
Pyrakoon Documentation

Release 0.0.1

Nicolas Trangez

December 08, 2015

1	Usage	1
1.1	Basic Usage	1
1.2	Twisted Support	2
2	API	3
2.1	pyrakoon	3
2.2	pyrakoon.client	3
2.3	pyrakoon.client.admin	9
2.4	pyrakoon.errors	10
2.5	pyrakoon.sequence	12
2.6	pyrakoon.tx	13
2.7	pyrakoon.test	13
2.8	pyrakoon.utils	14
2.9	pyrakoon.protocol	17
2.10	pyrakoon.protocol.admin	30
2.11	pyrakoon.client.utils	31
3	Indices and tables	33
Python Module Index		35

Usage

1.1 Basic Usage

The *pyrakoon* library provides the building blocks to create an Arakoon client library in Python. It contains the required data types used in the client protocol, including serialization and parsing routines. It also includes descriptions of the available operations.

The code is not bound to a specific method of communicating with Arakoon nodes though. This is abstracted, and left up to the user to implement according to specific needs and communication mechanisms.

1.1.1 Socket Handling

To provide an implementation of the abstract communication mechanism, the *AbstractClient* interface must be fulfilled. A very basic implementation, which can be used to communicate with a single Arakoon node (i.e. no master failover or reconnection is provided) using blocking socket calls is provided by *SocketClient*.

Warning: *SocketClient* should only be used for (manual) testing purposes. Due to the lack of good exception handling, timeouts,... it should not be used in real-world code.

1.1.2 Mixins

When given an *AbstractClient* implementation, this doesn't give access to actual client operations. Whilst it's possible to create instances of the calls as defined in *pyrakoon.protocol* and related modules and pass these through *_process()*, this is rather clumsy.

To provide a uniform interface, not bound to a specific *AbstractClient* implementation, a couple of mixins are provided, which expose client operations in a user-friendly way. Several mixins are available:

- *pyrakoon.client.ClientMixin* for standard client operations.
- *pyrakoon.client.admin.ClientMixin* for administrative operations.

These can be combined with an *AbstractClient* implementation and used as-is. Here's an example using *SocketClient*, mixing in *pyrakoon.client.ClientMixin* and *pyrakoon.client.admin.ClientMixin*:

```
>>> from pyrakoon import client
>>> from pyrakoon.client import admin

>>> class Client(client.SocketClient, client.ClientMixin, admin.ClientMixin):
...     '''An Arakoon client'''

>>> c = Client('localhost', 4000), 'ricky')
>>> c.connect()
```

```
>>> c.set('key', 'value') # from client.ClientMixin
>>> c.collapse_tlogs(4) # from admin.ClientMixin
[]
```

1.2 Twisted Support

pyrakoon comes with an `AbstractClient` implementation supporting the Twisted framework, provided as a Protocol in the `pyrakoon.tx` module.

Here's a demonstration of how it could be used:

```
>>> from twisted.internet import defer, endpoints, reactor
>>> from pyrakoon import client, tx

>>> class Protocol(tx.ArakoonProtocol, client.ClientMixin): pass

>>> @defer.inlineCallbacks
... def connected(proto):
...     yield proto.set('key', 'value')
...     value = yield proto.get('key')
...     print 'Value:', value
...     yield proto.delete('key')
...     reactor.stop()

>>> endpoint = endpoints.TCP4ClientEndpoint(reactor, 'localhost', 4000)
>>> d = endpoints.connectProtocol(endpoint, Protocol('ricky'))
>>> d.addCallback(connected)
<Deferred at ...>
>>> reactor.run()
Value: value
```

API

<code>pyrakoon</code>	pyrakoon, an Arakoon client for Python
<code>pyrakoon.client</code>	Arakoon client interface
<code>pyrakoon.client.admin</code>	Administrative client interface
<code>pyrakoon.errors</code>	Exceptions raised by client operations, as returned by a node
<code>pyrakoon.sequence</code>	Sequence implementation
<code>pyrakoon.tx</code>	
<code>pyrakoon.test</code>	Testing utilities
<code>pyrakoon.utils</code>	Utility functions
<code>pyrakoon.protocol</code>	Arakoon protocol implementation
<code>pyrakoon.protocol.admin</code>	Arakoon administrative call implementations
<code>pyrakoon.client.utils</code>	Utility functions for building client mixins

2.1 pyrakoon

pyrakoon, an Arakoon client for Python

2.2 pyrakoon.client

Arakoon client interface

class `pyrakoon.client.ClientMixin`

Mixin providing client actions for standard cluster functionality

This can be mixed into any class implementing `AbstractClient`.

hello (`client_id`, `cluster_id`)

Send a “hello” command to the server

This method will return the string returned by the server when receiving a “hello” command.

Parameters

- **client_id** (`str`) – Identifier of the client
- **cluster_id** (`str`) – Identifier of the cluster connecting to. This must match the cluster configuration.

Returns Message returned by the server

Return type `str`

exists (`key`, `allow_dirty=False`)

Send an “exists” command to the server

This method returns a boolean which tells whether the given `key` is set on the server.

Parameters

- **key** (`str`) – Key to test
- **allow_dirty** (`bool`) – Allow reads from slave nodes

Returns Whether the given key is set on the server

Return type `bool`

`who_master()`

Send a “who_master” command to the server

This method returns the name of the current master node in the Arakoon cluster.

Returns Name of cluster master node

Return type `str`

`get(key, allow_dirty=False)`

Send a “get” command to the server

This method returns the value of the requested key.

Parameters

- **key** (`str`) – Key to retrieve
- **allow_dirty** (`bool`) – Allow reads from slave nodes

Returns Value for the given key

Return type `str`

`set(key, value)`

Send a “set” command to the server

This method sets a given key to a given value on the server.

Parameters

- **key** (`str`) – Key to set
- **value** (`str`) – Value to set

`delete(key)`

Send a “delete” command to the server

This method deletes a given key from the cluster.

Parameters `key` (`str`) – Key to delete

`prefix(prefix, max_elements=-1, allow_dirty=False)`

Send a “prefix_keys” command to the server

This method retrieves a list of keys from the cluster matching a given prefix. A maximum number of returned keys can be provided. If set to `-1` (the default), all matching keys will be returned.

Parameters

- **prefix** (`str`) – Prefix to match
- **max_elements** (`int`) – Maximum number of keys to return
- **allow_dirty** (`bool`) – Allow reads from slave nodes

Returns Keys matching the given prefix

Return type iterable of `str`

`test_and_set(key, test_value, set_value)`

Send a “test_and_set” command to the server

When `test_value` is not `None`, the value for `key` will only be modified if the existing value on the server is equal to `test_value`. When `test_value` is `None`, the `key` will only be set if there was no value set for the `key` before.

When `set_value` is `None`, the `key` will be deleted on the server.

The original value for `key` is returned.

Parameters

- `key` (`str`) – Key to act on
- `test_value` (`str` or `None`) – Expected value to test for
- `set_value` (`str` or `None`) – New value to set

Returns Original value of `key`

Return type `str`

`sequence(steps, sync=False)`

Send a “sequence” or “synced_sequence” command to the server

The operations passed to the constructor should be instances of implementations of the `pyrakoon.sequence.Step` class. These operations will be executed in an all-or-nothing transaction.

Parameters

- `steps` (iterable of `pyrakoon.sequence.Step`) – Steps to execute
- `sync` (`bool`) – Use `synced_sequence`

`range(begin_key, begin_inclusive, end_key, end_inclusive, max_elements=-1, allow_dirty=False)`

Send a “range” command to the server

The operation will return a list of keys, in the range between `begin_key` and `end_key`. The `begin_inclusive` and `end_inclusive` flags denote whether the delimiters should be included.

The `max_elements` flag can limit the number of returned keys. If it is negative, all matching keys are returned.

Parameters

- `begin_key` (`str`) – Begin of range
- `begin_inclusive` (`bool`) – `begin_key` is in- or exclusive
- `end_key` (`str`) – End of range
- `end_inclusive` – `end_key` is in- or exclusive
- `max_elements` (`int`) – Maximum number of keys to return
- `allow_dirty` (`bool`) – Allow reads from slave nodes

Returns List of matching keys

Return type iterable of `str`

`range_entries(begin_key, begin_inclusive, end_key, end_inclusive, max_elements=-1, allow_dirty=False)`

Send a “range_entries” command to the server

The operation will return a list of (key, value) tuples, for keys in the range between `begin_key` and `end_key`. The `begin_inclusive` and `end_inclusive` flags denote whether the delimiters should be included.

The `max_elements` flag can limit the number of returned items. If it is negative, all matching items are returned.

Parameters

- **begin_key** (`str`) – Begin of range
- **begin_inclusive** (`bool`) – *begin_key* is in- or exclusive
- **end_key** (`str`) – End of range
- **end_inclusive** (`bool`) – *end_key* is in- or exclusive
- **max_elements** (`int`) – Maximum number of items to return
- **allow_dirty** (`bool`) – Allow reads from slave nodes

Returns List of matching (key, value) pairs

Return type iterable of (`str`, `str`)

multi_get (`keys`, `allow_dirty=False`)

Send a “multi_get” command to the server

This method returns a list of the values for all requested keys.

Parameters

- **keys** (iterable of `str`) – Keys to look up
- **allow_dirty** (`bool`) – Allow reads from slave nodes

Returns Requested values

Return type iterable of `str`

multi_get_option (`keys`, `allow_dirty=False`)

Send a “multi_get_option” command to the server

This method returns a list of value options for all requested keys.

Parameters

- **keys** (iterable of `str`) – Keys to look up
- **allow_dirty** (`bool`) – Allow reads from slave nodes

Returns Requested values

Return type iterable of (`str` or `None`)

expect_progress_possible ()

Send a “expect_progress_possible” command to the server

This method returns whether the master thinks progress is possible.

Returns Whether the master thinks progress is possible

Return type `bool`

get_key_count ()

Send a “get_key_count” command to the server

This method returns the number of items stored in Arakoon.

Returns Number of items stored in the database

Return type `int`

user_function (`function`, `argument`)

Send a “user_function” command to the server

This method returns the result of the function invocation.

Parameters

- **function** (`str`) – Name of the user function to invoke
- **argument** (`str` or `None`) – Argument to pass to the function

Returns Result of the function invocation

Return type `str or None`

confirm(*key, value*)

Send a “confirm” command to the server

This method sets a given key to a given value on the server, unless the value bound to the key is already equal to the provided value, in which case the action becomes a no-op.

Parameters

- **key** (`str`) – Key to set
- **value** (`str`) – Value to set

assert_(*key, value, allow_dirty=False*)

Send an “assert” command to the server

assert key vo throws an exception if the value associated with the key is not what was expected.

Parameters

- **key** (`str`) – Key to check
- **value** (`str or None`) – Optional value to compare
- **allow_dirty** (`bool`) – Allow reads from slave nodes

rev_range_entries(*begin_key, begin_inclusive, end_key, end_inclusive, max_elements=-1, allow_dirty=False*)

Send a “rev_range_entries” command to the server

The operation will return a list of (key, value) tuples, for keys in the reverse range between *begin_key* and *end_key*. The *begin_inclusive* and *end_inclusive* flags denote whether the delimiters should be included.

The *max_elements* flag can limit the number of returned items. If it is negative, all matching items are returned.

Parameters

- **begin_key** (`str`) – Begin of range
- **begin_inclusive** (`bool`) – *begin_key* is in- or exclusive
- **end_key** (`str`) – End of range
- **end_inclusive** (`bool`) – *end_key* is in- or exclusive
- **max_elements** (`int`) – Maximum number of items to return
- **allow_dirty** (`bool`) – Allow reads from slave nodes

Returns List of matching (key, value) pairs

Return type iterable of (`str; str`)

statistics()

Send a “statistics” command to the server

This method returns some server statistics.

Returns Server statistics

Return type *Statistics*

version()

Send a “version” command to the server

This method returns the server version.

Returns Server version

Return type (`int, int, int, str`)

assert_exists (*key*, *allow_dirty=False*)

Send an “assert_exists” command to the server

assert_exists key throws an exception if the key doesn’t exist in the database.

Parameters

- **key** (`str`) – Key to check
- **allow_dirty** (`bool`) – Allow reads from slave nodes

delete_prefix (*prefix*)

Send a “delete_prefix” command to the server

delete_prefix prefix will delete all key/value-pairs from the database where given *prefix* is a prefix of *key*.

Parameters **prefix** (`str`) – Prefix of binding keys to delete

Returns Number of deleted bindings

Return type `int`

replace (*key*, *value*)

Send a “replace” command to the server

replace key value will replace the value bound to the given key with the provided value, and return the old value bound to the key. If *value* is `None`, the key is deleted. If the key was not present in the database, `None` is returned.

Parameters

- **key** (`str`) – Key to replace
- **value** (`str` or `None`) – Value to set

Returns Original value bound to the key

Return type `str` or `None`

nop ()

Send a “nop” command to the server

This enforces consensus throughout a cluster, but has no further effects.

get_current_state ()

Send a “get_current_state” command to the server

This call returns a string representing the current state of the node, and can be used for troubleshooting purposes.

Returns State of the server

Return type `str`

exception `pyrakoon.client.NotConnectedError`

Bases: `exceptions.RuntimeError`

Error used when a call on a not-connected client is made

class `pyrakoon.client.AbstractClient`

Abstract base class for implementations of Arakoon clients

connected = False

Flag to denote whether the client is connected

If this is `False`, a `NotConnectedError` will be raised when a call is issued.

Type `bool`

_process (message)

Submit a message to the server, parse the result and return it

The given *message* should be serialized using its `serialize()` method and submitted to the server. Then the `receive()` coroutine of the *message* should be used to retrieve and parse a result from the server. The result value should be returned by this method, or any exceptions should be rethrown if caught.

Parameters `message` (`pyrakoon.protocol.Message`) – Message to handle

Returns Server result value

Return type `object`

See `pyrakoon.utils.process_blocking()`

class pyrakoon.client.SocketClient (address, cluster_id)

Bases: `object`, `pyrakoon.client.AbstractClient`

Arakoon client using TCP to contact a cluster node

Warning Due to the lack of resource and exception management, this is not intended to be used in real-world code.

Parameters

- `address` ((`str`, `int`)) – Node address (host & port)
- `cluster_id` (`str`) – Identifier of the cluster

connect ()

Create client socket and connect to server

connected

Check whether a connection is available

_process (message)

2.3 pyrakoon.client.admin

Administrative client interface

class pyrakoon.client.admin.ClientMixin

Mixin providing client actions for node administration

This can be mixed into any class implementing `pyrakoon.client.AbstractClient`.

optimize_db ()

Send a “optimize_db” command to the server

This method will trigger optimization of the store on the node this command is sent to.

Note This only works on slave nodes

defrag_db ()

Send a “defrag_db” command to the server

This method will trigger defragmentation of the store on the node this comamand is sent to.

Note This only works on slave nodes

drop_master ()

Send a “drop_master” command to the server

This method instructs a node to drop its master role, if possible. When the call returns successfully, the node was no longer master, but could have gained the master role already in-between.

Note This doesn’t work in a single-node environment

collapse_tlogs (count)

Send a “collapse_tlogs” command to the server

This method instructs a node to collapse its *TLOG* collection by constructing a *head* database and removing superfluous *TLOG* files.

The number of *TLOG* files to keep should be passed as a parameter.

Parameters `count` (`int`) – Number of *TLOG* files to keep

Returns For every *TLOG*, the time it took to collapse it

Return type `[int]`

flush_store ()

Send a “flush_store” command to the server

This method instructs a node to flush its store to disk.

2.4 pyrakoon.errors

Exceptions raised by client operations, as returned by a node

exception `pyrakoon.errors.ArakoonError`

Bases: `exceptions.Exception`

Base type for all Arakoon client errors

CODE = None

Error code sent by the Arakoon server

exception `pyrakoon.errors.NoMagic`

Bases: `pyrakoon.errors.ArakoonError`

Server received a command without the magic mask

CODE = 1

exception `pyrakoon.errors.TooManyDeadNodes`

Bases: `pyrakoon.errors.ArakoonError`

Too many nodes in the cluster are unavailable to process the request

CODE = 2

exception `pyrakoon.errors.NoHello`

Bases: `pyrakoon.errors.ArakoonError`

No *Hello* message was sent to the server after connecting

CODE = 3

exception `pyrakoon.errors.NotMaster`

Bases: `pyrakoon.errors.ArakoonError`

This node is not a master node

CODE = 4

exception `pyrakoon.errors.NotFound`

Bases: `exceptions.KeyError, pyrakoon.errors.ArakoonError`

Key not found

CODE = 5

exception `pyrakoon.errors.WrongCluster`

Bases: `exceptions.ValueError, pyrakoon.errors.ArakoonError`

Wrong cluster ID passed

CODE = 6

exception `pyrakoon.errors.AssertionFailed`
Bases: `pyrakoon.errors.ArakoonError`
Assertion failed

CODE = 7

exception `pyrakoon.errors.ReadOnly`
Bases: `pyrakoon.errors.ArakoonError`
Node is read-only

CODE = 8

exception `pyrakoon.errors.OutsideInterval`
Bases: `exceptions.ValueError, pyrakoon.errors.ArakoonError`
Request outside interval handled by node

CODE = 9

exception `pyrakoon.errors.GoingDown`
Bases: `pyrakoon.errors.ArakoonError`
Node is going down

CODE = 16

exception `pyrakoon.errors.NotSupported`
Bases: `pyrakoon.errors.ArakoonError`
Unsupported operation

CODE = 32

exception `pyrakoon.errors.NoLongerMaster`
Bases: `pyrakoon.errors.ArakoonError`
No longer master

CODE = 33

exception `pyrakoon.errors.InconsistentRead`
Bases: `pyrakoon.errors.ArakoonError`
Inconsistent read

CODE = 128

exception `pyrakoon.errors.MaxConnections`
Bases: `pyrakoon.errors.ArakoonError`
Connection limit reached

CODE = 254

exception `pyrakoon.errors.UnknownFailure`
Bases: `pyrakoon.errors.ArakoonError`
Unknown failure

CODE = 255

`pyrakoon.errors.ERROR_MAP = {32: <class 'pyrakoon.errors.NotSupported'>, 33: <class 'pyrakoon.errors.NoLongerMaster'>, 128: <class 'pyrakoon.errors.InconsistentRead'>, 254: <class 'pyrakoon.errors.MaxConnections'>, 255: <class 'pyrakoon.errors.UnknownFailure'>}`
Map of Arakoon error codes to exception types

2.5 pyrakoon.sequence

Sequence implementation

class `pyrakoon.sequence.Step` (*args)

Bases: `object`

A step in a sequence operation

TAG = None

Operation command tag

ARGS = None

Argument definition

serialize()

Serialize the operation

Returns Serialized operation

Return type iterable of `str`

class `pyrakoon.sequence.Set` (key, value)

Bases: `pyrakoon.sequence.Step`

“Set” operation

TAG = 1

ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘value’, <pyrakoon.protocol.String object at

key

Key to set

Type `str`

value

Value to set

Type `str`

class `pyrakoon.sequence.Delete` (key)

Bases: `pyrakoon.sequence.Step`

“Delete” operation

TAG = 2

ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>),)

key

Key to delete

Type `str`

class `pyrakoon.sequence.Assert` (key, value)

Bases: `pyrakoon.sequence.Step`

“Assert” operation

TAG = 8

ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘value’, <pyrakoon.protocol.Option object at

key

Key for which to assert the given value

Type `str`

value

Expected value

```

Type str or None

class pyrakoon.sequence.AssertExists (key)
    Bases: pyrakoon.sequence.Step

    “AssertExists” operation

TAG = 15

ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>),)

key
    Key to check

Type str

class pyrakoon.sequence.Sequence (steps)
    Bases: pyrakoon.sequence.Step

    “Sequence” operation

    This is a container for a list of other operations.

TAG = 5

ARGS = ()

steps
    Sequence steps

Type iterable of Step

serialize()

```

2.6 pyrakoon.tx

2.7 pyrakoon.test

Testing utilities

```

class pyrakoon.test.FakeClient
    Bases: object, pyrakoon.client.AbstractClient, pyrakoon.client.ClientMixin

    Fake, in-memory Arakoon client

VERSION = ‘FakeRakoon/0.1’
    Version of the server we fake

MASTER = ‘arakoon0’
    Name of master node

connected = True

class pyrakoon.test.ArakoonEnvironmentMixin
    Test mixin to manage an Arakoon process

setUpArakoon (name, config_template)
    Launch an Arakoon daemon process

Parameters
    • name (str) – Cluster name
    • config_template (str) – Configuration file template

Returns Client configuration tuple, config path and base path

Return type ((str, dict<str, (str, int)>), str, str)

```

```
tearDownArakoon()
    Teardown a managed Arakoon process

class pyrakoon.test.NurseryEnvironmentMixin
    Bases: pyrakoon.test.ArakoonEnvironmentMixin

    Test mixin to manage an Arakoon nursery keeper

    setUpNursery(name, config_template)
        Launch an Arakoon nursery keeper daemon process

        Parameters
            • name (str) – Cluster name
            • config_template (str) – Configuration file template

        Returns Client configuration tuple, config path and base path

        Return type ((str, dict<str, (str, int)>), str, str)

    tearDownNursery()
        Teardown a managed Arakoon nursery keeper process
```

2.8 pyrakoon.utils

Utility functions

```
pyrakoon.utils.LOGGER = <logging.Logger object>
    Logger for code in this module
```

```
pyrakoon.utils.update_argspec(*argnames)
    Wrap a callable to use real argument names
```

When generating functions at runtime, one often needs to fall back to `*args` and `**kwargs` usage. Using these features require well-documented code though, and renders API documentation tools less useful.

The decorator generated by this function wraps a decorated function, which takes `**kwargs`, into a function which takes the given argument names as parameters, and passes them to the decorated function as keyword arguments.

The given argnames can be strings (for normal named arguments), or tuples of a string and a value (for arguments with default values). Only a couple of default value types are supported, an exception will be thrown when an unsupported value type is given.

Example usage:

```
>>> @update_argspec('a', 'b', 'c')
... def fun(**kwargs):
...     return kwargs['a'] + kwargs['b'] + kwargs['c']

>>> import inspect
>>> tuple(inspect.getargspec(fun))
(['a', 'b', 'c'], None, None, None)

>>> print fun(1, 2, 3)
6
>>> print fun(1, c=3, b=2)
6

>>> print fun(1, 2)
Traceback (most recent call last):
...
TypeError: fun() takes exactly 3 arguments (2 given)
```

```
>>> @update_argspec()
... def g():
...     print 'Hello'

>>> tuple(inspect.getargspec(g))
([], None, None, None)

>>> g()
Hello

>>> @update_argspec('name', ('age', None))
... def hello(**kwargs):
...     name = kwargs['name']
...
...     if kwargs['age'] is None:
...         return 'Hello, %s' % name
...     else:
...         age = kwargs['age']
...         return 'Hello, %s, who is %d years old' % (name, age)

>>> tuple(inspect.getargspec(hello))
(['name', 'age'], None, None, (None,))

>>> hello('Nicolas')
'Hello, Nicolas'
>>> hello('Nicolas', 25)
'Hello, Nicolas, who is 25 years old'
```

Parameters `argnames` (iterable of `str` or `(str, object)`) – Names of the arguments to be used

Returns Decorator which wraps a given callable into one with a correct argspec

Return type `callable`

`pyrakoon.utils.format_doc(doc)`

Try to format a docstring

This function will split the given string on line boundaries, strip all lines, and stitch everything back together.

Parameters `doc` (`str` or `unicode`) – Docstring to format

Returns Formatted docstring

Return type `unicode`

`pyrakoon.utils.kill_coroutine(coroutine, log_fun=None)`

Kill a coroutine by injecting `StopIteration`

If the coroutine has exited already, we ignore any errors.

The provided `log_fun` function will be called when an unexpected error occurs. It should take a *message* argument.

Example:

```
>>> import sys
```

```
>>> def f():
...     a = yield 1
...     b = yield 3
...     c = yield 5
```

```
>>> f_ = f()
>>> print f_.next()
1
>>> print f_.send('2')
```

```
3
>>> kill_coroutine(f_)

>>> def incorrect():
...     try:
...         yield 1
...         a = yield 2
...     except:
...         raise Exception
...     yield 3

>>> i = incorrect()
>>> print i.next()
1
>>> kill_coroutine(i,
...     lambda msg: sys.stdout.write('Error: %s' % msg))
Error: Failure while killing coroutine
```

Parameters

- **coroutine** (*generator*) – Coroutine to kill
- **log_fun** (*callable*) – Function to call when an exception is encountered

`pyrakoon.utils.process_blocking(message, stream)`

Process a message using a blocking stream API

The given *message* will be serialized and written to the stream. Once the message was written, the result will be read using `read_blocking()`.

The given stream object should implement *write* and *read* methods, somewhat like the file interface.

Parameters

- **message** (`pyrakoon.protocol.Message`) – Message to process
- **stream** (*object*) – Stream to work on

Returns Result of the command execution

Return type *object*

See `pyrakoon.client.AbstractClient._process()`

See `pyrakoon.protocol.Message.serialize()`

See `pyrakoon.protocol.Message.receive()`

`pyrakoon.utils.read_blocking(receiver, read_fun)`

Process message result parsing using a blocking stream read function

Given a function to read a given amount of bytes from a result channel, this function handles the interaction with the parsing coroutine of a message (as passed to `pyrakoon.client.AbstractClient._process()`).

Parameters

- **receiver** (*generator*) – Message result parser coroutine
- **read_fun** (*callable*) – Callable to read a given number of bytes from a result stream

Returns Message result

Return type *object*

Raises `TypeError` Coroutine didn't return a `Result`

See `pyrakoon.protocol.Message.receive()`

2.9 pyrakoon.protocol

Arakoon protocol implementation

`pyrakoon.protocol.RESULT_SUCCESS = 0`

Success return code

`pyrakoon.protocol.PROTOCOL_VERSION = 1`

Protocol version

`class pyrakoon.protocol.Request (count)`

Bases: `object`

Wrapper for data requests generated by `Type.receive()`

`count`

Number of requested bytes

`class pyrakoon.protocol.Result (value)`

Bases: `object`

Wrapper for value results generated by `Type.receive()`

`value`

Result value

`class pyrakoon.protocol.Type`

Bases: `object`

Base type for Arakoon serializable types

`PACKER = None`

`Struct` instance used by default `serialize()` and `receive()` implementations

`Type struct.Struct`

`check (value)`

Check whether a value is valid for this type

Parameters `value (object)` – Value to test

Returns Whether the value is valid for this type

Return type `bool`

`serialize (value)`

Serialize value

Parameters `value (object)` – Value to serialize

Returns Iterable of bytes of the serialized value

Return type iterable of `str`

`receive ()`

Receive and parse a result from the server

This method is a coroutine which yields `Request` instances, and finally a `Result`. When a `Request` instance is yielded, the number of bytes as specified in the `count` attribute should be sent back.

If finally a `Result` instance is yield, its `value` attribute contains the actual message result.

See `Message.receive()`

`class pyrakoon.protocol.String`

Bases: `pyrakoon.protocol.Type`

String type

`check (value)`

```
    serialize(value)
    receive()

class pyrakoon.protocol.UnsignedInteger(bits, pack)
    Bases: pyrakoon.protocol.Type
    Unsigned integer type
    Initialize an unsigned integer type

    Parameters
        • bits (int) – Bits containing the value
        • pack (str) – Struct type, passed to struct.Struct

    check(value)

class pyrakoon.protocol.SignedInteger(bits, pack)
    Bases: pyrakoon.protocol.Type
    Signed integer type
    Initialize an unsigned integer type

    Parameters
        • bits (int) – Bits containing the value
        • pack (str) – Struct type, passed to struct.Struct

    check(value)

class pyrakoon.protocol.Float
    Bases: pyrakoon.protocol.Type
    Float type
    PACKER = <Struct object>

    check(value)

class pyrakoon.protocol.Bool
    Bases: pyrakoon.protocol.Type
    Bool type
    PACKER = <Struct object>
    TRUE = ‘\x01’
    FALSE = ‘\x00’

    check(value)
    serialize(value)
    receive()

class pyrakoon.protocol.Unit
    Bases: pyrakoon.protocol.Type
    Unit type
    check(value)
    serialize(value)
    receive()

class pyrakoon.protocol.Step
    Bases: pyrakoon.protocol.Type
    Step type
```

```

check (value)
serialize (value)
receive ()

class pyrakoon.protocol.Option (inner_type)
    Bases: pyrakoon.protocol.Type

    Option type

    check (value)
    serialize (value)
    receive ()

class pyrakoon.protocol.List (inner_type)
    Bases: pyrakoon.protocol.Type

    List type

    check (value)
    serialize (value)
    receive ()

class pyrakoon.protocol.Array (inner_type)
    Bases: pyrakoon.protocol.Type

    Array type

    check (value)
    serialize (value)
    receive ()

class pyrakoon.protocol.Product (*inner_types)
    Bases: pyrakoon.protocol.Type

    Product type

    check (value)
    serialize (value)
    receive ()

class pyrakoon.protocol.StatisticsType
    Bases: pyrakoon.protocol.Type

    Statistics type

    check (value)
    serialize (value)
    receive ()

pyrakoon.protocol.ALLOW_DIRTY_ARG = ('allow_dirty', <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False)
    Well-known allow_dirty argument

class pyrakoon.protocol.Message
    Bases: object

    Base type for Arakoon command messages

MASK = 2986278912
    Generic command mask value

TAG = None
    Tag (code) of the command

```

ARGS = None

Arguments required for the command

RETURN_TYPE = None

Return type of the command

DOC = None

Docstring for methods exposing this command

serialize()

Serialize the command

Returns Iterable of bytes of the serialized version of the command

Return type iterable of `str`

receive()

Read and deserialize the return value of the command

Running as a coroutine, this method can read and parse the server result value once this command has been submitted.

This method yields values of type `Request` to request more data (which should then be injected using the `send()` method of the coroutine). The number of requested bytes is provided in the `count` attribute of the `Request` object.

Finally a `Result` value is generated, which contains the server result in its `value` attribute.

Raises ArakoonError Server returned an error code

See `pyrakoon.utils.process_blocking()`

class pyrakoon.protocol.Hello(client_id, cluster_id)

Bases: `pyrakoon.protocol.Message`

“hello” message

TAG = 2986278913

ARGS = ((‘client_id’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘cluster_id’, <pyrakoon.protocol.String object at 0x7f186d139d50>))

RETURN_TYPE = <pyrakoon.protocol.String object>

DOC = u’\nSend a “hello” command to the server\n\nThis method will return the string returned by the server when the command is received.’

client_id

attrgetter(attr, ...) → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter(‘name’)`, the call `f(r)` returns `r.name`. After `g = attrgetter(‘name’, ‘date’)`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter(‘name.first’, ‘name.last’)`, the call `h(r)` returns `(r.name.first, r.name.last)`.

cluster_id

attrgetter(attr, ...) → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter(‘name’)`, the call `f(r)` returns `r.name`. After `g = attrgetter(‘name’, ‘date’)`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter(‘name.first’, ‘name.last’)`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class pyrakoon.protocol.WhoMaster

Bases: `pyrakoon.protocol.Message`

“who_master” message

TAG = 2986278914

ARGS = ()

RETURN_TYPE = <pyrakoon.protocol.Option object>

DOC = u’\nSend a “who_master” command to the server\n\nThis method returns the name of the current master node.’

```

class pyrakoon.protocol.Exists (allow_dirty, key)
    Bases: pyrakoon.protocol.Message
        “exists” message
    TAG = 2986278919
    ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>))
    RETURN_TYPE = <pyrakoon.protocol.Bool object>
    DOC = u’\nSend an “exists” command to the server\n\nThis method returns a boolean which tells whether the given ‘key’ exists.
    key
        attrgetter(attr, ...) -> attrgetter object
        Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter('name'), the call f(r) returns r.name. After g = attrgetter('name', 'date'), the call g(r) returns (r.name, r.date). After h = attrgetter('name.first', 'name.last'), the call h(r) returns (r.name.first, r.name.last).
    allow_dirty
        attrgetter(attr, ...) -> attrgetter object
        Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter('name'), the call f(r) returns r.name. After g = attrgetter('name', 'date'), the call g(r) returns (r.name, r.date). After h = attrgetter('name.first', 'name.last'), the call h(r) returns (r.name.first, r.name.last).

class pyrakoon.protocol.Get (allow_dirty, key)
    Bases: pyrakoon.protocol.Message
        “get” message
    TAG = 2986278920
    ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>))
    RETURN_TYPE = <pyrakoon.protocol.String object>
    DOC = u’\nSend a “get” command to the server\n\nThis method returns the value of the requested key.\n\n:param key: the key to get\n:type key: str
    allow_dirty
        attrgetter(attr, ...) -> attrgetter object
        Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter('name'), the call f(r) returns r.name. After g = attrgetter('name', 'date'), the call g(r) returns (r.name, r.date). After h = attrgetter('name.first', 'name.last'), the call h(r) returns (r.name.first, r.name.last).
    key
        attrgetter(attr, ...) -> attrgetter object
        Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter('name'), the call f(r) returns r.name. After g = attrgetter('name', 'date'), the call g(r) returns (r.name, r.date). After h = attrgetter('name.first', 'name.last'), the call h(r) returns (r.name.first, r.name.last).

class pyrakoon.protocol.Set (key, value)
    Bases: pyrakoon.protocol.Message
        “set” message
    TAG = 2986278921
    ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘value’, <pyrakoon.protocol.String object at 0x7f186d139d50>))
    RETURN_TYPE = <pyrakoon.protocol.Unit object>
    DOC = u’\nSend a “set” command to the server\n\nThis method sets a given key to a given value on the server.\n\n:param key: the key to set\n:type key: str\n:param value: the value to set\n:type value: str
    key
        attrgetter(attr, ...) -> attrgetter object

```

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

value

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class `pyrakoon.protocol.Delete(key)`
Bases: `pyrakoon.protocol.Message`

“delete” message

TAG = `2986278922`

ARGS = ((`'key'`, <`pyrakoon.protocol.String` object at `0x7f186d139d50`>),)

RETURN_TYPE = <`pyrakoon.protocol.Unit` object>

DOC = u’\nSend a “delete” command to the server\n\nThis method deletes a given key from the cluster.\n\n:param key:

key

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class `pyrakoon.protocol.PrefixKeys(allow_dirty, prefix, max_elements)`

Bases: `pyrakoon.protocol.Message`

“prefix_keys” message

TAG = `2986278924`

ARGS = ((`'allow_dirty'`, <`pyrakoon.protocol.Bool` object at `0x7f186d0c6050`>, `False`), (`'prefix'`, <`pyrakoon.protocol.String` object at `0x7f186d139d50`>))

RETURN_TYPE = <`pyrakoon.protocol.List` object>

DOC = u’\nSend a “prefix_keys” command to the server\n\nThis method retrieves a list of keys from the cluster matching the prefix.

allow_dirty

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

prefix

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

max_elements

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class `pyrakoon.protocol.TestAndSet(key, test_value, set_value)`

Bases: `pyrakoon.protocol.Message`

“test_and_set” message

TAG = `2986278925`

```

ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘test_value’, <pyrakoon.protocol.Option object at 0x7f186d139d50>)
RETURN_TYPE = <pyrakoon.protocol.Option object>
DOC = u’\nSend a “test_and_set” command to the server\n\nWhen ‘test_value’ is not :data:‘None’, the value for ‘key’
key
    attrgetter(attr, ...) -> attrgetter object
    Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

test_value
    attrgetter(attr, ...) -> attrgetter object
    Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

set_value
    attrgetter(attr, ...) -> attrgetter object
    Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

class pyrakoon.protocol.Sequence (steps, sync)
    Bases: pyrakoon.protocol.Message
    “sequence” and “synced_sequence” message
ARGS = ((‘steps’, <pyrakoon.protocol.List object at 0x7f186d0c6b10>), (‘sync’, <pyrakoon.protocol.Bool object at 0x7f186d0c6b10>)
RETURN_TYPE = <pyrakoon.protocol.Unit object>
DOC = u’\nSend a “sequence” or “synced_sequence” command to the server\n\nThe operations passed to the constructor
sequence
    attrgetter(attr, ...) -> attrgetter object
    Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

sync
    attrgetter(attr, ...) -> attrgetter object
    Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

serialize()

class pyrakoon.protocol.Range (allow_dirty, begin_key, begin_inclusive, end_key, end_inclusive, max_elements)
    Bases: pyrakoon.protocol.Message
    “Range” message
TAG = 2986278923
ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘begin_key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘end_key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘max_elements’, <pyrakoon.protocol.Int object at 0x7f186d139d50>))
RETURN_TYPE = <pyrakoon.protocol.List object>
DOC = u’\nSend a “range” command to the server\n\nThe operation will return a list of keys, in the range between\n‘begin_key’ and ‘end_key’
allow_dirty
    attrgetter(attr, ...) -> attrgetter object

```

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

begin_key

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

begin_inclusive

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

end_key

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

end_inclusive

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

max_elements

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class pyrakoon.protocol.RangeEntries (allow_dirty, begin_key, begin_inclusive, end_key, end_inclusive, max_elements)

Bases: `pyrakoon.protocol.Message`

“RangeEntries” message

TAG = 2986278927

ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘begin_key’, <pyrakoon.protocol.

RETURN_TYPE = <pyrakoon.protocol.List object>

DOC = u’\nSend a “range_entries” command to the server\n\nThe operation will return a list of (key, value) tuples, for

allow_dirty

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

begin_key

`attrgetter(attr, ...)` → attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

begin_inclusive

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After $f = \text{attrgetter}(\text{'name'})$, the call $f(r)$ returns $r.name$. After $g = \text{attrgetter}(\text{'name'}, \text{'date'})$, the call $g(r)$ returns $(r.name, r.date)$. After $h = \text{attrgetter}(\text{'name.first'}, \text{'name.last'})$, the call $h(r)$ returns $(r.name.first, r.name.last)$.

end_key

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After $f = \text{attrgetter}(\text{'name'})$, the call $f(r)$ returns $r.name$. After $g = \text{attrgetter}(\text{'name'}, \text{'date'})$, the call $g(r)$ returns $(r.name, r.date)$. After $h = \text{attrgetter}(\text{'name.first'}, \text{'name.last'})$, the call $h(r)$ returns $(r.name.first, r.name.last)$.

end_inclusive

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After $f = \text{attrgetter}(\text{'name'})$, the call $f(r)$ returns $r.name$. After $g = \text{attrgetter}(\text{'name'}, \text{'date'})$, the call $g(r)$ returns $(r.name, r.date)$. After $h = \text{attrgetter}(\text{'name.first'}, \text{'name.last'})$, the call $h(r)$ returns $(r.name.first, r.name.last)$.

max_elements

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After $f = \text{attrgetter}(\text{'name'})$, the call $f(r)$ returns $r.name$. After $g = \text{attrgetter}(\text{'name'}, \text{'date'})$, the call $g(r)$ returns $(r.name, r.date)$. After $h = \text{attrgetter}(\text{'name.first'}, \text{'name.last'})$, the call $h(r)$ returns $(r.name.first, r.name.last)$.

class pyrakoon.protocol.MultiGet (allow_dirty, keys)Bases: *pyrakoon.protocol.Message*

“multi_get” message

TAG = 2986278929**ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘keys’, <pyrakoon.protocol.List object>)****RETURN_TYPE = <pyrakoon.protocol.List object>****DOC = u’\nSend a “multi_get” command to the server\n\nThis method returns a list of the values for all requested key****allow_dirty**

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After $f = \text{attrgetter}(\text{'name'})$, the call $f(r)$ returns $r.name$. After $g = \text{attrgetter}(\text{'name'}, \text{'date'})$, the call $g(r)$ returns $(r.name, r.date)$. After $h = \text{attrgetter}(\text{'name.first'}, \text{'name.last'})$, the call $h(r)$ returns $(r.name.first, r.name.last)$.

keys

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After $f = \text{attrgetter}(\text{'name'})$, the call $f(r)$ returns $r.name$. After $g = \text{attrgetter}(\text{'name'}, \text{'date'})$, the call $g(r)$ returns $(r.name, r.date)$. After $h = \text{attrgetter}(\text{'name.first'}, \text{'name.last'})$, the call $h(r)$ returns $(r.name.first, r.name.last)$.

class pyrakoon.protocol.MultigetOption (allow_dirty, keys)Bases: *pyrakoon.protocol.Message*

“multi_get_option” message

TAG = 2986278961**ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘keys’, <pyrakoon.protocol.List object>)****RETURN_TYPE = <pyrakoon.protocol.Array object>****DOC = u’\nSend a “multi_get_option” command to the server\n\nThis method returns a list of value options for all req**

allow_dirty

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

keys

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class pyrakoon.protocol.ExpectProgressPossible

Bases: `pyrakoon.protocol.Message`

“expect_progress_possible” message

TAG = 2986278930

ARGS = ()

RETURN_TYPE = <pyrakoon.protocol.Bool object>

DOC = u'\nSend a “expect_progress_possible” command to the server\n\nThis method returns whether the master thin

class pyrakoon.protocol.GetKeyCount

Bases: `pyrakoon.protocol.Message`

“get_key_count” message

TAG = 2986278938

ARGS = ()

RETURN_TYPE = <pyrakoon.protocol.UnsignedInteger object>

DOC = u'\nSend a “get_key_count” command to the server\n\nThis method returns the number of items stored in Ara

class pyrakoon.protocol.UserFunction (function, argument)

Bases: `pyrakoon.protocol.Message`

“user_function” message

TAG = 2986278933

ARGS = ((‘function’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘argument’, <pyrakoon.protocol.Option

RETURN_TYPE = <pyrakoon.protocol.Option object>

DOC = u'\nSend a “user_function” command to the server\n\nThis method returns the result of the function invocation

function

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

argument

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class pyrakoon.protocol.Confirm (key, value)

Bases: `pyrakoon.protocol.Message`

“confirm” message

TAG = 2986278940**ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘value’, <pyrakoon.protocol.String object at 0x7f186d139d50>),)**
RETURN_TYPE = <pyrakoon.protocol.Unit object>**DOC = u’\nSend a “confirm” command to the server\n\nThis method sets a given key to a given value on the server, under the given key****key**
attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

value

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

class pyrakoon.protocol.Assert (allow_dirty, key, value)Bases: *pyrakoon.protocol.Message*

“assert” message

TAG = 2986278934**ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>),)**
RETURN_TYPE = <pyrakoon.protocol.Unit object>**DOC = u’\nSend an “assert” command to the server\n\n‘assert key vo’ throws an exception if the value associated with the key does not match the given value****allow_dirty**

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

key

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

value

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

class pyrakoon.protocol.AssertExists (allow_dirty, key)Bases: *pyrakoon.protocol.Message*

“assert_exists” message

TAG = 2986278953**ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>),)**
RETURN_TYPE = <pyrakoon.protocol.Unit object>**DOC = u’\nSend an “assert_exists” command to the server\n\n‘assert_exists key’ throws an exception if the key doesn’t exist**

allow_dirty

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

key

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

class `pyrakoon.protocol.RevRangeEntries(allow_dirty, begin_key, begin_inclusive, end_key, end_inclusive, max_elements)`

Bases: `pyrakoon.protocol.Message`

“rev_range_entries” message

TAG = 2986278947

ARGS = ((‘allow_dirty’, <pyrakoon.protocol.Bool object at 0x7f186d0c6050>, False), (‘begin_key’, <pyrakoon.protocol.

RETURN_TYPE = <pyrakoon.protocol.List object>

DOC = u’\nSend a “rev_range_entries” command to the server\n\nThe operation will return a list of (key, value) tuples

allow_dirty

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

begin_key

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

begin_inclusive

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

end_key

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

end_inclusive

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

max_elements

attrgetter(attr, ...) -> attrgetter object

Return a callable object that fetches the given attribute(s) from its operand. After `f = attrgetter('name')`, the call `f(r)` returns `r.name`. After `g = attrgetter('name', 'date')`, the call `g(r)` returns `(r.name, r.date)`. After `h = attrgetter('name.first', 'name.last')`, the call `h(r)` returns `(r.name.first, r.name.last)`.

```

class pyrakoon.protocol.Statistics
    Bases: pyrakoon.protocol.Message
    “statistics” message
    TAG = 2986278931
    ARGS = ()
    RETURN_TYPE = <pyrakoon.protocol.StatisticsType object>
    DOC = u’\nSend a “statistics” command to the server\n\nThis method returns some server statistics.\n\n:return: Server
class pyrakoon.protocol.Version
    Bases: pyrakoon.protocol.Message
    “version” message
    TAG = 2986278952
    ARGS = ()
    RETURN_TYPE = <pyrakoon.protocol.Product object>
    DOC = u’\nSend a “version” command to the server\n\nThis method returns the server version.\n\n:return: Server ver
class pyrakoon.protocol.DeletePrefix (prefix)
    Bases: pyrakoon.protocol.Message
    “delete_prefix” message
    TAG = 2986278951
    ARGS = ((‘prefix’, <pyrakoon.protocol.String object at 0x7f186d139d50>),)
    RETURN_TYPE = <pyrakoon.protocol.UnsignedInteger object>
    DOC = u’\nSend a “delete_prefix” command to the server\n\n‘delete_prefix prefix’ will delete all key/value-pairs from
    prefix
        attrgetter(attr, ...) -> attrgetter object
        Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’),  

        the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date).  

        After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).
class pyrakoon.protocol.Replace (key, value)
    Bases: pyrakoon.protocol.Message
    “replace” message
    TAG = 2986278963
    ARGS = ((‘key’, <pyrakoon.protocol.String object at 0x7f186d139d50>), (‘value’, <pyrakoon.protocol.Option object at
    RETURN_TYPE = <pyrakoon.protocol.Option object>
    DOC = u’\nSend a “replace” command to the server\n\n‘replace key value’ will replace the value bound to the given ke
    key
        attrgetter(attr, ...) -> attrgetter object
        Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’),  

        the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date).  

        After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).
value
        attrgetter(attr, ...) -> attrgetter object
        Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’),  

        the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date).  

        After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).

```

```
class pyrakoon.protocol.Nop
    Bases: pyrakoon.protocol.Message
    “nop” message
    TAG = 2986278977
    ARGS = ()
    RETURN_TYPE = <pyrakoon.protocol.Unit object>
    DOC = u'\nSend a “nop” command to the server\n\nThis enforces consensus throughout a cluster, but has no further\n'

class pyrakoon.protocol.GetCurrentState
    Bases: pyrakoon.protocol.Message
    “get_current_state” message
    TAG = 2986278962
    ARGS = ()
    RETURN_TYPE = <pyrakoon.protocol.String object>
    DOC = u'\nSend a “get_current_state” command to the server\n\nThis call returns a string representing the current state\n'

pyrakoon.protocol.build_prologue(cluster)
    Return the string to send as prologue

    Parameters cluster (str) – Name of the cluster to which a connection is made
    Returns Prologue to send to the Arakoon server
    Return type str
```

2.10 pyrakoon.protocol.admin

Arakoon administrative call implementations

```
class pyrakoon.protocol.admin.OptimizeDB
    Bases: pyrakoon.protocol.Message
    “optimize_db” message
    TAG = 2986278949
    ARGS = ()
    RETURN_TYPE = <pyrakoon.protocol.Unit object>
    DOC = u'\nSend a “optimize_db” command to the server\n\nThis method will trigger optimization of the store on the\n'

class pyrakoon.protocol.admin.DefragDB
    Bases: pyrakoon.protocol.Message
    “defrag_db” message
    TAG = 2986278950
    ARGS = ()
    RETURN_TYPE = <pyrakoon.protocol.Unit object>
    DOC = u'\nSend a “defrag_db” command to the server\n\nThis method will trigger defragmentation of the store on the\n'

class pyrakoon.protocol.admin.DropMaster
    Bases: pyrakoon.protocol.Message
    “drop_master” message
    TAG = 2986278960
```

```

ARGS = ()
RETURN_TYPE = <pyrakoon.protocol.Unit object>
DOC = u'\nSend a “drop_master” command to the server\n\nThis method instructs a node to drop its master role, if p
class pyrakoon.protocol.admin.CollapseTlogs (count)
    Bases: pyrakoon.protocol.Message
    “collapse_tlogs” message
TAG = 2986278932
ARGS = ((‘count’, <pyrakoon.protocol.SignedInteger object at 0x7f186d139e90>),)
RETURN_TYPE = None
DOC = u'\nSend a “collapse_tlogs” command to the server\n\nThis method instructs a node to collapse its *TLOG* co
count
    attrgetter(attr, ...) → attrgetter object
    Return a callable object that fetches the given attribute(s) from its operand. After f = attrgetter(‘name’), the call f(r) returns r.name. After g = attrgetter(‘name’, ‘date’), the call g(r) returns (r.name, r.date). After h = attrgetter(‘name.first’, ‘name.last’), the call h(r) returns (r.name.first, r.name.last).
receive()
class pyrakoon.protocol.admin.FlushStore
    Bases: pyrakoon.protocol.Message
    “flush_store” message
TAG = 2986278978
ARGS = ()
RETURN_TYPE = <pyrakoon.protocol.Unit object>
DOC = u'\nSend a “flush_store” command to the server\n\nThis method instructs a node to flush its store to disk.\n'

```

2.11 pyrakoon.client.utils

Utility functions for building client mixins

`pyrakoon.client.utils.validate_types(specs, args)`
Validate method call argument types

Parameters

- **specs** (iterable of (*str*; *pyrakoon.protocol.Type*)) – Spec of expected types
- **args** (iterable of *object*) – Argument values

Raises

- **TypeError** – Type of an argument is invalid
- **ValueError** – Value of an argument is invalid

`pyrakoon.client.utils.call(message_type)`

Expose a *Message* as a method on a client

Note If the client method has an *allow_dirty* option (i.e. *pyrakoon.protocol.ALLOW_DIRTY_ARG* is present in the *ARGS* field of *message_type*), this is automatically moved to the back.

Parameters **message_type** (*type*) – Type of the message this method should call

Returns Method which wraps a call to an Arakoon server using given message type

Return type *callable*

Indices and tables

- genindex
- modindex
- search

p

pyrakoon, 3
pyrakoon.client, 3
pyrakoon.client.admin, 9
pyrakoon.client.utils, 31
pyrakoon.errors, 10
pyrakoon.protocol, 17
pyrakoon.protocol.admin, 30
pyrakoon.sequence, 12
pyrakoon.test, 13
pyrakoon.utils, 14

Symbols

_process() (pyrakoon.client.AbstractClient method), 8
_process() (pyrakoon.client.SocketClient method), 9

A

AbstractClient (class in pyrakoon.client), 8
allow_dirty (pyrakoon.protocol.Assert attribute), 27
allow_dirty (pyrakoon.protocol.AssertExists attribute), 27
allow_dirty (pyrakoon.protocol.Exists attribute), 21
allow_dirty (pyrakoon.protocol.Get attribute), 21
allow_dirty (pyrakoon.protocol.MultiGet attribute), 25
allow_dirty (pyrakoon.protocol.MultiGetOption attribute), 25
allow_dirty (pyrakoon.protocol.PrefixKeys attribute), 22
allow_dirty (pyrakoon.protocol.Range attribute), 23
allow_dirty (pyrakoon.protocol.RangeEntries attribute), 24
allow_dirty (pyrakoon.protocol.RevRangeEntries attribute), 28
ALLOW_DIRTY_ARG (in module pyrakoon.protocol), 19
ArakoonEnvironmentMixin (class in pyrakoon.test), 13
ArakoonError, 10
ARGS (pyrakoon.protocol.admin.CollapseTlogs attribute), 31
ARGS (pyrakoon.protocol.admin.DefragDB attribute), 30
ARGS (pyrakoon.protocol.admin.DropMaster attribute), 30
ARGS (pyrakoon.protocol.admin.FlushStore attribute), 31
ARGS (pyrakoon.protocol.admin.OptimizeDB attribute), 30
ARGS (pyrakoon.protocol.Assert attribute), 27
ARGS (pyrakoon.protocol.AssertExists attribute), 27
ARGS (pyrakoon.protocol.Confirm attribute), 27
ARGS (pyrakoon.protocol.Delete attribute), 22
ARGS (pyrakoon.protocol.DeletePrefix attribute), 29
ARGS (pyrakoon.protocol.Exists attribute), 21
ARGS (pyrakoon.protocol.ExpectProgressPossible attribute), 26
ARGS (pyrakoon.protocol.Get attribute), 21

ARGS (pyrakoon.protocol.GetCurrentState attribute), 30
ARGS (pyrakoon.protocol.GetKeyCount attribute), 26
ARGS (pyrakoon.protocol.Hello attribute), 20
ARGS (pyrakoon.protocol.Message attribute), 19
ARGS (pyrakoon.protocol.MultiGet attribute), 25
ARGS (pyrakoon.protocol.MultiGetOption attribute), 25
ARGS (pyrakoon.protocol.Nop attribute), 30
ARGS (pyrakoon.protocol.PrefixKeys attribute), 22
ARGS (pyrakoon.protocol.Range attribute), 23
ARGS (pyrakoon.protocol.RangeEntries attribute), 24
ARGS (pyrakoon.protocol.Replace attribute), 29
ARGS (pyrakoon.protocol.RevRangeEntries attribute), 28
ARGS (pyrakoon.protocol.Sequence attribute), 23
ARGS (pyrakoon.protocol.Set attribute), 21
ARGS (pyrakoon.protocol.Statistics attribute), 29
ARGS (pyrakoon.protocol.TestAndSet attribute), 22
ARGS (pyrakoon.protocol.UserFunction attribute), 26
ARGS (pyrakoon.protocol.Version attribute), 29
ARGS (pyrakoon.protocol.WhoMaster attribute), 20
ARGS (pyrakoon.sequence.Assert attribute), 12
ARGS (pyrakoon.sequence.AssertExists attribute), 13
ARGS (pyrakoon.sequence.Delete attribute), 12
ARGS (pyrakoon.sequence.Sequence attribute), 13
ARGS (pyrakoon.sequence.Set attribute), 12
ARGS (pyrakoon.sequence.Step attribute), 12
argument (pyrakoon.protocol.UserFunction attribute), 26
Array (class in pyrakoon.protocol), 19
Assert (class in pyrakoon.protocol), 27
Assert (class in pyrakoon.sequence), 12
assert_() (pyrakoon.client.ClientMixin method), 7
assert_exists() (pyrakoon.client.ClientMixin method), 7
AssertExists (class in pyrakoon.protocol), 27
AssertExists (class in pyrakoon.sequence), 13
AssertionFailed, 11

B

begin_inclusive (pyrakoon.protocol.Range attribute), 24
begin_inclusive (pyrakoon.protocol.RangeEntries attribute), 24

begin_inclusive (pyrakoon.protocol.RevRangeEntries attribute), 28
begin_key (pyrakoon.protocol.Range attribute), 24
begin_key (pyrakoon.protocol.RangeEntries attribute), 24
begin_key (pyrakoon.protocol.RevRangeEntries attribute), 28
Bool (class in pyrakoon.protocol), 18
build_prologue() (in module pyrakoon.protocol), 30

C

call() (in module pyrakoon.client.utils), 31
check() (pyrakoon.protocol.Array method), 19
check() (pyrakoon.protocol.Bool method), 18
check() (pyrakoon.protocol.Float method), 18
check() (pyrakoon.protocol.List method), 19
check() (pyrakoon.protocol.Option method), 19
check() (pyrakoon.protocol.Product method), 19
check() (pyrakoon.protocol.SignedInteger method), 18
check() (pyrakoon.protocol.StatisticsType method), 19
check() (pyrakoon.protocol.Step method), 18
check() (pyrakoon.protocol.String method), 17
check() (pyrakoon.protocol.Type method), 17
check() (pyrakoon.protocol.Unit method), 18
check() (pyrakoon.protocol.UnsignedInteger method), 18
client_id (pyrakoon.protocol.Hello attribute), 20
ClientMixin (class in pyrakoon.client), 3
ClientMixin (class in pyrakoon.client.admin), 9
cluster_id (pyrakoon.protocol.Hello attribute), 20
CODE (pyrakoon.errors.ArakoonError attribute), 10
CODE (pyrakoon.errors.AssertionFailed attribute), 11
CODE (pyrakoon.errors.GoingDown attribute), 11
CODE (pyrakoon.errors.InconsistentRead attribute), 11
CODE (pyrakoon.errors.MaxConnections attribute), 11
CODE (pyrakoon.errors.NoHello attribute), 10
CODE (pyrakoon.errors.NoLongerMaster attribute), 11
CODE (pyrakoon.errors.NoMagic attribute), 10
CODE (pyrakoon.errors.NotFound attribute), 10
CODE (pyrakoon.errors.NotMaster attribute), 10
CODE (pyrakoon.errors.NotSupported attribute), 11
CODE (pyrakoon.errors.OutsideInterval attribute), 11
CODE (pyrakoon.errors.ReadOnly attribute), 11
CODE (pyrakoon.errors.TooManyDeadNodes attribute), 10
CODE (pyrakoon.errors.UnknownFailure attribute), 11
CODE (pyrakoon.errors.WrongCluster attribute), 10
collapse_tlogs() (pyrakoon.client.admin.ClientMixin method), 9
CollapseTlogs (class in pyrakoon.protocol.admin), 31
Confirm (class in pyrakoon.protocol), 26
confirm() (pyrakoon.client.ClientMixin method), 7
connect() (pyrakoon.client.SocketClient method), 9
connected (pyrakoon.client.AbstractClient attribute), 8
connected (pyrakoon.client.SocketClient attribute), 9
connected (pyrakoon.test.FakeClient attribute), 13
count (pyrakoon.protocol.admin.CollapseTlogs attribute), 31

count (pyrakoon.protocol.Request attribute), 17

D

defrag_db() (pyrakoon.client.admin.ClientMixin method), 9
DefragDB (class in pyrakoon.protocol.admin), 30
Delete (class in pyrakoon.protocol), 22
Delete (class in pyrakoon.sequence), 12
delete() (pyrakoon.client.ClientMixin method), 4
delete_prefix() (pyrakoon.client.ClientMixin method), 8
DeletePrefix (class in pyrakoon.protocol), 29
DOC (pyrakoon.protocol.admin.CollapseTlogs attribute), 31
DOC (pyrakoon.protocol.admin.DefragDB attribute), 30
DOC (pyrakoon.protocol.admin.DropMaster attribute), 31
DOC (pyrakoon.protocol.admin.FlushStore attribute), 31
DOC (pyrakoon.protocol.admin.OptimizeDB attribute), 30
DOC (pyrakoon.protocol.Assert attribute), 27
DOC (pyrakoon.protocol.AssertExists attribute), 27
DOC (pyrakoon.protocol.Confirm attribute), 27
DOC (pyrakoon.protocol.Delete attribute), 22
DOC (pyrakoon.protocol.DeletePrefix attribute), 29
DOC (pyrakoon.protocol.Exists attribute), 21
DOC (pyrakoon.protocol.ExpectProgressPossible attribute), 26
DOC (pyrakoon.protocol.Get attribute), 21
DOC (pyrakoon.protocol.GetCurrentState attribute), 30
DOC (pyrakoon.protocol.GetKeyCount attribute), 26
DOC (pyrakoon.protocol.Hello attribute), 20
DOC (pyrakoon.protocol.Message attribute), 20
DOC (pyrakoon.protocol.MultiGet attribute), 25
DOC (pyrakoon.protocol.MultiGetOption attribute), 25
DOC (pyrakoon.protocol.Nop attribute), 30
DOC (pyrakoon.protocol.PrefixKeys attribute), 22
DOC (pyrakoon.protocol.Range attribute), 23
DOC (pyrakoon.protocol.RangeEntries attribute), 24
DOC (pyrakoon.protocol.Replace attribute), 29
DOC (pyrakoon.protocol.RevRangeEntries attribute), 28
DOC (pyrakoon.protocol.Sequence attribute), 23
DOC (pyrakoon.protocol.Set attribute), 21
DOC (pyrakoon.protocol.Statistics attribute), 29
DOC (pyrakoon.protocol.TestAndSet attribute), 23
DOC (pyrakoon.protocol.UserFunction attribute), 26
DOC (pyrakoon.protocol.Version attribute), 29
DOC (pyrakoon.protocol.WhoMaster attribute), 20
drop_master() (pyrakoon.client.admin.ClientMixin method), 9
DropMaster (class in pyrakoon.protocol.admin), 30

E

end_inclusive (pyrakoon.protocol.Range attribute), 24

end_inclusive (pyrakoon.protocol.RangeEntries attribute), 25
 end_inclusive (pyrakoon.protocol.RevRangeEntries attribute), 28
 end_key (pyrakoon.protocol.Range attribute), 24
 end_key (pyrakoon.protocol.RangeEntries attribute), 25
 end_key (pyrakoon.protocol.RevRangeEntries attribute), 28
 ERROR_MAP (in module pyrakoon.errors), 11
 Exists (class in pyrakoon.protocol), 20
 exists() (pyrakoon.client.ClientMixin method), 3
 expect_progress_possible()
 (pyrakoon.client.ClientMixin method), 6
 ExpectProgressPossible (class in pyrakoon.protocol), 26

F

FakeClient (class in pyrakoon.test), 13
 FALSE (pyrakoon.protocol.Bool attribute), 18
 Float (class in pyrakoon.protocol), 18
 flush_store()
 (pyrakoon.client.admin.ClientMixin method), 10
 FlushStore (class in pyrakoon.protocol.admin), 31
 format_doc() (in module pyrakoon.utils), 15
 function (pyrakoon.protocol.UserFunction attribute), 26

G

Get (class in pyrakoon.protocol), 21
 get() (pyrakoon.client.ClientMixin method), 4
 get_current_state()
 (pyrakoon.client.ClientMixin method), 8
 get_key_count()
 (pyrakoon.client.ClientMixin method), 6
 GetCurrentState (class in pyrakoon.protocol), 30
 GetKeyCount (class in pyrakoon.protocol), 26
 GoingDown, 11

H

Hello (class in pyrakoon.protocol), 20
 hello() (pyrakoon.client.ClientMixin method), 3

I

InconsistentRead, 11

K

key (pyrakoon.protocol.Assert attribute), 27
 key (pyrakoon.protocol.AssertExists attribute), 28
 key (pyrakoon.protocol.Confirm attribute), 27
 key (pyrakoon.protocol.Delete attribute), 22
 key (pyrakoon.protocol.Exists attribute), 21
 key (pyrakoon.protocol.Get attribute), 21
 key (pyrakoon.protocol.Replace attribute), 29
 key (pyrakoon.protocol.Set attribute), 21
 key (pyrakoon.protocol.TestAndSet attribute), 23
 key (pyrakoon.sequence.Assert attribute), 12

key (pyrakoon.sequence.AssertExists attribute), 13
 key (pyrakoon.sequence.Delete attribute), 12
 key (pyrakoon.sequence.Set attribute), 12
 keys (pyrakoon.protocol.MultiGet attribute), 25
 keys (pyrakoon.protocol.MultiGetOption attribute), 26
 kill_coroutine() (in module pyrakoon.utils), 15

L

List (class in pyrakoon.protocol), 19
 LOGGER (in module pyrakoon.utils), 14

M

MASK (pyrakoon.protocol.Message attribute), 19
 MASTER (pyrakoon.test.FakeClient attribute), 13
 max_elements (pyrakoon.protocol.PrefixKeys attribute), 22
 max_elements (pyrakoon.protocol.Range attribute), 24
 max_elements (pyrakoon.protocol.RangeEntries attribute), 25
 max_elements (pyrakoon.protocol.RevRangeEntries attribute), 28
 MaxConnections, 11
 Message (class in pyrakoon.protocol), 19
 multi_get() (pyrakoon.client.ClientMixin method), 6
 multi_get_option()
 (pyrakoon.client.ClientMixin method), 6
 MultiGet (class in pyrakoon.protocol), 25
 MultiGetOption (class in pyrakoon.protocol), 25

N

NoHello, 10
 NoLongerMaster, 11
 NoMagic, 10
 Nop (class in pyrakoon.protocol), 29
 nop() (pyrakoon.client.ClientMixin method), 8
 NotConnectedError, 8
 NotFound, 10
 NotMaster, 10
 NotSupported, 11
 NurseryEnvironmentMixin (class in pyrakoon.test), 14

O

optimize_db()
 (pyrakoon.client.admin.ClientMixin method), 9
 OptimizeDB (class in pyrakoon.protocol.admin), 30
 Option (class in pyrakoon.protocol), 19
 OutsideInterval, 11

P

PACKER (pyrakoon.protocol.Bool attribute), 18
 PACKER (pyrakoon.protocol.Float attribute), 18
 PACKER (pyrakoon.protocol.Type attribute), 17
 prefix (pyrakoon.protocol.DeletePrefix attribute), 29
 prefix (pyrakoon.protocol.PrefixKeys attribute), 22
 prefix() (pyrakoon.client.ClientMixin method), 4
 PrefixKeys (class in pyrakoon.protocol), 22
 process_blocking() (in module pyrakoon.utils), 16

Product (class in pyrakoon.protocol), 19
PROTOCOL_VERSION (in module pyrakoon.protocol), 17
pyrakoon (module), 3
pyrakoon.client (module), 3
pyrakoon.client.admin (module), 9
pyrakoon.client.utils (module), 31
pyrakoon.errors (module), 10
pyrakoon.protocol (module), 17
pyrakoon.protocol.admin (module), 30
pyrakoon.sequence (module), 12
pyrakoon.test (module), 13
pyrakoon.utils (module), 14

R

Range (class in pyrakoon.protocol), 23
range() (pyrakoon.client.ClientMixin method), 5
range_entries() (pyrakoon.client.ClientMixin method), 5
RangeEntries (class in pyrakoon.protocol), 24
read_blocking() (in module pyrakoon.utils), 16
ReadOnly, 11
receive() (pyrakoon.protocol.admin.CollapseTlogs method), 31
receive() (pyrakoon.protocol.Array method), 19
receive() (pyrakoon.protocol.Bool method), 18
receive() (pyrakoon.protocol.List method), 19
receive() (pyrakoon.protocol.Message method), 20
receive() (pyrakoon.protocol.Option method), 19
receive() (pyrakoon.protocol.Product method), 19
receive() (pyrakoon.protocol.StatisticsType method), 19
receive() (pyrakoon.protocol.Step method), 19
receive() (pyrakoon.protocol.String method), 18
receive() (pyrakoon.protocol.Type method), 17
receive() (pyrakoon.protocol.Unit method), 18
Replace (class in pyrakoon.protocol), 29
replace() (pyrakoon.client.ClientMixin method), 8
Request (class in pyrakoon.protocol), 17
Result (class in pyrakoon.protocol), 17
RESULT_SUCCESS (in module pyrakoon.protocol), 17
RETURN_TYPE (pyrakoon.protocol.admin.CollapseTlogs attribute), 31
RETURN_TYPE (pyrakoon.protocol.admin.DefragDB attribute), 30
RETURN_TYPE (pyrakoon.protocol.admin.DropMaster attribute), 31
RETURN_TYPE (pyrakoon.protocol.admin.FlushStore attribute), 31
RETURN_TYPE (pyrakoon.protocol.admin.OptimizeDB attribute), 30
RETURN_TYPE (pyrakoon.protocol.Assert attribute), 27
RETURN_TYPE (pyrakoon.protocol.AssertExists attribute), 27
RETURN_TYPE (pyrakoon.protocol.Confirm attribute), 27

RETURN_TYPE (pyrakoon.protocol.Delete attribute), 22
RETURN_TYPE (pyrakoon.protocol.DeletePrefix attribute), 29
RETURN_TYPE (pyrakoon.protocol.Exists attribute), 21
RETURN_TYPE (pyrakoon.protocol.ExpectProgressPossible attribute), 26
RETURN_TYPE (pyrakoon.protocol.Get attribute), 21
RETURN_TYPE (pyrakoon.protocol.GetCurrentState attribute), 30
RETURN_TYPE (pyrakoon.protocol.GetKeyCount attribute), 26
RETURN_TYPE (pyrakoon.protocol.Hello attribute), 20
RETURN_TYPE (pyrakoon.protocol.Message attribute), 20
RETURN_TYPE (pyrakoon.protocol.MultiGet attribute), 25
RETURN_TYPE (pyrakoon.protocol.MultiGetOption attribute), 25
RETURN_TYPE (pyrakoon.protocol.Nop attribute), 30
RETURN_TYPE (pyrakoon.protocol.PrefixKeys attribute), 22
RETURN_TYPE (pyrakoon.protocol.Range attribute), 23
RETURN_TYPE (pyrakoon.protocol.RangeEntries attribute), 24
RETURN_TYPE (pyrakoon.protocol.Replace attribute), 29
RETURN_TYPE (pyrakoon.protocol.RevRangeEntries attribute), 28
RETURN_TYPE (pyrakoon.protocol.Sequence attribute), 23
RETURN_TYPE (pyrakoon.protocol.Set attribute), 21
RETURN_TYPE (pyrakoon.protocol.Statistics attribute), 29
RETURN_TYPE (pyrakoon.protocol.TestAndSet attribute), 23
RETURN_TYPE (pyrakoon.protocol.UserFunction attribute), 26
RETURN_TYPE (pyrakoon.protocol.Version attribute), 29
RETURN_TYPE (pyrakoon.protocol.WhoMaster attribute), 20
rev_range_entries() (pyrakoon.client.ClientMixin method), 7
RevRangeEntries (class in pyrakoon.protocol), 28

S

Sequence (class in pyrakoon.protocol), 23
Sequence (class in pyrakoon.sequence), 13
sequence (pyrakoon.protocol.Sequence attribute), 23
sequence() (pyrakoon.client.ClientMixin method), 5
serialize() (pyrakoon.protocol.Array method), 19
serialize() (pyrakoon.protocol.Bool method), 18
serialize() (pyrakoon.protocol.List method), 19
serialize() (pyrakoon.protocol.Message method), 20

serialize() (pyrakoon.protocol.Option method), 19
 serialize() (pyrakoon.protocol.Product method), 19
 serialize() (pyrakoon.protocol.Sequence method), 23
 serialize() (pyrakoon.protocol.StatisticsType method), 19
 serialize() (pyrakoon.protocol.Step method), 19
 serialize() (pyrakoon.protocol.String method), 17
 serialize() (pyrakoon.protocol.Type method), 17
 serialize() (pyrakoon.protocol.Unit method), 18
 serialize() (pyrakoon.sequence.Sequence method), 13
 serialize() (pyrakoon.sequence.Step method), 12
 Set (class in pyrakoon.protocol), 21
 Set (class in pyrakoon.sequence), 12
 set() (pyrakoon.client.ClientMixin method), 4
 set_value (pyrakoon.protocol.TestAndSet attribute), 23
 setUpArakoon() (pyrakoon.test.ArakoonEnvironmentMixin method), 13
 setUpNursery() (pyrakoon.test.NurseryEnvironmentMixin method), 14
 SignedInteger (class in pyrakoon.protocol), 18
 SocketClient (class in pyrakoon.client), 9
 Statistics (class in pyrakoon.protocol), 28
 statistics() (pyrakoon.client.ClientMixin method), 7
 StatisticsType (class in pyrakoon.protocol), 19
 Step (class in pyrakoon.protocol), 18
 Step (class in pyrakoon.sequence), 12
 steps (pyrakoon.sequence.Sequence attribute), 13
 String (class in pyrakoon.protocol), 17
 sync (pyrakoon.protocol.Sequence attribute), 23

T

TAG (pyrakoon.protocol.admin.CollapseTlogs attribute), 31
 TAG (pyrakoon.protocol.admin.DefragDB attribute), 30
 TAG (pyrakoon.protocol.admin.DropMaster attribute), 30
 TAG (pyrakoon.protocol.admin.FlushStore attribute), 31
 TAG (pyrakoon.protocol.admin.OptimizeDB attribute), 30
 TAG (pyrakoon.protocol.Assert attribute), 27
 TAG (pyrakoon.protocol.AssertExists attribute), 27
 TAG (pyrakoon.protocol.Confirm attribute), 26
 TAG (pyrakoon.protocol.Delete attribute), 22
 TAG (pyrakoon.protocol.DeletePrefix attribute), 29
 TAG (pyrakoon.protocol.Exists attribute), 21
 TAG (pyrakoon.protocol.ExpectProgressPossible attribute), 26
 TAG (pyrakoon.protocol.Get attribute), 21
 TAG (pyrakoon.protocol.GetCurrentState attribute), 30
 TAG (pyrakoon.protocol.GetKeyCount attribute), 26
 TAG (pyrakoon.protocol.Hello attribute), 20
 TAG (pyrakoon.protocol.Message attribute), 19
 TAG (pyrakoon.protocol.MultiGet attribute), 25
 TAG (pyrakoon.protocol.MultiGetOption attribute), 25
 TAG (pyrakoon.protocol.Nop attribute), 30
 TAG (pyrakoon.protocol.PrefixKeys attribute), 22

TAG (pyrakoon.protocol.Range attribute), 23
 TAG (pyrakoon.protocol.RangeEntries attribute), 24
 TAG (pyrakoon.protocol.Replace attribute), 29
 TAG (pyrakoon.protocol.RevRangeEntries attribute), 28
 TAG (pyrakoon.protocol.Set attribute), 21
 TAG (pyrakoon.protocol.Statistics attribute), 29
 TAG (pyrakoon.protocol.TestAndSet attribute), 22
 TAG (pyrakoon.protocol.UserFunction attribute), 26
 TAG (pyrakoon.protocol.Version attribute), 29
 TAG (pyrakoon.protocol.WhoMaster attribute), 20
 TAG (pyrakoon.sequence.Assert attribute), 12
 TAG (pyrakoon.sequence.AssertExists attribute), 13
 TAG (pyrakoon.sequence.Delete attribute), 12
 TAG (pyrakoon.sequence.Sequence attribute), 13

TearDownArakoon() (pyrakoon.test.ArakoonEnvironmentMixin method), 13
 tearDownNursery() (pyrakoon.test.NurseryEnvironmentMixin method), 14
 test_and_set() (pyrakoon.client.ClientMixin method), 4
 test_value (pyrakoon.protocol.TestAndSet attribute), 23
 TestAndSet (class in pyrakoon.protocol), 22
 TooManyDeadNodes, 10
 TRUE (pyrakoon.protocol.Bool attribute), 18
 Type (class in pyrakoon.protocol), 17

U

Unit (class in pyrakoon.protocol), 18
 UnknownFailure, 11
 UnsignedInteger (class in pyrakoon.protocol), 18
 update_argspec() (in module pyrakoon.utils), 14
 user_function() (pyrakoon.client.ClientMixin method), 6
 UserFunction (class in pyrakoon.protocol), 26

V

validate_types() (in module pyrakoon.client.utils), 31
 value (pyrakoon.protocol.Assert attribute), 27
 value (pyrakoon.protocol.Confirm attribute), 27
 value (pyrakoon.protocol.Replace attribute), 29
 value (pyrakoon.protocol.Result attribute), 17
 value (pyrakoon.protocol.Set attribute), 22
 value (pyrakoon.sequence.Assert attribute), 12
 value (pyrakoon.sequence.Set attribute), 12
 Version (class in pyrakoon.protocol), 29
 VERSION (pyrakoon.test.FakeClient attribute), 13
 version() (pyrakoon.client.ClientMixin method), 7

W

who_master() (pyrakoon.client.ClientMixin method), 4
 WhoMaster (class in pyrakoon.protocol), 20
 WrongCluster, 10