Pyrakoon Documentation

Release 0.0.1

Nicolas Trangez

December 08, 2015

Contents

CHAPTER 1

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

pyrakoon	pyrakoon, an Arakoon client for Python
pyrakoon.client	Arakoon client interface
pyrakoon.client.admin	Administrative client interface
pyrakoon.errors	Exceptions raised by client operations, as returned by a node
pyrakoon.sequence	Sequence implementation
pyrakoon.tx	
pyrakoon.test	Testing utilities
pyrakoon.utils	Utility functions
pyrakoon.protocol	Arakoon protocol implementation
pyrakoon.protocol.admin	Arakoon administrative call implementations
pyrakoon.client.utils	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 of 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.NoLonger**
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

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\n

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

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 'k

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

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

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

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:par

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.Str

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 match

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 obje

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 0x7

RETURN_TYPE = <pyrakoon.protocol.Unit object>

DOC = u'\nSend a "sequence" or "synced_sequence" command to the server\n\nThe operations passed to the construc

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

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'

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 = 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.**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

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 = 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.**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

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 thi

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 invocatio

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

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, un

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

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

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

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

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: Serve

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 st

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

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 th

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*

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

p

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W