
Expectation (Media data for image_exif_gps.jpg): result = { 'time': 1518783213,
↪  'exposure_program': 'Program Normal', 'exposure_time': 0.000535, 'flash': 'Auto Off',
↪  'aperture': 2.2, 'focal_length': 4.5, 'gps': { 'lon': 12.140646934444444, 'lat':
↪  53.68635940527778 }, 'height': 2240, 'iso': 50, 'orientation': 0, 'width': 3968,
↪  'camera': 'HUAWEI: EVA-L09', 'size': 4342955 } (<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'>).

```
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'>).

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

```
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 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, 'ratio': 0.0, 'duration': 26.531264, 'bitrate': 840554, 'time': 1086778620, 'size': 2787622} and Type is <class 'dict'>).

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

```
Result (Media data for video_special_time.avi): { 'width': 320, 'height': 240, 'ratio': 0.0,
↪  'duration': 26.531264, 'bitrate': 840554, 'time': 1086778620, 'size': 2787622 } (<class
↪  'dict'>)
```

```
Expectation (Media data for video_special_time.avi): result = { 'width': 320, 'height': 240,
↪  'ratio': 0.0, '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 Initialisation

Testresult

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

Success	Type of image stored in instance is correct (Content <class 'NoneType'> and Type is <class 'type'>).

```
Result (Type of image stored in instance): <class 'NoneType'> (<class 'type'>)
```

```
Expectation (Type of image stored in instance): result = <class 'NoneType'> (<class 'type'>)
```

Success	Type of image stored in instance is correct (Content <class 'NoneType'> and Type is <class 'type'>).

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

```
Result (Type of image stored in instance): <class 'NoneType'> (<class 'type'>)
```

```
Expectation (Type of image stored in instance): result = <class 'NoneType'> (<class 'type'>)
```

Success	Type of image stored in instance 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): <class 'NoneType'> (<class 'type'>)
```

```
Expectation (Type of image stored in instance): result = <class 'NoneType'> (<class 'type'>)
```

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

```
Result (Type of image stored in instance): <class 'PIL.Image.Image'> (<class 'type'>)
Expectation (Type of image stored in instance): result = <class 'PIL.Image.Image'> (<class
↪  'type'>)
```

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

```
Result (Type of image stored in instance): <class 'PIL.Image.Image'> (<class 'type'>)
Expectation (Type of image stored in instance): result = <class 'PIL.Image.Image'> (<class
↪  'type'>)
```

A.1.3 Save

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	Existance 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'>).

```
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 Resize

Testresult

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

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

```
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	Resulution 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 (Resulution of resized image): 300 (<class 'int'>)
Expectation (Resulution 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.1.5 Rotate

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'>)
```

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

```
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'>)
```

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

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

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

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

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

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

A.1.6 Join

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'>).

```
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'>).

```
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 is correct (Content True and Type is <class 'bool'>).

```
Join with position 3
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): True (<class 'bool'>)
Expectation (Filecompare): result = True (<class 'bool'>)
```

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

```
Join with position 4
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): True (<class 'bool'>)
Expectation (Filecompare): result = True (<class 'bool'>)
```

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

```
Join with position 5
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): True (<class 'bool'>)
Expectation (Filecompare): result = True (<class 'bool'>)
```

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

```
Join with position 1
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): True (<class 'bool'>)
Expectation (Filecompare): result = True (<class 'bool'>)
```

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

```
Join with position 2
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): True (<class 'bool'>)
Expectation (Filecompare): result = True (<class 'bool'>)
```

B Test-Coverage

B.1 media

The line coverage for media was 97.6%
The branch coverage for media was 96.4%

B.1.1 media.__init__.py

The line coverage for media.__init__.py was 99.2%
The branch coverage for media.__init__.py was 96.4%

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
"""
media (Media Tools)
===================

**Author:**

* Dirk Alders <sudo-dirk@mount-mockery.de>

**Description:**

    This module helps on all issues with media files, like tags (e.g. exif, id3) and
    transformations.

**Submodules:**

* :func:`media.get_media_data`
* :class:`media.image`

**Unittest:**

    See also the :download:`unittest <../../media/_testresults_/unittest.pdf>` documentation.
"""
__DEPENDENCIES__ = []

import logging
from PIL import Image, ImageEnhance

logger_name = 'MEDIA'
logger = logging.getLogger(logger_name)


__DESCRIPTION__ = """The Module {\\tt %s} is designed to help on all issues with media files,
    like tags (e.g. exif, id3) and transformations.
For more Information read the documentation.""" % __name__.replace('_', '\_')
"""The Module Description"""
__INTERPRETER__ = (3, )
"""The Tested Interpreter-Versions"""


```

```
41 KEY_ALBUM = 'album'
42 KEY_APERTURE = 'aperture'
43 KEY_ARTIST = 'artist'
44 KEY_BITRATE = 'bitrate'
45 KEY_CAMERA = 'camera'
46 KEY_DURATION = 'duration'
47 KEY_EXPOSURE_PROGRAM = 'exposure_program'
48 KEY_EXPOSURE_TIME = 'exposure_time'
49 KEY_FLASH = 'flash'
50 KEY_FOCAL_LENGTH = 'focal_length'
51 KEY_GENRE = 'genre'
52 KEY_GPS = 'gps'
53 KEY_HEIGHT = 'height'
54 KEY_ISO = 'iso'
55 KEY_ORIENTATION = 'orientation'
56 KEY_RATIO = 'ratio'
57 KEY_SIZE = 'size'
58 KEY_TIME = 'time'   # USE time.localtime(value) or datetime.fromtimestamp(value) to convert the
       timestamp
59 KEY_TIME_IS_SUBSTITUTION = 'tm_is_subst'
60 KEY_TITLE = 'title'
61 KEY_TRACK = 'track'
62 KEY_WIDTH = 'width'
63 KEY_YEAR = 'year'
64
65
66 def get_media_data(full_path):
67     from media.metadata import get_audio_data, get_image_data, get_video_data
68     from media.common import get_filetype, FILETYPE_AUDIO, FILETYPE_IMAGE, FILETYPE_VIDEO
69     #
70     ft = get_filetype(full_path)
71     #
72     if ft == FILETYPE_AUDIO:
73         return get_audio_data(full_path)
74     elif ft == FILETYPE_IMAGE:
75         return get_image_data(full_path)
76     elif ft == FILETYPE_VIDEO:
77         return get_video_data(full_path)
78     else:
79         logger.warning('Filetype not known: %s', full_path)
80
81
82 ORIENTATION_NORMAL = 1
83 ORIENTATION_VERTICAL_MIRRORED = 2
84 ORIENTATION_HALF_ROTATED = 3
85 ORIENTATION_HORIZONTAL_MIRRORED = 4
86 ORIENTATION_LEFT_ROTATED = 6
87 ORIENTATION_RIGHT_ROTATED = 8
88
89 JOIN_TOP_LEFT = 1
90 JOIN_TOP_RIGHT = 2
91 JOIN_BOT_LEFT = 3
92 JOIN_BOT_RIGHT = 4
93 JOIN_CENTER = 5
94
95
96 class image(object):
97     def __init__(self, media_instance=None):
98         if media_instance is not None:
99             self.load_from_file(media_instance)
100         else:
101             self._im = None
```

```
102
103     def load_from_file(self, media_instance):
104         from media.convert import get_pil_image
105         #
106         self._im = get_pil_image(media_instance)
107         if self._im is None:
108             return False
109         return True
110
111     def save(self, full_path):
112         if self._im is None:
113             logger.warning('No image available to be saved (%s)', repr(full_path))
114             return False
115         else:
116             logger.debug('Saving image to %s', repr(full_path))
117             with open(full_path, 'w') as fh:
118                 im = self._im.convert('RGB')
119                 im.save(fh, 'JPEG')
120         return True
121
122     def resize(self, max_size):
123         if self._im is None:
124             logger.warning('No image available to be resized')
125             return False
126         else:
127             logger.debug('Resizing picture to max %d pixel in whatever direction', max_size)
128             x, y = self._im.size
129             xy_max = max(x, y)
130             self._im = self._im.resize((int(x * float(max_size) / xy_max), int(y * float(max_size
    ) / xy_max)), Image.NEAREST).rotate(0)
131         return True
132
133     def rotate_by_orientation(self, orientation):
134         if self._im is None:
135             logger.warning('No image available, rotation not possible')
136             return False
137
138         if orientation == ORIENTATION_HALF_ROTATED:
139             angle = 180
140         elif orientation == ORIENTATION_LEFT_ROTATED:
141             angle = 270
142         elif orientation == ORIENTATION_RIGHT_ROTATED:
143             angle = 90
144         else:
145             logger.warning('Orientation %s unknown for rotation', repr(orientation))
146             return False
147         logger.debug('Rotating picture by %d  ', angle)
148         self._im = self._im.rotate(angle, expand=True)
149         return True
150
151     def join(self, join_image, join_pos=JOIN_TOP_RIGHT, opacity=0.7):
152         from media.convert import get_pil_image
153
154         def rgba_copy(im):
155             if im.mode != 'RGBA':
156                 return im.convert('RGBA')
157             else:
158                 return im.copy()
159
160         if self._im is None:
161             logger.warning('No image available, joining not possible')
162             return False
```

```
163
164         # ensure type of join_image is PIL.Image
165         join_image = get_pil_image(join_image)
166         if join_image is None:
167             logger.warning('Image to be joined is not supported %s', repr(join_image))
168             return False
169
170         im2 = rgba_copy(join_image)
171         # change opacity of im2
172         alpha = im2.split()[3]
173         alpha = ImageEnhance.Brightness(alpha).enhance(opacity)
174         im2.putalpha(alpha)
175
176         self._im = rgba_copy(self._im)
177
178         # create a transparent layer
179         layer = Image.new('RGBA', self._im.size, (0, 0, 0, 0))
180         # draw im2 in layer
181         if join_pos == JOIN_TOP_LEFT:
182             layer.paste(im2, (0, 0))
183         elif join_pos == JOIN_TOP_RIGHT:
184             layer.paste(im2, ((self._im.size[0] - im2.size[0]), 0))
185         elif join_pos == JOIN_BOT_LEFT:
186             layer.paste(im2, (0, (self._im.size[1] - im2.size[1])))
187         elif join_pos == JOIN_BOT_RIGHT:
188             layer.paste(im2, ((self._im.size[0] - im2.size[0]), (self._im.size[1] - im2.size[1]))
    )
189         elif join_pos == JOIN_CENTER:
190             layer.paste(im2, (int((self._im.size[0] - im2.size[0]) / 2), int((self._im.size[1] -
    im2.size[1]) / 2)))
191         else:
192             logger.warning("Join position value %s is not supported", join_pos)
193             return False
194
195         logger.debug('Joining two images')
196         self._im = Image.composite(layer, self._im, layer)
197
198         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 96.4%

```
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', '.
       flv', '.3gp', ]
10
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 96.4%

```
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     if type(media_instance) == str:
15         ft = common.get_filetype(media_instance)
16         if ft == common.FILETYPE_IMAGE:
17             return Image.open(media_instance).copy()
18         elif ft == common.FILETYPE_VIDEO:
19             if platform.system() == 'Linux':
20                 cmd = 'ffmpeg -ss 0.5 -i "' + media_instance + '" -vframes 1 -f image2pipe pipe:1
       2> /dev/null'
21             else:
22                 cmd = 'ffmpeg -ss 0.5 -i "' + media_instance + '" -vframes 1 -f image2pipe pipe:1
       2> NULL'
23             try:
24                 data = subprocess.check_output(cmd, shell=True)
25             except subprocess.CalledProcessError:
26                 logger.warning('ffmpeg seems to be not installed')
27                 return None
28             ffmpeg_handle = io.BytesIO(data)
29             im = Image.open(ffmpeg_handle)
30             return im.copy()
31         logger.warning('Filetype is not supported (%s)', media_instance)
32     elif type(media_instance) == Image.Image:
33         return media_instance.copy()
34     else:
35         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 98.1%
The branch coverage for media.metadata.py was 96.4%

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

```
__KEY_CAMERA_VENDOR__ = 'camera_vendor'
__KEY_CAMERA_MODEL__ = 'camera_model'


def get_audio_data(full_path):
    conv_key_dict = {}
    conv_key_dict['album'] = (str, media.KEY_ALBUM)
    conv_key_dict['TAG:album'] = (str, media.KEY_ALBUM)
    conv_key_dict['TAG:artist'] = (str, media.KEY_ARTIST)
    conv_key_dict['artist'] = (str, media.KEY_ARTIST)
    conv_key_dict['bit_rate'] = (__int_conv__, media.KEY_BITRATE)
    conv_key_dict['duration'] = (float, media.KEY_DURATION)
    conv_key_dict['TAG:genre'] = (str, media.KEY_GENRE)
    conv_key_dict['genre'] = (str, media.KEY_GENRE)
    conv_key_dict['TAG:title'] = (str, media.KEY_TITLE)
    conv_key_dict['title'] = (str, media.KEY_TITLE)
    conv_key_dict['TAG:track'] = (__int_conv__, media.KEY_TRACK)
    conv_key_dict['track'] = (__int_conv__, media.KEY_TRACK)
    conv_key_dict['TAG:date'] = (__int_conv__, media.KEY_YEAR)
    conv_key_dict['date'] = (__int_conv__, media.KEY_YEAR)
    return __adapt__data__(__get_xxprobe_data__(full_path, conv_key_dict), full_path)


def get_video_data(full_path):
    conv_key_dict = {}
    conv_key_dict['creation_time'] = (__vid_datetime_conv__, media.KEY_TIME)
    conv_key_dict['TAG:creation_time'] = (__vid_datetime_conv__, media.KEY_TIME)
    conv_key_dict['bit_rate'] = (__int_conv__, media.KEY_BITRATE)
    conv_key_dict['duration'] = (float, media.KEY_DURATION)
    conv_key_dict['height'] = (__int_conv__, media.KEY_HEIGHT)
    conv_key_dict['width'] = (__int_conv__, media.KEY_WIDTH)
    conv_key_dict['display_aspect_ratio'] = (__ratio_conv__, media.KEY_RATIO)
    return __adapt__data__(__get_xxprobe_data__(full_path, conv_key_dict), full_path)


def get_image_data(full_path):
    return __adapt__data__(__get_exif_data__(full_path), full_path)


def __adapt__data__(data, full_path):
    data[media.KEY_SIZE] = os.path.getsize(full_path)
    # Join Camera Vendor and Camera Model
    if __KEY_CAMERA_MODEL__ in data and __KEY_CAMERA_VENDOR__ in data:
        model = data.pop(__KEY_CAMERA_MODEL__)
        vendor = data.pop(__KEY_CAMERA_VENDOR__)
        data[media.KEY_CAMERA] = '%s: %s' % (vendor, model)
    # Add time if not exists
    if media.KEY_TIME not in data:
        if media.KEY_YEAR in data and media.KEY_TRACK in data:
            if data[media.KEY_YEAR] != 0:  # ignore year 0 - must be wrong
                # Use a date where track 1 is the newest in the given year
                minute = int(data[media.KEY_TRACK] / 60)
                second = (data[media.KEY_TRACK] - 60 * minute) % 60
                #
                data[media.KEY_TIME] = int(time.mktime((data[media.KEY_YEAR], 1, 1, 0, 59 -
    minute, 59 - second, 0, 0, 0)))
                data[media.KEY_TIME_IS_SUBSTITUTION] = True
        else:
            data[media.KEY_TIME] = int(os.path.getmtime(full_path))
            data[media.KEY_TIME_IS_SUBSTITUTION] = True
    return data
```

```
72
73
74  def __get_xxprobe_data__(full_path, conv_key_dict):
75      def _ffprobe_command(full_path):
76          return ['ffprobe', '-v', 'quiet', '-show_format', '-show_streams', full_path]
77
78      def _avprobe_command(full_path):
79          return ['avprobe', '-v', 'quiet', '-show_format', '-show_streams', full_path]
80
81      try:
82          xxprobe_text = subprocess.check_output(_avprobe_command(full_path))
83      except FileNotFoundError:
84          try:
85              xxprobe_text = subprocess.check_output(_ffprobe_command(full_path))
86          except FileNotFoundError:
87              logger.warning('ffprobe and avprobe seem to be not installed')
88              return {}
89      #
90      rv = {}
91      for line in xxprobe_text.decode('utf-8').splitlines():
92          try:
93              key, val = [snippet.strip() for snippet in line.split('=')]
94          except ValueError:
95              continue
96          else:
97              if key in conv_key_dict:
98                  tp, name = conv_key_dict[key]
99                  try:
100                     rv[name] = tp(val)
101                 except ValueError:
102                     logger.log(logging.WARNING if val else logger.INFO, 'Can\'t convert %s (%s)
        for %s', repr(val), name, name)
103     return rv
104
105
106 def __get_exif_data__(full_path):
107     rv = {}
108     im = Image.open(full_path)
109     try:
110         exif = dict(im._getexif().items())
111     except AttributeError:
112         logger.debug('%s does not have any exif information', full_path)
113     else:
114         conv_key_dict = {}
115         # IMAGE
116         conv_key_dict[0x9003] = (__datetime_conv__, media.KEY_TIME)
117         conv_key_dict[0x8822] = (__exposure_program_conv__, media.KEY_EXPOSURE_PROGRAM)
118         conv_key_dict[0x829A] = (__num_denum_conv__, media.KEY_EXPOSURE_TIME)
119         conv_key_dict[0x9209] = (__flash_conv__, media.KEY_FLASH)
120         conv_key_dict[0x829D] = (__num_denum_conv__, media.KEY_APERTURE)
121         conv_key_dict[0x920A] = (__num_denum_conv__, media.KEY_FOCAL_LENGTH)
122         conv_key_dict[0x8825] = (__gps_conv__, media.KEY_GPS)
123         conv_key_dict[0xA003] = (__int_conv__, media.KEY_HEIGHT)
124         conv_key_dict[0x8827] = (__int_conv__, media.KEY_ISO)
125         conv_key_dict[0x010F] = (str, __KEY_CAMERA_VENDOR__)
126         conv_key_dict[0x0110] = (str, __KEY_CAMERA_MODEL__)
127         conv_key_dict[0x0112] = (__int_conv__, media.KEY_ORIENTATION)
128         conv_key_dict[0xA002] = (__int_conv__, media.KEY_WIDTH)
129         for key in conv_key_dict:
130             if key in exif:
131                 tp, name = conv_key_dict[key]
132                 value = tp(exif[key])
133                 if value is not None:
134                     rv[name] = value
135     return rv
```

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

```
197
198
199 def __num_denum_conv__(data):
200     num, denum = data
201     return num / denum
202
203
204 def __gps_conv__(data):
205     def lat_lon_cal(lon_or_lat):
206         lon_lat = 0.
207         fac = 1.
208         for num, denum in lon_or_lat:
209             lon_lat += float(num) / float(denum) * fac
210             fac *= 1. / 60.
211         return lon_lat
212     try:
213         lon = lat_lon_cal(data[0x0004])
214         lat = lat_lon_cal(data[0x0002])
215         if lon != 0 or lat != 0:    # do not use lon and lat equal 0, caused by motorola gps
    weakness
216             return {'lon': lon, 'lat': lat}
217     except KeyError:
218         logger.warning('GPS data extraction failed for %s', repr(data))
219
220
221 def __ratio_conv__(ratio):
222     ratio = ratio.replace('\\', '')
223     num, denum = ratio.split(':')
224     return float(num) / float(denum)
```