Unittest for `helpers`

May 16, 2026

Contents

1 Test Information

1.1 Testobject Information

The Module helpers is designed to help with some function which don't have an own Module (yet). For more Information read the documentation.

Information	
Name	helpers
State	In test
Version	0.0.4
Dependencies	
	Python standards
	stringtools
	task

1.2 Testdata Information

Information	
Version	0.0.1
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/helpers/unittest/src
System	Linux
Username	dirk

2 Statistic

2.1 Test-Statistic for testrun: Library test

Number of tests	**0**
Number of successfull tests	**0**
Number of possibly failed tests	**0**
Number of failed tests	**0**
Executionlevel	Full Test (all defined tests)
Time consumption	0.000s

2.2 Coverage Statistic

Module- or Filename	Line-Coverage	Branch-Coverage
helpers	32.5%	
helpers.__init__.py	32.5%	

A Trace for testrun: Library test

B Test-Coverage

B.1 helpers

The line coverage for helpers was 32.5%

B.1.1 helpers.__init__.py

The line coverage for helpers.__init__.py was 32.5%

```
#
"""
helpers (Helpers)
=================

**Author:**

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

**Description:**

    This module supports functions and classes which don't have an own Module (yet).

**Submodules:**

* :class:`helpers.continues_statistic`
* :class:`helpers.continues_statistic_multivalue`
* :class:`helpers.direct_socket_client`
* :class:`helpers.direct_socket_server`
* :class:`helpers.ringbuffer`

**Unittest:**

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

**Module Documentation:**

"""

__VERSION__ = "0.0.4"
__DEPENDENCIES__ = ["Python standards", "stringtools", "task"]

import logging
import numbers
import os
import time

import stringtools
import task


try:
    from config import APP_NAME as ROOT_LOGGER_NAME
except ImportError:
    ROOT_LOGGER_NAME = "root"
```

```
46
47 logger = logging.getLogger(ROOT_LOGGER_NAME).getChild(__name__)
48
49
50 __DESCRIPTION__ = """The Module {\\tt %s} is designed to help with some function which don't have
       an own Module (yet).
51 For more Information read the documentation.""" % __name__.replace("_", "\\_")
52 """The Module Description"""
53
54
55 class continues_statistic(dict):
56     """
57     This class stores a statistic for a stream of values. The statistic includes: mean, min, max,
       quantifier.
58
59     .. note:: You can use the mathematic operators "+, +=" to add a value to the statistic.
60
61     :param mean: The mean start value, the start value or None.
62     :type mean: numeric
63     :param min_val: The min start value or None
64     :type min_val: numeric
65     :param max_val: The max start value or None
66     :type max_val: numeric
67     :param quantifier: The quantifier start value or 0
68     :type quantifier: int
69
70     .. note:: You are able to initialise this class in the following ways:
71
72               * Without arguments to get an empty instance
73               * With one single value to get an starting instance
74               * With mean, min_val, max_val, quantifier to initialise an existing statistic
75
76     **Example:**
77
78     .. literalinclude:: helpers/_examples_/continues_statistic.py
79
80     Will result to the following output:
81
82     .. literalinclude:: helpers/_examples_/continues_statistic.log
83     """
84
85     def __init__(self, mean=None, min_val=None, max_val=None, quantifier=0, max_quantifier=None):
86         super().__init__()
87         #
88         if mean is None:
89             self.__init_data__(None, None, None, 0, max_quantifier)
90         elif min_val is not None and max_val is not None and quantifier > 0:
91             self.__init_data__(mean, min_val, max_val, quantifier, max_quantifier)
92         else:
93             self.__init_data__(mean, mean, mean, 1, max_quantifier)
94
95     def __init_data__(self, mean, min_val, max_val, quantifier, max_quantifier):
96         self["quantifier"] = quantifier
97         self["max_quantifier"] = max_quantifier
98         self["max_val"] = max_val
99         self["min_val"] = min_val
100        self["mean"] = mean
101
102    def __min_of_max_quantifier__(self, other):
103        if self.max_quantifier is None or other.max_quantifier is None:
104            return self.max_quantifier or other.max_quantifier
105        else:
```

```
106             return min(self.max_quantifier, other.max_quantifier)
107
108     def __add__(self, other):
109         if self.quantifier == 0:
110             return continues_statistic(other.mean, other.min, other.max, other.quantifier, self.
    __min_of_max_quantifier__(other))
111         elif other.quantifier == 0:
112             return continues_statistic(self.mean, self.min, self.max, self.quantifier, self.
    __min_of_max_quantifier__(other))
113         else:
114             new_quantifier = self.quantifier + other.quantifier
115             return continues_statistic(
116                 mean=(self.mean * self.quantifier + other.mean * other.quantifier) /
    new_quantifier,
117                 min_val=min(self.min, other.min),
118                 max_val=max(self.max, other.max),
119                 quantifier=min(self.__min_of_max_quantifier__(other) or new_quantifier,
    new_quantifier),
120                 max_quantifier=self.__min_of_max_quantifier__(other),
121             )
122
123     @property
124     def mean(self):
125         """
126         The current mean value.
127         """
128         return self.get("mean")
129
130     @property
131     def min(self):
132         """
133         The current min value.
134         """
135         return self.get("min_val")
136
137     @property
138     def max(self):
139         """
140         The current max value.
141         """
142         return self.get("max_val")
143
144     @property
145     def quantifier(self):
146         """
147         The current quantifier (number of values for continues statistic).
148         """
149         return self.get("quantifier")
150
151     @property
152     def max_quantifier(self):
153         """
154         The maximum quantifier (limitation of quantifier).
155         """
156         return self.get("max_quantifier")
157
158     def pop(self):
159         """
160         This pops out the current statistic data and returns these as an instance of :class:`
    helpers.continues_statistic`.
161
162         :returns: The current statistic or None, if no value is available.
163         :rtype: :class:`helpers.continues_statistic` or None
164         """
```

```
165         if self.quantifier == 0:
166             return continues_statistic(max_quantifier=self.max_quantifier)
167         else:
168             rv = continues_statistic(self.mean, self.min, self.max, self.quantifier, self.
    max_quantifier)
169             self.__init_data__(None, None, None, 0, self.max_quantifier)
170             return rv
171 
172     def __str__(self):
173         return "mean=%(mean)s, min=%(min_val)s, max=%(max_val)s, quantifier=%(quantifier)s" %
    self
174 
175 
176 class continues_statistic_multivalue(dict):
177     """
178     This class stores multiple statistics of stream values. The statistic of each value is stored
     as :class:`helpers.continues_statistic`.
179 
180     .. note:: You can use the mathematic operators "+, +=" to add instances.
181 
182     .. note:: You are able to initialise this class in the following ways:
183 
184               * Without arguments to get an empty instance
185               * With a keword argument(s) which will be passed to :class:`continues_statistic`
186               * With a dict equivalent of :class:`continues_statistic_multivalue` to initialise
    an existing statistic
187 
188     **Example:**
189 
190     .. literalinclude:: helpers/_examples_/continues_statistic_multivalue.py
191 
192     Will result to the following output:
193 
194     .. literalinclude:: helpers/_examples_/continues_statistic_multivalue.log
195     """
196 
197     def __init__(self, *args, **kwargs):
198         dict.__init__(self)
199         if len(args) == 0 and len(kwargs) >= 0:
200             for key in kwargs:
201                 if type(kwargs[key]) is continues_statistic:
202                     self[key] = kwargs[key]
203                 elif type(kwargs[key]) is dict:
204                     self[key] = continues_statistic(**kwargs[key])
205                 else:
206                     self[key] = continues_statistic(kwargs[key])
207         elif len(args) == 1 and len(kwargs) == 0:
208             for key in args[0]:
209                 self[key] = continues_statistic(**args[0][key])
210         else:
211             raise TypeError("No valid initialisation value(s)!")
212 
213     def __add__(self, other):
214         rv_dict = {}
215         for key in set(list(self.keys()) + list(other.keys())):
216             rv_dict[key] = self.get(key, continues_statistic()) + other.get(key,
    continues_statistic())
217         return continues_statistic_multivalue(**rv_dict)
218 
219     def pop(self, key=None):
```

```
220         """
221         This pops out the current statistic data for a given key and returns these as an instance
        of :class:`helpers.continues_statistic`.
222 
223         If no key is given it pops out the current statistic data and returns these as an
      instance of :class:`helpers.continues_statistic_multiple`.
224 
225         :param key: The key to get data for
226         :type key: str or NoneType
227 
228         :returns: The current statistic or None, if no value is available.
229         :rtype: :class:`helpers.continues_statistic` or None
230         """
231         if key is None:
232             if len(self) == 0:
233                 return continues_statistic_multivalue()
234             else:
235                 rv = continues_statistic_multivalue(**self)
236                 self.clear()
237                 return rv
238         else:
239             return self[key].pop()
240 
241     def __str__(self):
242         if len(self) == 0:
243             return "-"
244         else:
245             return "\n".join([key + ": " + str(self[key]) for key in self])
246 
247 
248 class direct_socket_base(object):
249     """
250     This is the base class for other classes in this module.
251     """
252 
253     DEFAULT_CHANNEL_NAME = "all_others"
254     IS_CLIENT = False
255 
256     def __init__(self, max_len=None, virtual_rate_bps=None):
257         self.__max_length__ = max_len
258         self.__rate_bps__ = virtual_rate_bps
259         #
260         self.init_channel_name()
261         self.__queue__ = task.threaded_queue()
262         self.__queue__.run()
263         #
264         self.__remote_socket__ = None
265         self.__last_remote_socket__ = None
266         self.__data_callback__ = None
267         self.__connect_callback__ = None
268         self.__disconnect_callback__ = None
269         #
270         self.__clean_buffer__()
271 
272     def __chunks__(self, data):
273         chunks = []
274         if self.__max_length__ is None:
275             chunks.append(data)
276         else:
277             for i in range(0, len(data), self.__max_length__):
278                 chunks.append(data[i : i + self.__max_length__])
279         return chunks
280 
```

```
281     def __clean_buffer__(self):
282         self.__rx_buffer__ = b""
283         self.logger.debug("%s Cleaning up receive-buffer", self.__log_prefix__())
284 
285     def __connect__(self, remote_socket):
286         if self.__remote_socket__ is None:
287             self.__remote_socket__ = remote_socket
288             self.logger.info("%s Connection established...", self.__log_prefix__())
289             self.__clean_buffer__()
290             if self.__connect_callback__ is not None:
291                 self.__connect_callback__()
292             remote_socket.__connect__(self)
293 
294     def __log_prefix__(self):
295         return "comm-client:" if self.IS_CLIENT else "comm-server:"
296 
297     def __rx__(self, data):
298         self.__rx_buffer__ += data
299         self.logger.info("%s RX <- %s", self.__log_prefix__(), stringtools.hexlify(data))
300         if self.__data_callback__ is not None:
301             self.__data_callback__(self)
302 
303     def __tx__(self, q, data):
304         self.logger.info("%s TX -> %s", self.__log_prefix__(), stringtools.hexlify(data))
305         if self.__rate_bps__ is not None:
306             time.sleep(len(data) / self.__rate_bps__)
307         self.__remote_socket__.__rx__(data)
308 
309     def disconnect(self):
310         """
311         Method to disconnect client and server.
312         """
313         if self.__remote_socket__ is not None:
314             self.__last_remote_socket__ = self.__remote_socket__
315             self.__remote_socket__ = None
316             self.logger.info("%s Connection Lost...", self.__log_prefix__())
317             if self.__disconnect_callback__ is not None:
318                 self.__disconnect_callback__()
319             self.__last_remote_socket__.disconnect()
320 
321     def init_channel_name(self, channel_name=None):
322         """
323         With this Method, the channel name for logging can be changed.
324 
325         :param channel_name: The name for the logging channel
326         :type channel_name: str
327         """
328         if channel_name is None:
329             self.logger = logger.getChild(self.DEFAULT_CHANNEL_NAME)
330         else:
331             self.logger = logger.getChild(channel_name)
332 
333     def is_connected(self):
334         """
335         With this Method the connection status can be identified.
336 
337         :return: True, if a connection is established, otherwise False.
338         :rtype: bool
339         """
340         return self.__remote_socket__ is not None
341 
342     def receive(self, timeout=1.0, num=None):
```

```
343         """
344         This method returns received data.
345 
346         :param timeout: The timeout for receiving data (at least after the timeout the method
    returns data or None).
347         :type timeout: float
348         :param num: the number of bytes to receive (use None to get all available data).
349         :type num: int or None
350         :return: The received data.
351         :rtype: bytes
352         """
353         i = 0
354         while len(self.__rx_buffer__) < (num or 1) and i < timeout * 10:
355             i += 1
356             time.sleep(0.1)
357         if len(self.__rx_buffer__) < (num or 1):
358             return self.__rx_buffer__[0:0]
359         else:
360             if num is None:
361                 rv = self.__rx_buffer__
362                 self.__rx_buffer__ = rv[0:0]
363             else:
364                 rv = self.__rx_buffer__[:num]
365                 self.__rx_buffer__ = self.__rx_buffer__[num:]
366             return rv
367 
368     def register_callback(self, callback):
369         """
370         This method stores the callback which is executed, if data is available. You need to
    execute :func:`receive` of this instance
371         given as first argument.
372 
373         :param callback: The callback which will be executed, when data is available.
374         :type callback:
375         """
376         self.__data_callback__ = callback
377 
378     def register_connect_callback(self, callback):
379         """
380         This method stores the callback which is executed, if a connection is established.
381 
382         :param callback: The callback which will be executed, when a connection is established.
383         :type callback:
384         """
385         self.__connect_callback__ = callback
386 
387     def register_disconnect_callback(self, callback):
388         """
389         This method stores the callback which is executed, after the connection is lost.
390 
391         :param callback: The callback which will be executed, after the connection is lost.
392         :type callback:
393         """
394         self.__disconnect_callback__ = callback
395 
396     def send(self, data, timeout=1.0):
397         """
398         This method sends data via the initiated communication channel.
399 
400         :param data: The data to be send over the communication channel.
401         :type data: bytes
402         :param timeout: The timeout for sending data (e.g. time to establish new connection).
403         :type timeout: float
404         :return: True if data had been sent, otherwise False.
405         :rtype: bool
406         """
```

```
407         i = 0
408         while not self.is_connected() and i < timeout * 10:
409             i += 1
410             time.sleep(0.1)
411         if not self.is_connected():
412             return False
413         else:
414             for tx_data in self.__chunks__(data):
415                 self.__queue__.enqueue(1, self.__tx__, tx_data)
416         return True
417 
418 
419 class direct_socket_stp_base(direct_socket_base):
420     IS_CLIENT = False
421 
422     def __init__(self, *args, **kwargs):
423         direct_socket_base.__init__(self, *args, **kwargs)
424         self.__stp_rx__ = stringtools.stp.stp()
425 
426     def __chunks__(self, data):
427         return direct_socket_base.__chunks__(self, stringtools.stp.build_frame(data))
428 
429     def __clean_buffer__(self):
430         self.__rx_buffer__ = []
431         self.logger.debug("%s Cleaning up receive-buffer", self.__log_prefix__())
432 
433     def __rx__(self, data):
434         self.logger.debug("%s RX <- %s", self.__log_prefix__(), stringtools.hexlify(data))
435         msg = self.__stp_rx__.process(data)
436         if len(msg) > 0:
437             self.__rx_buffer__.extend(msg)
438         if len(self.__rx_buffer__) > 0:
439             if self.__data_callback__ is not None:
440                 self.__data_callback__(self)
441 
442     def receive(self, timeout=1):
443         """
444         This method returns one received messages via the initiated communication channel.
445 
446         :param timeout: The timeout for receiving data (at least after the timeout the method
    returns data or None).
447         :type timeout: float
448         :return: The received data.
449         :rtype: bytes
450         """
451         try:
452             return direct_socket_base.receive(self, timeout=timeout, num=1)[0]
453         except TypeError:
454             return None
455 
456 
457 class direct_socket_client(direct_socket_base):
458     """
459     Class to create a direct client socket. See also parent :class:`helpers.direct_socket_base`.
460 
461     **Example:**
462 
463     .. literalinclude:: helpers/_examples_/direct_socket.py
464 
465     Will result to the following output:
466 
467     .. literalinclude:: helpers/_examples_/direct_socket.log
468     """
469 
```

```
470     IS_CLIENT = True
471
472     def connect(self, remote_socket):
473         """
474         Method to create a connection between this client and a :class:`helpers.
    direct_socket_server` instance.
475
476         :param remote_socket: The remote socket to connect to.
477         :type remote_socket: :class:`helpers.direct_socket_server`
478         """
479         self.__connect__(remote_socket)
480
481     def reconnect(self):
482         """
483         Method to do a reconnect.
484
485         .. note:: The :const:`remote_socket` of the prefious :func:`connect` call will be used.
486         """
487         if self.__last_remote_socket__ is not None and self.__remote_socket__ is None:
488             self.connect(self.__last_remote_socket__)
489             return True
490         return False
491
492
493 class direct_socket_server(direct_socket_base):
494     """
495     Class to create a direct server socket. See also parent :class:`helpers.direct_socket_base`.
496
497     **Example:**
498
499     .. literalinclude:: helpers/_examples_/direct_socket.py
500
501     Will result to the following output:
502
503     .. literalinclude:: helpers/_examples_/direct_socket.log
504     """
505
506     IS_CLIENT = False
507
508     def __init__(self, *args, **kwargs):
509         direct_socket_base.__init__(self, *args, **kwargs)
510         self.logger.info("%s Waiting for incomming connection", self.__log_prefix__())
511
512
513 class direct_socket_stp_client(direct_socket_stp_base):
514     IS_CLIENT = True
515
516     def connect(self, remote_socket):
517         """
518         Method to create a connection between this client and a :class:`helpers.
    direct_socket_server` instance.
519
520         :param remote_socket: The remote socket to connect to.
521         :type remote_socket: :class:`helpers.direct_socket_server`
522         """
523         self.__connect__(remote_socket)
524
525     def reconnect(self):
526         """
527         Method to do a reconnect.
528
529         .. note:: The :const:`remote_socket` of the prefious :func:`connect` call will be used.
530         """
```

```
531         if self.__last_remote_socket__ is not None and self.__remote_socket__ is None:
532             self.connect(self.__last_remote_socket__)
533             return True
534         return False
535 
536 
537 class direct_socket_stp_server(direct_socket_stp_base):
538     IS_CLIENT = False
539 
540     def __init__(self, *args, **kwargs):
541         direct_socket_stp_base.__init__(self, *args, **kwargs)
542         self.logger.info("%s Waiting for incomming connection", self.__log_prefix__())
543 
544 
545 class ringbuffer(list):
546     """
547     Class for a list with a limited number of elements.
548 
549     .. note:: On adding Objects, the list will be reduced to the maximum length by deleting
    entries starting from index 0.
550 
551     **Example:**
552 
553     .. literalinclude:: helpers/_examples_/ringbuffer.py
554 
555     Will result to the following output:
556 
557     .. literalinclude:: helpers/_examples_/ringbuffer.log
558     """
559 
560     def __init__(self, *args, **kwargs):
561         self.__max_length__ = kwargs.pop("length")
562         list.__init__(self, *args, **kwargs)
563         self.__reduce_list__()
564 
565     def __reduce_list__(self):
566         while len(self) > self.__max_length__:
567             self.pop(0)
568 
569     def append(self, obj):
570         rv = list.append(self, obj)
571         self.__reduce_list__()
572         return rv
573 
574     def extend(self, iterable):
575         rv = list.extend(self, iterable)
576         self.__reduce_list__()
577         return rv
```