Unittest for `media`

June 10, 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	f446893f5cde38801fb0d031f1e42c7c
Dependencies	

1.2 Unittest Information

Unittest Information	
Version	9ce15e9e72155894dc2e3ca04e883f92
Testruns with	python 3.6.9 (final)

1.3 Test System Information

System Information	
Architecture	64bit
Distribution	LinuxMint 19.3 tricia
Hostname	ahorn
Kernel	5.3.0-53-generic (#47 18.04.1-Ubuntu SMP Thu May 7 13:10:50 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	**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	5.500s

2.2 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
media	97.8%	96.7%
media.__init__.py	99.3%	
media.common.py	100.0%	
media.convert.py	86.7%	
media.metadata.py	98.2%	

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 (27)
Start-Time:	2020-06-10 14:57:04,536
Finished-Time:	2020-06-10 14:57:05,119
Time-Consumption	0.583s

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': '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 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, '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'>).

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 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (31)
Start-Time:	2020-06-10 14:57:05,119
Finished-Time:	2020-06-10 14:57:05,549
Time-Consumption	0.429s

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 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (32)
Start-Time:	2020-06-10 14:57:05,549
Finished-Time:	2020-06-10 14:57:05,840
Time-Consumption	0.291s

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'>).

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 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (33)
Start-Time:	2020-06-10 14:57:05,840
Finished-Time:	2020-06-10 14:57:06,056
Time-Consumption	0.216s

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 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (34)
Start-Time:	2020-06-10 14:57:06,059
Finished-Time:	2020-06-10 14:57:06,196
Time-Consumption	0.137s

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'>).

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 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (35)
Start-Time:	2020-06-10 14:57:06,196
Finished-Time:	2020-06-10 14:57:07,471
Time-Consumption	1.275s

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 3.6.9 (final)
Caller:	/user_data/data/dirk/prj/unittest/media/unittest/src/tests/__init__.py (36)
Start-Time:	2020-06-10 14:57:07,473
Finished-Time:	2020-06-10 14:57:10,042
Time-Consumption	2.569s

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 3.6.9 (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'>).

```
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'>).

```
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 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'>).

```
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, '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 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 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'>).

```
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	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.1.6 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'>)
```

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'>)
```

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
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
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
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
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 97.8%
The branch coverage for media was 96.7%

B.1.1 media.__init__.py

The line coverage for media.__init__.py was 99.3%
The branch coverage for media.__init__.py was 96.7%

```
#!/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 io
import logging
from PIL import Image, ImageEnhance, ExifTags

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"""


KEY_ALBUM = 'album'
KEY_APERTURE = 'aperture'
KEY_ARTIST = 'artist'
KEY_BITRATE = 'bitrate'
KEY_CAMERA = 'camera'
KEY_DURATION = 'duration'
KEY_EXPOSURE_PROGRAM = 'exposure_program'
KEY_EXPOSURE_TIME = 'exposure_time'
KEY_FLASH = 'flash'
KEY_FOCAL_LENGTH = 'focal_length'
KEY_GENRE = 'genre'
KEY_GPS = 'gps'
KEY_HEIGHT = 'height'
```

```
55 KEY_ISO = 'iso'
56 KEY_ORIENTATION = 'orientation'
57 KEY_RATIO = 'ratio'
58 KEY_SIZE = 'size'
59 KEY_TIME = 'time'   # USE time.localtime(value) or datetime.fromtimestamp(value) to convert the
       timestamp
60 KEY_TIME_IS_SUBSTITUTION = 'tm_is_subst'
61 KEY_TITLE = 'title'
62 KEY_TRACK = 'track'
63 KEY_WIDTH = 'width'
64 KEY_YEAR = 'year'
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  ', 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         elif join_pos == JOIN_CENTER:
213             layer.paste(im2, (int((self._im.size[0] - im2.size[0]) / 2), int((self._im.size[1] -
    im2.size[1]) / 2)))
214         else:
215             logger.warning("Join position value %s is not supported", join_pos)
216             return False
217 
218         logger.debug('Joining two images')
219         self._im = Image.composite(layer, self._im, layer)
220 
221         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 96.7%

```
1 import os
2 
3 FILETYPE_AUDIO = 'audio'
4 FILETYPE_IMAGE = 'image'
5 FILETYPE_VIDEO = 'video'
6 
```

```
EXTENTIONS_AUDIO = ['.mp3', ]
EXTENTIONS_IMAGE = ['.jpg', '.jpeg', '.jpe', '.png', '.tif', '.tiff', '.gif', ]
EXTENTIONS_VIDEO = ['.avi', '.mpg', '.mpeg', '.mpe', '.mov', '.qt', '.mp4', '.webm', '.ogv', '.flv', '.3gp', ]


def get_filetype(full_path):
    ext = os.path.splitext(full_path.lower())[1]
    if ext in EXTENTIONS_AUDIO:
        return FILETYPE_AUDIO
    elif ext in EXTENTIONS_IMAGE:
        return FILETYPE_IMAGE
    elif ext in EXTENTIONS_VIDEO:
        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 96.7%

```
import io
from media import common, logger
from PIL import Image
import subprocess
import platform


def get_pil_image(media_instance):
    try:
        media_instance = media_instance._im
    except AttributeError:
        pass
    #
    if type(media_instance) == str:
        ft = common.get_filetype(media_instance)
        if ft == common.FILETYPE_IMAGE:
            return Image.open(media_instance)
        elif ft == common.FILETYPE_VIDEO:
            if platform.system() == 'Linux':
                cmd = 'ffmpeg -ss 0.5 -i "' + media_instance + '" -vframes 1 -f image2pipe pipe:1 2> /dev/null'
            else:
                cmd = 'ffmpeg -ss 0.5 -i "' + media_instance + '" -vframes 1 -f image2pipe pipe:1 2> NULL'
            try:
                data = subprocess.check_output(cmd, shell=True)
            except subprocess.CalledProcessError:
                logger.warning('ffmpeg seems to be not installed')
                return None
            ffmpeg_handle = io.BytesIO(data)
            im = Image.open(ffmpeg_handle)
            return im.copy()
        logger.warning('Filetype is not supported (%s)', media_instance)
    elif type(media_instance) == Image.Image:
        return media_instance.copy()
    else:
        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 98.2%
The branch coverage for media.metadata.py was 96.7%

```
import logging
import os
from PIL import Image
import media
import subprocess
import time


logger = media.logger


__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


def __get_xxprobe_data__(full_path, conv_key_dict):
    def _ffprobe_command(full_path):
        return ['ffprobe', '-v', 'quiet', '-show_format', '-show_streams', full_path]

    def _avprobe_command(full_path):
        return ['avprobe', '-v', 'quiet', '-show_format', '-show_streams', full_path]

    try:
        xxprobe_text = subprocess.check_output(_avprobe_command(full_path))
    except FileNotFoundError:
        try:
            xxprobe_text = subprocess.check_output(_ffprobe_command(full_path))
        except FileNotFoundError:
            logger.warning('ffprobe and avprobe seem to be not installed')
            return {}
    #
    rv = {}
    for line in xxprobe_text.decode('utf-8').splitlines():
        try:
            key, val = [snippet.strip() for snippet in line.split('=')]
        except ValueError:
            continue
        else:
            if key in conv_key_dict:
                tp, name = conv_key_dict[key]
                try:
                    rv[name] = tp(val)
                except ValueError:
                    logger.log(logging.WARNING if val else logger.INFO, 'Can\'t convert %s (%s)
    for %s', repr(val), name, name)
    return rv


def __get_exif_data__(full_path):
    rv = {}
    im = Image.open(full_path)
    try:
        exif = dict(im._getexif().items())
    except AttributeError:
        logger.debug('%s does not have any exif information', full_path)
    else:
        conv_key_dict = {}
        # IMAGE
```

```
        conv_key_dict[0x9003] = (__datetime_conv__, media.KEY_TIME)
        conv_key_dict[0x8822] = (__exposure_program_conv__, media.KEY_EXPOSURE_PROGRAM)
        conv_key_dict[0x829A] = (__num_denum_conv__, media.KEY_EXPOSURE_TIME)
        conv_key_dict[0x9209] = (__flash_conv__, media.KEY_FLASH)
        conv_key_dict[0x829D] = (__num_denum_conv__, media.KEY_APERTURE)
        conv_key_dict[0x920A] = (__num_denum_conv__, media.KEY_FOCAL_LENGTH)
        conv_key_dict[0x8825] = (__gps_conv__, media.KEY_GPS)
        conv_key_dict[0xA003] = (__int_conv__, media.KEY_HEIGHT)
        conv_key_dict[0x8827] = (__int_conv__, media.KEY_ISO)
        conv_key_dict[0x010F] = (str, __KEY_CAMERA_VENDOR__)
        conv_key_dict[0x0110] = (str, __KEY_CAMERA_MODEL__)
        conv_key_dict[0x0112] = (__int_conv__, media.KEY_ORIENTATION)
        conv_key_dict[0xA002] = (__int_conv__, media.KEY_WIDTH)
        for key in conv_key_dict:
            if key in exif:
                tp, name = conv_key_dict[key]
                value = tp(exif[key])
                if value is not None:
                    rv[name] = value
    return rv


# TODO: Join datetime converter __datetime_conv__ and __vid_datetime_conv_
def __datetime_conv__(dt):
    format_string = "%Y:%m:%d %H:%M:%S"
    return int(time.mktime(time.strptime(dt, format_string)))


def __vid_datetime_conv__(dt):
    try:
        dt = dt[:dt.index('.')]
    except ValueError:
        pass  # time string seems to have no '.'
    dt = dt.replace('T', ' ').replace('/', '').replace('\\', '')
    if len(dt) == 16:
        dt += ':00'
    format_string = '%Y-%m-%d %H:%M:%S'
    return int(time.mktime(time.strptime(dt, format_string)))


def __exposure_program_conv__(n):
    return {
        0: 'Unidentified',
        1: 'Manual',
        2: 'Program Normal',
        3: 'Aperture Priority',
        4: 'Shutter Priority',
        5: 'Program Creative',
        6: 'Program Action',
        7: 'Portrait Mode',
        8: 'Landscape Mode'
    }.get(n, None)


def __flash_conv__(n):
    return {
        0: 'No',
        1: 'Fired',
        5: 'Fired (?)',  # no return sensed
        7: 'Fired (!)',  # return sensed
        9: 'Fill Fired',
```

```
        13: 'Fill Fired (?)',
        15: 'Fill Fired (!)',
        16: 'Off',
        24: 'Auto Off',
        25: 'Auto Fired',
        29: 'Auto Fired (?)',
        31: 'Auto Fired (!)',
        32: 'Not Available'
    }.get(n, None)


def __int_conv__(value):
    try:
        return int(value)
    except ValueError:
        for c in ['.', '/', '-']:
            p = value.find(c)
            if p >= 0:
                value = value[:p]
    try:
        return int(value)
    except ValueError:
        return None


def __num_denum_conv__(data):
    num, denum = data
    return num / denum


def __gps_conv__(data):
    def lat_lon_cal(lon_or_lat):
        lon_lat = 0.
        fac = 1.
        for num, denum in lon_or_lat:
            try:
                lon_lat += float(num) / float(denum) * fac
            except ZeroDivisionError:
                return 0.
            fac *= 1. / 60.
        return lon_lat
    try:
        lon = lat_lon_cal(data[0x0004])
        lat = lat_lon_cal(data[0x0002])
        if lon != 0 or lat != 0:    # do not use lon and lat equal 0, caused by motorola gps weakness
            return {'lon': lon, 'lat': lat}
    except KeyError:
        logger.warning('GPS data extraction failed for %s', repr(data))


def __ratio_conv__(ratio):
    ratio = ratio.replace('\\', '')
    num, denum = ratio.split(':')
    return float(num) / float(denum)
```