Unittest for caching

May 15, 2026

Contents

1 Test Information **3**

1.1 Testobject Information . . . 3

1.2 Testdata Information . . . 3

1.3 Testsystem Information . . . 3

2 Statistic **3**

2.1 Test-Statistic for testrun: Library test . . . 3

2.2 Coverage Statistic . . . 4

3 Tested Requirements for testrun: Library test **5**

3.1 Cache generation (json /pickle) . . . 5

3.1.1 Data generation from source instance, if no cache is available (REQ-0003) . . . 5

3.1.2 Create complete cache from the given data instance (REQ-0001) . . . 6

3.1.3 Create cache partially from a given data instance by get method (REQ-0005) . . . 7

3.1.4 Ignore corrupt cache file (REQ-0015) . . . 8

3.2 Load spreading for full update . . . 8

3.2.1 Full update with delay between each data generation for the cache (REQ-0004) . . . 8

3.2.2 No cache generation if disabled (REQ-0002) . . . 9

3.3 Dump cache conditions . . . 10

3.3.1 Dump cache if time is expired (REQ-0006) . . . 10

3.3.2 Dump cache if data version increases (REQ-0007) . . . 11

3.3.3 Dump cache if data uid is changed (REQ-0008) . . . 11

3.3.4 Dump cache if storage version is changed (REQ-0009) . . . 12

3.4 Edge cases . . . 13

3.4.1 Define uncached data (REQ-0010) . . . 13

3.4.2 Ignore cache if value is 'None' (REQ-0014) . . . 14

3.4.3 Return 'None' as value from cache, if configured. (REQ-0016) . . . 15

3.5 Callback on data storage . . . 15

3.5.1 If no data is changed, no callback will be executed (REQ-0011) . . . 15

3.5.2 Callback execution in case of a full update (REQ-0012) . . . 16

3.5.3 Callback execution in case of get function (REQ-0013) . . . 16

A Trace for testrun: Library test **18**

A.1 Tests with status Info (16) . . . 18

A.1.1 Data generation from source instance, if no cache is available (REQ-0003) . . . 18

A.1.2 Create complete cache from the given data instance (REQ-0001) . . . 19

A.1.3 Create cache partially from a given data instance by get method (REQ-0005) . . . 21

A.1.4 Ignore corrupt cache file (REQ-0015) . . . 23

A.1.5 Full update with delay between each data generation for the cache (REQ-0004) . . . 24

A.1.6 No cache generation if disabled (REQ-0002) . . . 24

A.1.7 Dump cache if time is expired (REQ-0006) . . . 26

A.1.8 Dump cache if data version increases (REQ-0007) . . . 27

A.1.9 Dump cache if data uid is changed (REQ-0008) . . . 29

A.1.10 Dump cache if storage version is changed (REQ-0009) . . . 30

A.1.11 Define uncached data (REQ-0010) . . . 31

A.1.12 Ignore cache if value is 'None' (REQ-0014) . . . 33

A.1.13 Return 'None' as value from cache, if configured. (REQ-0016) . . . 33

A.1.14 If no data is changed, no callback will be executed (REQ-0011) . . . 34

A.1.15 Callback execution in case of a full update (REQ-0012) . . . 35

A.1.16 Callback execution in case of get function (REQ-0013) . . . 36

B Test-Coverage **36**

B.1 `caching` . . . 36

B.1.1 `caching.__init__.py` . . . 37

1 Test Information

1.1 Testobject Information

The Module caching is designed to store information in json or pickle files to support them much faster then generating them from the original source file. For more Information read the documentation.

Information	
Name	caching
State	Released
Version	1.2.6
Dependencies	
	Python standards

1.2 Testdata Information

Information	
Version	1.2.6
Testruns	Library test

1.3 Testsystem Information

System Information	
Architecture	64bit
Distribution	Debian GNU/Linux 13 trixie
Hostname	ahorn
Kernel	6.12.86+deb13-amd64 (#1 SMP PREEMPT_DYNAMIC Debian 6.12.86-1 (2026-05-08))
Machine	x86_64
Path	/home/dirk/work/unittest_collection/caching/unittest/src
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun: Library test

Number of tests	**16**
Number of successfull tests	**16**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	8.179s

2.2 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
caching	98.8%	100.0%
caching.__init__.py	98.8%	

3 Tested Requirements for testrun: Library test

3.1 Cache generation (json /pickle)

3.1.1 Data generation from source instance, if no cache is available (REQ-0003)

Description
If the cache is not available, the data shall be generated from the source instance.

Reason for the implementation
There shall be the posibility to create the cache on demand, so the fallback is to generate the data from the source instance.

Fitcriterion
Caching is called without previous cache generation and the data from the source instance is completely available.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (26)
Start-Time:	2026-05-15 14:52:36,194
Finished-Time:	2026-05-15 14:52:36,197
Time-Consumption	0.003s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Create a test cache instance without previous cache generation.
Info	Checking that all source data is available in the cached instance
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).
Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).
Info	Checking none existing keys of source instance result in the default value.
Success	Data from cached instance with key=unknown_key is correct (Content None and Type is <class 'NoneType'>).
Success	Data from cached instance with key=unknown_key is correct (Content 5 and Type is <class 'int'>).
Success	Data from cached instance with key=unknown_key is correct (Content 'five' and Type is <class 'str'>).

3.1.2 Create complete cache from the given data instance (REQ-0001)

Description

There shall be a method caching all information from the given instance.

Reason for the implementation

Independent usage of data generation and data usage (e.g. the user requesting the data is not able to create the data).

Fitcriterion

Caching is called twice with different data instances and the cached data from the first call is completely available.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (27)
Start-Time:	2026-05-15 14:52:36,197
Finished-Time:	2026-05-15 14:52:36,201
Time-Consumption	0.004s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing full cache file including None and creating cache test instance with different source instance
Info	Checking that all cached source data is available in the test instance
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).
Success	Data from cached instance with key=None is correct (Content None and Type is <class 'NoneType'>).
Info	Checking none existing keys of source instance result in the default value.
Success	Data from cached instance with key=unknown_key is correct (Content None and Type is <class 'NoneType'>).
Success	Data from cached instance with key=unknown_key is correct (Content 5 and Type is <class 'int'>).
Success	Data from cached instance with key=unknown_key is correct (Content 'five' and Type is <class 'str'>).

3.1.3 Create cache partially from a given data instance by get method (REQ-0005)

Description

On getting data from the cached instance, the information shall be stored in the cache file.

Reason for the implementation

There shall be the posibility to create the cache on demand, so the fallback is to generate the data from the source instance.

Fitcriterion

Caching is called twice with different data instances and the cached data from the first call is available for all keys cached on the first run.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (28)
Start-Time:	2026-05-15 14:52:36,201
Finished-Time:	2026-05-15 14:52:36,218
Time-Consumption	0.017s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file including partial data from source instance and creating test cache instance (pickle).
Info	Cached keys shall be equivalent cache generation data. Other keys shall be equivalent to current different source data.
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).
Success	Data from cached instance with key=None is correct (Content None and Type is <class 'NoneType'>).
Info	Checking none existing keys of source instance result in the default value.
Success	Data from cached instance with key=unknown_key is correct (Content None and Type is <class 'NoneType'>).
Success	Data from cached instance with key=unknown_key is correct (Content 5 and Type is <class 'int'>).
Success	Data from cached instance with key=unknown_key is correct (Content 'five' and Type is <class 'str'>).

3.1.4 Ignore corrupt cache file (REQ-0015)

Description
Ignore corrupt cachefile, while loading cache.

Reason for the implementation
Suppress exceptions while caching.

Fitcriterion
Loading cache results in no exception, when cache file is empty.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (31)
Start-Time:	2026-05-15 14:52:36,218
Finished-Time:	2026-05-15 14:52:36,219
Time-Consumption	0.001s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing test instance and creating empty (corrupted) cache file
Success	Empty cache file ignored on loading cache.

3.2 Load spreading for full update

3.2.1 Full update with delay between each data generation for the cache (REQ-0004)

Description
The full update method shall pause for a given time between every cached item.

Reason for the implementation
System load spreading in case of (cyclic called) `.full_update()`.

Fitcriterion
The time consumption of the method `.full_update(<sleep_time>)` shall consume n times the given `sleep_time`. Where n is the number of items which will be cached from the source instance.

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (34)

Start-Time:	2026-05-15 14:52:36,219
Finished-Time:	2026-05-15 14:52:42,228
Time-Consumption	6.008s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing test cache instance including None and adding one key to get from source
Info	Executing full update with sleep time of 1.0 seconds
Success	Consumed time for full_update is greater expectation (Content 6.003793478012085 and Type is <class 'float'>).
Success	Consumed time for full_update is greater expectation (Content 6.003793478012085 and Type is <class 'float'>).

3.2.2 No cache generation if disabled (REQ-0002)

Description

The cache shall be generated by the `.get()` method, only if the cache instance parameter `store_on_get` is set to `True`.

Reason for the implementation

Independent usage of data generation and data usage (e.g. the user requesting the data is not able to create the data).

Fitcriterion

Create a caching instance with `store_on_get` set to `False`. Get every item of the source instance with the `.get()` method and check that no cache file exists.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (35)
Start-Time:	2026-05-15 14:52:42,229
Finished-Time:	2026-05-15 14:52:42,237
Time-Consumption	0.008s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing test cache instance including None without creating cache and store_on_get disabled
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

Success	Data from cached instance with key=None is correct (Content None and Type is <class 'NoneType'>).
Success	The cache file (/home/dirk/work/unittest_collection/caching/unittest/output_data/req0002.json) shall not exist is correct (Content False and Type is <class 'bool'>).

3.3 Dump cache conditions

3.3.1 Dump cache if time is expired (REQ-0006)

Description

Dump the cached item, if this item is older then the given expirery time.

Reason for the implementation

Ensure, that the cache is updated from time to time. For example for items which do not change very often.

Fitcriterion

Create a cache instance, cache some data. Intialise a second caching instance with a different source instance and a expire time. Wait for longer than the given expiry time and check that the items from the second source instance are returned.

Testresult

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

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (38)
Start-Time:	2026-05-15 14:52:42,238
Finished-Time:	2026-05-15 14:52:44,281
Time-Consumption	2.043s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file and cache test instance with max_age=2.
Info	Waiting for cache to expire
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).
Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

3.3.2 Dump cache if data version increases (REQ-0007)

Description

Dump the complete cache, if the *data version* of the source instance is increased.

Reason for the implementation

The data version is part of the source instance. Increasing the data version indicates, that the source instance generates the data in another way or the structure of the data is changed. In that condition, the cache needs to be ignored.

Fitcriterion

Create a cached instance and cache some items. Generate a second cached instance with different source data and a increased data version. Ensure, that the cache instance returns the values from the second source. It is required to set `load_all_on_init` to `False` and `store_on_get` to `True`.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.8!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (39)
Start-Time:	2026-05-15 14:52:44,281
Finished-Time:	2026-05-15 14:52:44,310
Time-Consumption	0.029s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file and test cache instance with different data version of source instance
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).
Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

3.3.3 Dump cache if data uid is changed (REQ-0008)

Description

Dump the complete cache, if the *data uid* of the source instance is changed.

Reason for the implementation

The data uid is part of the source instance. Changing the data uid indicates, that the source of the data created by the

source instance is changed (e.g. the uid of a file or folder) and the cache needs to be ignored.

Fitcriterion

Create a cached instance and cache some items. Generate a second cached instance with different source data and a changed data uid. Ensure, that the cache instance returns the values from the second source. It is required to set `load_all_on_init` to `False` and `store_on_get` to `True`.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.9!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (40)
Start-Time:	2026-05-15 14:52:44,311
Finished-Time:	2026-05-15 14:52:44,340
Time-Consumption	0.029s

Testsummary:	
Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file and test cache instance with different uid of source instance
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).
Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

3.3.4 Dump cache if storage version is changed (REQ-0009)

Description

Dump the complete cache, if the *storage version* of the caching class is changed.

Reason for the implementation

The storage version is part of the caching class. Changing the storage version indicates, that the previously stored cache is not compatible due to new data storage and the cache needs to be ignored.

Fitcriterion

Create a cached instance and cache some items. Generate a second cached instance with different source data and a changed storage version. Ensure, that the cache instance returns the values from the second source. It is required to set `load_all_on_init` to `False` and `store_on_get` to `True`.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.10!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (41)
Start-Time:	2026-05-15 14:52:44,341
Finished-Time:	2026-05-15 14:52:44,348
Time-Consumption	0.007s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file and test cache instance with different storage version
Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).
Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

3.4 Edge cases

3.4.1 Define uncached data (REQ-0010)

Description
It shall be possible to define items which are not cached.

Reason for the implementation
If there is dynamic changed data in the source instance, it shall be possible to define these items as non cached to get them always from the source instance.

Fitcriterion
Create a cached instance and cache some items. Generate a second cached instance with different source data and set at least one item as source item. This item should be previously cached. Ensure, that the source item isis the one from the second source instance.

Testresult
This test was passed with the state: **Success**. See also full trace in section A.1.11!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (44)
Start-Time:	2026-05-15 14:52:44,349

Finished-Time:	2026-05-15 14:52:44,356
Time-Consumption	0.007s

Testsummary:	
Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file and creating cache test instance
Info	Check cached data against expected data. Source keys: ['str', 'float']
Success	Data from cached instance with key=str is correct (Content '__string__' and Type is <class 'str'>).
Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).
Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).
Success	Data from cached instance with key=float is correct (Content 2.71828 and Type is <class 'float'>).
Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).
Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

3.4.2 Ignore cache if value is 'None' (REQ-0014)

Description

Ignore the cached value, if the stored value in the cache is `None`.

Reason for the implementation

In most cases, "None" as value means, that the value does not exist (yet) or it wasn't possible to determine the value while calling the get function. In that case, it might be a good idea to ignore the cache and return the value of the source instance (which possibly results in a real value).

Fitcriterion

Create a cached instance and cache some items. One needs to have `None` as value. Generate a second cached instance with different source data (especially, the previous item with value `None` needs to have a not `None` value. Ensure, that the caching instance returns not `None` from the second source.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.12!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (45)
Start-Time:	2026-05-15 14:52:44,356
Finished-Time:	2026-05-15 14:52:44,360
Time-Consumption	0.004s

Testsummary:	
Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file including None and creating cache test instance
Success	Uncached value (None) is correct (Content 'notNone' and Type is <class 'str'>).

3.4.3 Return 'None' as value from cache, if configured. (REQ-0016)

Description

None shall be returned as cached value, if it is available in the cache.

Reason for the implementation

In some cases it might be usefull to return "None" as cached value, because "None" is a valid stable value of the source instance.

Fitcriterion

Create a cached instance and cache some items. One needs to have None as value. Generate a second cached instance with different source data (especially, the previous item with value None needs to have a not None value. Ensure, that None is returned from the cache.

Testresult

This test was passed with the state: Success. See also full trace in section A.1.13!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (46)
Start-Time:	2026-05-15 14:52:44,361
Finished-Time:	2026-05-15 14:52:44,364
Time-Consumption	0.004s

Testsummary:	
Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing cache file including None and creating cache test instance
Success	Cached value (None) is correct (Content None and Type is <class 'NoneType'>).

3.5 Callback on data storage

3.5.1 If no data is changed, no callback will be executed (REQ-0011)

Description

The store callback shall not be executed, if no cache is stored.

Reason for the implementation

Do actions, if cache data is stored to disk.

Fitcriterion

Initialise the cache instance without storing cache data. Ensure, that the callback is never executed.

Testresult

This test was passed with the state: Success. See also full trace in section A.1.14!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (49)
Start-Time:	2026-05-15 14:52:44,365
Finished-Time:	2026-05-15 14:52:44,367
Time-Consumption	0.002s

Testsummary:	
Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing test instance with save callback. No actions with test cache instance.
Success	Save callback execution counter is correct (Content 0 and Type is <class 'int'>).
Success	Save callback cache instance is correct (Content None and Type is <class 'NoneType'>).

3.5.2 Callback execution in case of a full update (REQ-0012)

Description

The storage callback shall be called once on every `full_update()`.

Reason for the implementation

Do actions, if cache data is stored to disk.

Fitcriterion

Initialise the cache instance and ensure, that the callback is executed as often as the `.full_update()` method is executed.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.15!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (50)
Start-Time:	2026-05-15 14:52:44,367
Finished-Time:	2026-05-15 14:52:44,371
Time-Consumption	0.004s

Testsummary:	
Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing test instance with save callback and executing full_update.
Success	Save callback execution counter is correct (Content 1 and Type is <class 'int'>).
Success	Save callback execution counter is correct (Content <caching.property_cache_json object at 0x7fb45a50a850> and Type is <class 'caching.property_cache_json'>).

3.5.3 Callback execution in case of get function (REQ-0013)

Description

The storage callback, shall be called once on every `.get()`, if `storage_on_get` is set to `True`.

Reason for the implementation

Do actions, if cache data is stored to disk.

Fitcriterion

Initialise the cache instance and ensure, that the callback is executed as often as the `.get()` method is executed.

Testresult

This test was passed with the state: **Success**. See also full trace in section A.1.16!

Testrun:	
Caller:	/home/dirk/work/unittest_collection/caching/unittest/src/tests/__init__.py (51)
Start-Time:	2026-05-15 14:52:44,372
Finished-Time:	2026-05-15 14:52:44,380
Time-Consumption	0.008s

Testsummary:

Info	Prepare: Cleanup cache file before testcase execution
Info	Preparing test instance with save callback and executing get method 3 times.
Success	Save callback execution counter is correct (Content 3 and Type is <class 'int'>).
Success	Save callback execution counter is correct (Content <caching.property_cache_json object at 0x7fb45a50ab70> and Type is <class 'caching.property_cache_json'>).

A Trace for testrun: Library test

A.1 Tests with status Info (16)

A.1.1 Data generation from source instance, if no cache is available (REQ-0003)

Description
If the cache is not available, the data shall be generated from the source instance.

Reason for the implementation
There shall be the posibility to create the cache on demand, so the fallback is to generate the data from the source instance.

Fitcriterion
Caching is called without previous cache generation and the data from the source instance is completely available.

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Create a test cache instance without previous cache generation.

Info	Checking that all source data is available in the cached instance

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
```

```
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

```
Result (Data from cached instance with key=None): 'notNone' (<class 'str'>)
Expectation (Data from cached instance with key=None): result = 'notNone' (<class 'str'>)
```

Info	Checking none existing keys of source instance result in the default value.

Success	Data from cached instance with key=unknown_key is correct (Content None and Type is <class 'NoneType'>).

```
Result (Data from cached instance with key=unknown_key): None (<class 'NoneType'>)
Expectation (Data from cached instance with key=unknown_key): result = None (<class
↪  'NoneType'>)
```

Success	Data from cached instance with key=unknown_key is correct (Content 5 and Type is <class 'int'>).

```
Result (Data from cached instance with key=unknown_key): 5 (<class 'int'>)
Expectation (Data from cached instance with key=unknown_key): result = 5 (<class 'int'>)
```

Success	Data from cached instance with key=unknown_key is correct (Content 'five' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unknown_key): 'five' (<class 'str'>)
Expectation (Data from cached instance with key=unknown_key): result = 'five' (<class 'str'>)
```

A.1.2 Create complete cache from the given data instance (REQ-0001)

Description

There shall be a method caching all information from the given instance.

Reason for the implementation

Independent usage of data generation and data usage (e.g. the user requesting the data is not able to create the data).

Fitcriterion

Caching is called twice with different data instances and the cached data from the first call is completely available.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing full cache file including None and creating cache test instance with different source instance

Info	Checking that all cached source data is available in the test instance

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
```

```
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content None and Type is <class 'NoneType'>).

```
Result (Data from cached instance with key=None): None (<class 'NoneType'>)
Expectation (Data from cached instance with key=None): result = None (<class 'NoneType'>)
```

Info	Checking none existing keys of source instance result in the default value.

Success	Data from cached instance with key=unknown_key is correct (Content None and Type is <class 'NoneType'>).

```
Result (Data from cached instance with key=unknown_key): None (<class 'NoneType'>)
Expectation (Data from cached instance with key=unknown_key): result = None (<class
↪  'NoneType'>)
```

Success	Data from cached instance with key=unknown_key is correct (Content 5 and Type is <class 'int'>).

```
Result (Data from cached instance with key=unknown_key): 5 (<class 'int'>)
Expectation (Data from cached instance with key=unknown_key): result = 5 (<class 'int'>)
```

Success	Data from cached instance with key=unknown_key is correct (Content 'five' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unknown_key): 'five' (<class 'str'>)
Expectation (Data from cached instance with key=unknown_key): result = 'five' (<class 'str'>)
```

A.1.3 Create cache partially from a given data instance by get method (REQ-0005)

Description

On getting data from the cached instance, the information shall be stored in the cache file.

Reason for the implementation

There shall be the posibility to create the cache on demand, so the fallback is to generate the data from the source instance.

Fitcriterion

Caching is called twice with different data instances and the cached data from the first call is available for all keys cached on the first run.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file including partial data from source instance and creating test cache instance (pickle).

Info	Cached keys shall be equivalent cache generation data. Other keys shall be equivalent to current different source data.

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content None and Type is <class 'NoneType'>).

```
Result (Data from cached instance with key=None): None (<class 'NoneType'>)
Expectation (Data from cached instance with key=None): result = None (<class 'NoneType'>)
```

Info	Checking none existing keys of source instance result in the default value.

Success	Data from cached instance with key=unknown_key is correct (Content None and Type is <class 'NoneType'>).

```
Result (Data from cached instance with key=unknown_key): None (<class 'NoneType'>)
Expectation (Data from cached instance with key=unknown_key): result = None (<class
↪  'NoneType'>)
```

Success	Data from cached instance with key=unknown_key is correct (Content 5 and Type is <class 'int'>).

```
Result (Data from cached instance with key=unknown_key): 5 (<class 'int'>)
Expectation (Data from cached instance with key=unknown_key): result = 5 (<class 'int'>)
```

Success	Data from cached instance with key=unknown_key is correct (Content 'five' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unknown_key): 'five' (<class 'str'>)
Expectation (Data from cached instance with key=unknown_key): result = 'five' (<class 'str'>)
```

A.1.4 Ignore corrupt cache file (REQ-0015)

Description

Ignore corrupt cachefile, while loading cache.

Reason for the implementation

Suppress exceptions while caching.

Fitcriterion

Loading cache results in no exception, when cache file is empty.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing test instance and creating empty (corrupted) cache file

Success	Empty cache file ignored on loading cache.

A.1.5 Full update with delay between each data generation for the cache (REQ-0004)

Description
The full update method shall pause for a given time between every cached item.

Reason for the implementation
System load spreading in case of (cyclic called) `.full_update()`.

Fitcriterion
The time consumption of the method `.full_update(<sleep_time>)` shall consume n times the given `sleep_time`. Where n is the number of items which will be cached from the source instance.

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing test cache instance including None and adding one key to get from source

Info	Executing full update with sleep time of 1.0 seconds

Success	Consumed time for full_update is greater expectation (Content 6.003793478012085 and Type is <class 'float'>).

```
Result (Consumed time for full_update): 6.003793478012085 (<class 'float'>)
Expectation (Consumed time for full_update): result > 6.0 (<class 'float'>)
```

Success	Consumed time for full_update is greater expectation (Content 6.003793478012085 and Type is <class 'float'>).

```
Result (Consumed time for full_update): 6.003793478012085 (<class 'float'>)
Expectation (Consumed time for full_update): result < 6.5 (<class 'float'>)
```

A.1.6 No cache generation if disabled (REQ-0002)

Description
The cache shall be generated by the `.get()` method, only if the cache instance parameter `store_on_get` is set to `True`.

Reason for the implementation
Independent usage of data generation and data usage (e.g. the user requesting the data is not able to create the data).

Fitcriterion

Create a caching instance with `store_on_get` set to `False`. Get every item of the source instance with the `.get()` method and check that no cache file exists.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing test cache instance including None without creating cache and store_on_get disabled

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content None and Type is <class 'NoneType'>).

```
Result (Data from cached instance with key=None): None (<class 'NoneType'>)
```

```
Expectation (Data from cached instance with key=None): result = None (<class 'NoneType'>)
```

Success	The cache file (/home/dirk/work/unittest_collection/caching/unittest/output_data/req0002.json) shall not exist is correct (Content False and Type is <class 'bool'>).

```
Result (The cache file
↪  (/home/dirk/work/unittest_collection/caching/unittest/output_data/req0002.json) shall not
↪  exist): False (<class 'bool'>)
```

```
Expectation (The cache file
↪  (/home/dirk/work/unittest_collection/caching/unittest/output_data/req0002.json) shall not
↪  exist): result = False (<class 'bool'>)
```

A.1.7 Dump cache if time is expired (REQ-0006)

Description

Dump the cached item, if this item is older then the given expirery time.

Reason for the implementation

Ensure, that the cache is updated from time to time. For example for items which do not change very often.

Fitcriterion

Create a cache instance, cache some data. Intialise a second caching instance with a different source instance and a expire time. Wait for longer than the given expiry time and check that the items from the second source instance are returned.

Testresult

This test was passed with the state: Success.

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file and cache test instance with max_age=2.

Info	Waiting for cache to expire

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
```

```
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

```
Result (Data from cached instance with key=None): 'notNone' (<class 'str'>)
Expectation (Data from cached instance with key=None): result = 'notNone' (<class 'str'>)
```

A.1.8 Dump cache if data version increases (REQ-0007)

Description

Dump the complete cache, if the *data version* of the source instance is increased.

Reason for the implementation

The data version is part of the source instance. Increasing the data version indicates, that the source instance generates the data in another way or the structure of the data is changed. In that condition, the cache needs to be ignored.

Fitcriterion

Create a cached instance and cache some items. Generate a second cached instance with different source data and a increased data version. Ensure, that the cache instance returns the values from the second source. It is required to set `load_all_on_init` to `False` and `store_on_get` to `True`.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file and test cache instance with different data version of source instance

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
```

```
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

```
Result (Data from cached instance with key=None): 'notNone' (<class 'str'>)
Expectation (Data from cached instance with key=None): result = 'notNone' (<class 'str'>)
```

A.1.9 Dump cache if data uid is changed (REQ-0008)

Description

Dump the complete cache, if the *data uid* of the source instance is changed.

Reason for the implementation

The data uid is part of the source instance. Changing the data uid indicates, that the source of the data created by the source instance is changed (e.g. the uid of a file or folder) and the cache needs to be ignored.

Fitcriterion

Create a cached instance and cache some items. Generate a second cached instance with different source data and a changed data uid. Ensure, that the cache instance returns the values from the second source. It is required to set `load_all_on_init` to `False` and `store_on_get` to `True`.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file and test cache instance with different uid of source instance

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
```

```
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

```
Result (Data from cached instance with key=None): 'notNone' (<class 'str'>)
Expectation (Data from cached instance with key=None): result = 'notNone' (<class 'str'>)
```

A.1.10 Dump cache if storage version is changed (REQ-0009)

Description

Dump the complete cache, if the *storage version* of the caching class is changed.

Reason for the implementation

The storage version is part of the caching class. Changing the storage version indicates, that the previously stored cache is not compatible due to new data storage and the cache needs to be ignored.

Fitcriterion

Create a cached instance and cache some items. Generate a second cached instance with different source data and a changed storage version. Ensure, that the cache instance returns the values from the second source. It is required to set `load_all_on_init` to `False` and `store_on_get` to `True`.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file and test cache instance with different storage version

Success	Data from cached instance with key=str is correct (Content 'string' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): 'string' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = 'string' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 3.14159 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 3.14159 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 3.14159 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

Success	Data from cached instance with key=None is correct (Content 'notNone' and Type is <class 'str'>).

```
Result (Data from cached instance with key=None): 'notNone' (<class 'str'>)
Expectation (Data from cached instance with key=None): result = 'notNone' (<class 'str'>)
```

A.1.11 Define uncached data (REQ-0010)

Description

It shall be possible to define items which are not cached.

Reason for the implementation

If there is dynamic changed data in the source instance, it shall be possible to define these items as non cached to get them always from the source instance.

Fitcriterion

Create a cached instance and cache some items. Generate a second cached instance with different source data and set at least one item as source item. This item should be previously cached. Ensure, that the source item isis the one from the second source instance.

Testresult

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file and creating cache test instance

Info	Check cached data against expected data. Source keys: ['str', 'float']

Success	Data from cached instance with key=str is correct (Content '__string__' and Type is <class 'str'>).

```
Result (Data from cached instance with key=str): '__string__' (<class 'str'>)
Expectation (Data from cached instance with key=str): result = '__string__' (<class 'str'>)
```

Success	Data from cached instance with key=unicode is correct (Content 'unicode' and Type is <class 'str'>).

```
Result (Data from cached instance with key=unicode): 'unicode' (<class 'str'>)
Expectation (Data from cached instance with key=unicode): result = 'unicode' (<class 'str'>)
```

Success	Data from cached instance with key=integer is correct (Content 17 and Type is <class 'int'>).

```
Result (Data from cached instance with key=integer): 17 (<class 'int'>)
Expectation (Data from cached instance with key=integer): result = 17 (<class 'int'>)
```

Success	Data from cached instance with key=float is correct (Content 2.71828 and Type is <class 'float'>).

```
Result (Data from cached instance with key=float): 2.71828 (<class 'float'>)
Expectation (Data from cached instance with key=float): result = 2.71828 (<class 'float'>)
```

Success	Data from cached instance with key=list is correct (Content [1, 'two', '3', 4] and Type is <class 'list'>).

```
Result (Data from cached instance with key=list): [ 1, 'two', '3', 4 ] (<class 'list'>)
```

```
Expectation (Data from cached instance with key=list): result = [ 1, 'two', '3', 4 ] (<class
↪  'list'>)
```

Success	Data from cached instance with key=dict is correct (Content {'1': 1, '2': 'two', '3': '3', '4': 4} and Type is <class 'dict'>).

```
Result (Data from cached instance with key=dict): { '1': 1, '2': 'two', '3': '3', '4': 4 }
↪  (<class 'dict'>)
```

```
Expectation (Data from cached instance with key=dict): result = { '1': 1, '2': 'two', '3':
↪  '3', '4': 4 } (<class 'dict'>)
```

A.1.12 Ignore cache if value is 'None' (REQ-0014)

Description

Ignore the cached value, if the stored value in the cache is `None`.

Reason for the implementation

In most cases, "None" as value means, that the value does not exist (yet) or it wasn't possible to determine the value while calling the get function. In that case, it might be a good idea to ignore the cache and return the value of the source instance (which possibly results in a real value).

Fitcriterion

Create a cached instance and cache some items. One needs to have `None` as value. Generate a second cached instance with different source data (especially, the previous item with value `None` needs to have a not `None` value. Ensure, that the caching instance returns not `None` from the second source.

Testresult

This test was passed with the state: Success.

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file including None and creating cache test instance

Success	Uncached value (None) is correct (Content 'notNone' and Type is <class 'str'>).

```
Result (Uncached value (None)): 'notNone' (<class 'str'>)
```

```
Expectation (Uncached value (None)): result = 'notNone' (<class 'str'>)
```

A.1.13 Return 'None' as value from cache, if configured. (REQ-0016)

Description

`None` shall be returned as cached value, if it is available in the cache.

Reason for the implementation
In some cases it might be usefull to return "None" as cached value, because "None" is a valid stable value of the source instance.

Fitcriterion
Create a cached instance and cache some items. One needs to have `None` as value. Generate a second cached instance with different source data (especially, the previous item with value `None` needs to have a not `None` value. Ensure, that `None` is returned from the cache.

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing cache file including None and creating cache test instance

Success	Cached value (None) is correct (Content None and Type is <class 'NoneType'>).

```
Result (Cached value (None)): None (<class 'NoneType'>)
Expectation (Cached value (None)): result = None (<class 'NoneType'>)
```

A.1.14 If no data is changed, no callback will be executed (REQ-0011)

Description
The store callback shall not be executed, if no cache is stored.

Reason for the implementation
Do actions, if cache data is stored to disk.

Fitcriterion
Initialise the cache instance without storing cache data. Ensure, that the callback is never executed.

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing test instance with save callback. No actions with test cache instance.

Success	Save callback execution counter is correct (Content 0 and Type is <class 'int'>).

```
Result (Save callback execution counter): 0 (<class 'int'>)
Expectation (Save callback execution counter): result = 0 (<class 'int'>)
```

Success	Save callback cache instance is correct (Content None and Type is <class 'NoneType'>).

```
Result (Save callback cache instance): None (<class 'NoneType'>)
Expectation (Save callback cache instance): result = None (<class 'NoneType'>)
```

A.1.15 Callback execution in case of a full update (REQ-0012)

Description
The storage callback shall be called once on every `full_update()`.

Reason for the implementation
Do actions, if cache data is stored to disk.

Fitcriterion
Initialise the cache instance and ensure, that the callback is executed as often as the `.full_update()` method is executed.

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing test instance with save callback and executing full_update.

Success	Save callback execution counter is correct (Content 1 and Type is <class 'int'>).

```
Result (Save callback execution counter): 1 (<class 'int'>)
Expectation (Save callback execution counter): result = 1 (<class 'int'>)
```

Success	Save callback execution counter is correct (Content <caching.property_cache_json object at 0x7fb45a50a850> and Type is <class 'caching.property_cache_json'>).

```
Result (Save callback execution counter): <caching.property_cache_json object at
↪  0x7fb45a50a850> (<class 'caching.property_cache_json'>)
Expectation (Save callback execution counter): result = <caching.property_cache_json object at
↪  0x7fb45a50a850> (<class 'caching.property_cache_json'>)
```

A.1.16 Callback execution in case of get function (REQ-0013)

Description
The storage callback, shall be called once on every `.get()`, if `storage_on_get` is set to `True`.

Reason for the implementation
Do actions, if cache data is stored to disk.

Fitcriterion
Initialise the cache instance and ensure, that the callback is executed as often as the `.get()` method is executed.

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

Info	Prepare: Cleanup cache file before testcase execution

```
Cache file does not exist on filesystem.
```

Info	Preparing test instance with save callback and executing get method 3 times.

Success	Save callback execution counter is correct (Content 3 and Type is <class 'int'>).

```
Result (Save callback execution counter): 3 (<class 'int'>)
Expectation (Save callback execution counter): result = 3 (<class 'int'>)
```

Success	Save callback execution counter is correct (Content <caching.property_cache_json object at 0x7fb45a50ab70> and Type is <class 'caching.property_cache_json'>).

```
Result (Save callback execution counter): <caching.property_cache_json object at
↪  0x7fb45a50ab70> (<class 'caching.property_cache_json'>)
Expectation (Save callback execution counter): result = <caching.property_cache_json object at
↪  0x7fb45a50ab70> (<class 'caching.property_cache_json'>)
```

B Test-Coverage

B.1 caching

The line coverage for caching was 98.8%
The branch coverage for caching was 100.0%

B.1.1 caching.__init__.py

The line coverage for caching.__init__.py was 98.8%
The branch coverage for caching.__init__.py was 100.0%

```
"""
caching (Caching Module)
========================

**Author:**

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

**Description:**

    This Module supports functions and classes for caching e.g. properties of other instances.

**Submodules:**

* :class:`caching.property_cache_json`
* :class:`caching.property_cache_pickle`

**Unittest:**

    See also the :download:`unittest <caching/_testresults_/unittest.pdf>` documentation.
"""

__VERSION__ = "1.2.6"
__DEPENDENCIES__ = ["Python standards"]

import fcntl
import json
import logging
import os
import pickle
import time

try:
    from config import APP_NAME as ROOT_LOGGER_NAME
except ImportError:
    ROOT_LOGGER_NAME = "root"
logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)

__DESCRIPTION__ = """The Module {\\tt %s} is designed to store information in {\\tt json} or {\\tt pickle} files to support them much faster then generating them from the original source file.
For more Information read the documentation.""" % __name__.replace("_", "\\_")
"""The Module Description"""


class property_cache_pickle(object):
    """
    This class caches the data from a given `source_instance`. It takes the data from the cache instead of generating the data from the `source_instance`,
    if the conditions for the cache usage are given.

    .. admonition:: Required properties for the `source_instance`

                * **uid():** returns the unique id of the source's source or None, if you don't want to use the unique id.
                * **keys():** returns a list of all available keys.
```

```
53                * **data_version():** returns a version number of the current data (it should be increased, if the get method of the source instance returns improved values or the data structure had been changed).
54                * **get(key, default):** returns the property for a key. If key does not exists, default will be returned.
55
56    :param source_instance: The source instance holding the data
57    :type source_instance: instance
58    :param cache_filename: File name, where the properties are stored as cache
59    :type cache_filename: str
60    :param load_all_on_init: True will load all data from the source instance, when the cache will be initialised the first time.
61    :type load_all_on_init: bool
62    :param callback_on_data_storage: The callback will be executed every time when the cache file is stored. It will be executed with the instance of this class as first argument.
63    :type callback_on_data_storage: method
64    :param max_age: The maximum age of the cached data in seconds or None for no maximum age.
65    :type max_age: int or None
66    :param store_on_get: False will prevent cache storage with execution of the `.get(key, default)` method. You need to store the cache somewhere else.
67    :type store_on_get: bool
68
69    .. admonition:: The cache will be used, if all following conditions are given
70
71                * The key is in the list returned by `.keys()` method of the `source_instance`
72                * The key is not in the list of keys added by the `.add_source_get_keys()` method.
73                * The cache age is less then the given max_age parameter or the given max_age is None.
74                * The uid of the source instance (e.g. a checksum or unique id of the source) is identically to to uid stored in the cache.
75                * The data version of the `source_instance` is <= the data version stored in the cache.
76                * The value is available in the previous stored information
77
78    **Example:**
79
80    .. literalinclude:: caching/_examples_/property_cache_pickle.py
81
82    Will result on the first execution to the following output (with a long execution time):
83
84    .. literalinclude:: caching/_examples_/property_cache_pickle.log_1st
85
86    With every following execution the time cosumption my by much smaller:
87
88    .. literalinclude:: caching/_examples_/property_cache_pickle.log
89    """
90
91    DATA_VERSION_TAG = "_property_cache_data_version_"
92    STORAGE_VERSION_TAG = "_storage_version_"
93    UID_TAG = "_property_cache_uid_"
94    DATA_TAG = "_data_"
95    AGE_TAG = "_age_"
96    #
97    STORAGE_VERSION = 1
98
99    def __init__(
100       self,
101       source_instance,
102       cache_filename,
103       load_all_on_init=False,
```

```
104         callback_on_data_storage=None,
105         max_age=None,
106         store_on_get=True,
107         return_source_on_none=False,
108     ):
109         self._source_instance = source_instance
110         self._cache_filename = cache_filename
111         self._load_all_on_init = load_all_on_init
112         self._callback_on_data_storage = callback_on_data_storage
113         self._max_age = max_age
114         self._store_on_get = store_on_get
115         self._return_source_on_none = return_source_on_none
116         #
117         self._source_get_keys = []
118         self._cached_props = None
119 
120     def add_source_get_keys(self, keys):
121         """
122         This will add one or more keys to a list of keys which will always be provided by the `source_instance` instead of the cache.
123 
124         :param keys: The key or keys to be added
125         :type keys: list, tuple, str
126         """
127         if type(keys) in [list, tuple]:
128             self._source_get_keys.extend(keys)
129         else:
130             self._source_get_keys.append(keys)
131 
132     def full_update(self, sleep_between_keys=0):
133         """
134         With the execution of this method, the complete source data which needs to be cached, will be read from the source instance
135         and the resulting cache will be stored to the given file.
136 
137         :param sleep_between_keys: Time to sleep between each source data generation
138         :type sleep_between_keys: float, int
139 
140         .. hint:: Use this method, if you initiallised the class with `store_on_get=False`
141         """
142         self._load_source(sleep_between_keys=sleep_between_keys)
143         logger.debug("Storing all source data to cache: %s", repr(self._cached_props[self.DATA_TAG]))
144         self._save_cache()
145 
146     def get(self, key, default=None):
147         """
148         Method to get the cached property. If the key does not exists in the cache or `source_instance`, `default` will be returned.
149 
150         :param key: key for value to get.
151         :param default: value to be returned, if key does not exists.
152         :returns: value for a given key or default value.
153         """
154         # Init cache
155         if self._cached_props is None:
156             self._init_cache()
157         # Load data from cache
158         cache_val = self._cached_props[self.DATA_TAG].get(key, default)
159         cache_tm = self._cached_props[self.AGE_TAG].get(key, default)
```

```
160         #
161         # Conditions for returning cached data
162         #
163         # cached data is available
164         if key in self._cached_props[self.DATA_TAG]:
165             # Max age okay
166             if self._max_age is None or time.time() - cache_tm < self._max_age:
167                 # Allow None return
168                 if cache_val is not None or not self._return_source_on_none:
169                     # Force data from source
170                     if key not in self._source_get_keys:
171                         logger.debug("Returning cached data (%s: %s).", repr(key), repr(cache_val
    ))
172                         return cache_val
173                     else:
174                         logger.debug("No cached value: Key %s is forced from source by
    source_get_keys.", repr(key))
175                 else:
176                     logger.debug("No cached value: Cached value is None and shall be ignored")
177             else:
178                 logger.debug('No cached value for key "%s": Cached value is old.', key)
179         else:
180             logger.debug(
181                 "No cached value: Cache key %s is not in cache. Currently cached keys: %s",
182                 repr(key),
183                 repr(self._cached_props[self.DATA_TAG].keys()),
184             )
185
186         #
187         # Conditions for returning source and storage of source data
188         #
189         source_val = self._source_instance.get(key, default)
190
191         # Storage condition
192         if self._store_on_get and self._store_data(key):
193             logger.debug("Storing source data (%s) to cache.", key)
194             self._cached_props[self.DATA_TAG][key] = source_val
195             self._cached_props[self.AGE_TAG][key] = int(time.time())
196             self._save_cache()
197
198         logger.debug("Returning source data (%s: %s).", repr(key), repr(source_val))
199         return source_val
200
201     def _data_version(self):
202         if self._cached_props is None:
203             return None
204         else:
205             return self._cached_props.get(self.DATA_VERSION_TAG, None)
206
207     def _storage_version(self):
208         if self._cached_props is None:
209             return None
210         else:
211             return self._cached_props.get(self.STORAGE_VERSION_TAG, None)
212
213     def _init_cache(self):
214         load_cache = self._load_cache()
215         uid = self._source_instance.uid() != self._uid()
216         try:
217             data_version = self._source_instance.data_version() > self._data_version()
218         except TypeError:
219             data_version = True
220         try:
221             storage_version = self._storage_version() != self.STORAGE_VERSION
```

```
222             except TypeError:
223                 storage_version = True
224             #
225             if not load_cache or uid or data_version or storage_version:
226                 if load_cache:
227                     if self._uid() is not None and uid:
228                         logger.debug("Source uid changed, purging complete previous cache")
229                     if self._data_version() is not None and data_version:
230                         logger.debug("Data version increased, purging complete  previous cache")
231                     if storage_version:
232                         logger.debug("Storage version changed, purging complete previous cache")
233                 self._cached_props = {self.AGE_TAG: {}, self.DATA_TAG: {}}
234                 if self._load_all_on_init:
235                     self._load_source()
236                 self._cached_props[self.UID_TAG] = self._source_instance.uid()
237                 self._cached_props[self.DATA_VERSION_TAG] = self._source_instance.data_version()
238                 self._cached_props[self.STORAGE_VERSION_TAG] = self.STORAGE_VERSION
239
240     def _load_only(self):
241         with open(self._cache_filename, "rb") as fh:
242             # LOCK BLOCKING
243             fcntl.lockf(fh.fileno(), fcntl.LOCK_SH)
244             # Read data
245             self._cached_props = pickle.load(fh)
246         logger.debug("Loading properties from cache (%s)", self._cache_filename)
247
248     def _load_cache(self):
249         if os.path.exists(self._cache_filename):
250             try:
251                 self._load_only()
252             except:
253                 logger.exception("Exception while loading cache file %s", self._cache_filename)
254             else:
255                 return True
256         else:
257             logger.debug("Cache file does not exists (yet).")
258         return False
259
260     def _store_data(self, key):
261         return key not in self._source_get_keys and key in self._source_instance.keys()
262
263     def _load_source(self, sleep_between_keys=0):
264         if self._cached_props is None:
265             self._init_cache()
266         logger.debug("Loading all data from source - %s", repr(self._source_instance.keys()))
267         for key in self._source_instance.keys():
268             if self._store_data(key):
269                 data = self._source_instance.get(key)
270                 self._cached_props[self.DATA_TAG][key] = data
271                 self._cached_props[self.AGE_TAG][key] = int(time.time())
272                 time.sleep(sleep_between_keys)
273
274     def _save_only(self):
275         if not os.path.exists(self._cache_filename):
276             with open(self._cache_filename, "wb") as fh:
277                 pickle.dump(self._cached_props, fh)  # Write data
278         else:
279             with open(self._cache_filename, "rb+") as fh:
280                 # LOCK BLOCKING
281                 fcntl.lockf(fh.fileno(), fcntl.LOCK_EX)
282                 # Reset file pointer to the start
283                 fh.seek(0)
284                 # Write data
```

```
285                     pickle.dump(self._cached_props, fh)
286                     # Truncate rest of the existing file data
287                     fh.truncate()
288                     fh.flush()
289                     os.fsync(fh.fileno())  # Sync to filehandle
290             logger.debug("cache-file stored (%s)", self._cache_filename)
291
292         def _save_cache(self):
293             self._save_only()
294             if self._callback_on_data_storage is not None:
295                 logger.debug("Executing data storage callback: %s", self._callback_on_data_storage.
    __name__)
296                 self._callback_on_data_storage(self)
297
298         def _uid(self):
299             if self._cached_props is None:
300                 return None
301             else:
302                 return self._cached_props.get(self.UID_TAG, None)
303
304
305 class property_cache_json(property_cache_pickle):
306     """
307     See also parent :py:class:`property_cache_pickle` for detailed information.
308
309     .. important::
310         * This class uses json. You should **only** use keys of type string!
311         * Unicode types are transfered to strings
312
313         See limitations of json.
314
315     **Example:**
316
317     .. literalinclude:: caching/_examples_/property_cache_json.py
318
319     Will result on the first execution to the following output (with a long execution time):
320
321     .. literalinclude:: caching/_examples_/property_cache_json.log_1st
322
323     With every following execution the time cosumption my by much smaller:
324
325     .. literalinclude:: caching/_examples_/property_cache_json.log
326     """
327
328     def _load_only(self):
329         with open(self._cache_filename, "r") as fh:
330             # LOCK_BLOCKING
331             fcntl.lockf(fh.fileno(), fcntl.LOCK_SH)
332             # Write data
333             self._cached_props = json.load(fh)
334         logger.debug("Loading properties from cache (%s)", self._cache_filename)
335
336     def _save_only(self):
337         if not os.path.exists(self._cache_filename):
338             with open(self._cache_filename, "w") as fh:
339                 json.dump(self._cached_props, fh, sort_keys=True, indent=4)  # Write data
340         else:
341             with open(self._cache_filename, "r+") as fh:
342                 fcntl.lockf(fh.fileno(), fcntl.LOCK_EX)  # LOCK_BLOCKING
343                 fh.seek(0)  # Reset file pointer to the start
344                 json.dump(self._cached_props, fh, sort_keys=True, indent=4)  # Write data
345                 fh.truncate()  # Truncate rest of the existing file data
346                 fh.flush()  # Flush
347                 os.fsync(fh.fileno())  # Sync to filehandle
348         logger.debug("cache-file stored (%s)", self._cache_filename)
```