aerospike Documentation

Ronen Botzer

May 11, 2021

CONTENTS

1 Content **3**

1.1 `aerospike` — Aerospike Client for Python . . . 3

1.1.1 Methods . . . 3

1.1.2 Operators . . . 13

1.1.3 Policy Options . . . 13

1.1.3.1 Commit Level Policy Options . . . 13

1.1.3.2 AP Read Mode Policy Options . . . 13

1.1.3.3 SC Read Mode Policy Options . . . 14

1.1.3.4 Existence Policy Options . . . 14

1.1.3.5 Generation Policy Options . . . 14

1.1.3.6 Key Policy Options . . . 15

1.1.3.7 Replica Options . . . 15

1.1.3.8 Retry Policy Options . . . 15

1.1.4 Constants . . . 15

1.1.4.1 TTL Constants . . . 15

1.1.4.2 Auth Mode Constants . . . 16

1.1.4.3 Scan Constants . . . 16

1.1.4.4 Job Constants . . . 16

1.1.4.5 Job Statuses . . . 16

1.1.4.6 Serialization Constants . . . 17

1.1.4.7 Send Bool Constants . . . 17

1.1.4.8 List Write Flags . . . 17

1.1.4.9 List Return Types . . . 18

1.1.4.10 List Order . . . 18

1.1.4.11 List Sort Flags . . . 18

1.1.4.12 Map Write Flag . . . 18

1.1.4.13 Map Write Mode . . . 19

1.1.4.14 Map Order . . . 19

1.1.4.15 Map Return Types . . . 19

1.1.4.16 Bitwise Write Flags . . . 20

1.1.4.17 Bitwise Resize Flags . . . 20

1.1.4.18 Bitwise Overflow . . . 21

1.1.4.19 HyperLogLog Write Flags . . . 21

1.1.4.20 Expression Write Flags . . . 21

1.1.4.21 Expression Read Flags . . . 22

1.1.4.22 Bin Types . . . 22

1.1.4.23 Miscellaneous . . . 23

1.1.4.24 Log Level . . . 23

1.1.4.25 Privileges . . . 24

1.1.4.26 Regex Flag Values . . . 24

1.2 Client Class — Client . . . 24
1.2.1 Client . . . 24
1.2.1.1 Connection . . . 26
1.2.1.2 Key Tuple . . . 26
1.2.1.3 Record Tuple . . . 27
1.2.2 Operations . . . 27
1.2.2.1 Record Operations . . . 27
1.2.2.2 Batch Operations . . . 34
1.2.2.3 String Operations . . . 38
1.2.2.4 Numeric Operations . . . 40
1.2.2.5 List Operations . . . 41
1.2.2.6 Map Operations . . . 41
1.2.2.7 Multi-Ops (Operate) . . . 41
1.2.2.8 Scan and Query . . . 46
1.2.2.9 User Defined Functions . . . 46
1.2.2.10 Info Operations . . . 52
1.2.2.11 Index Operations . . . 57
1.2.2.12 Admin Operations . . . 60
1.2.3 Policies . . . 64
1.2.3.1 Write Policies . . . 64
1.2.3.2 Read Policies . . . 65
1.2.3.3 Operate Policies . . . 66
1.2.3.4 Apply Policies . . . 68
1.2.3.5 Remove Policies . . . 69
1.2.3.6 Batch Policies . . . 70
1.2.3.7 Info Policies . . . 72
1.2.3.8 Admin Policies . . . 72
1.2.3.9 List Policies . . . 72
1.2.3.10 Map Policies . . . 73
1.2.3.11 Bit Policies . . . 73
1.2.3.12 HyperLogLog Policies . . . 74
1.2.4 Misc . . . 74
1.2.4.1 Role Objects . . . 74
1.2.4.2 Privilege Objects . . . 75
1.2.4.3 Unicode Handling . . . 75
1.3 Scan Class — Scan . . . 76
1.3.1 Scan . . . 76
1.3.1.1 Scan Methods . . . 76
1.3.1.2 Scan Policies . . . 82
1.3.1.3 Scan Options . . . 83
1.4 Query Class — Query . . . 84
1.4.1 Query . . . 84
1.4.1.1 Query Methods . . . 84
1.4.1.2 Query Policies . . . 93
1.4.1.3 Query Options . . . 94
1.5 aerospike.predicates — Query Predicates . . . 94
1.6 aerospike.predexp — Query Predicate Expressions . . . 100
1.7 aerospike.exception — Aerospike Exceptions . . . 111
1.7.1 In Doubt Status . . . 112
1.7.2 Exception Types . . . 112
1.7.3 Exception Hierarchy . . . 115
1.8 aerospike_helpers — Aerospike Helper Package for bin operations (list, map, bit, etc.) . . . 117
1.8.1 Subpackages . . . 117
1.8.1.1 aerospike_helpers.operations package . . . 117

1.8.1.2 aerospike_helpers.expressions package . . . 159
1.8.1.3 aerospike_helpers.cdt_ctx module . . . 223
1.9 GeoJSON Class — GeoJSON . . . 226
1.9.1 GeoJSON . . . 226
1.10 Data_Mapping — Python Data Mappings . . . 227
1.11 KeyOrderedDict Class — KeyOrderedDict . . . 228
1.11.1 KeyOrderedDict . . . 228

2 Indices and tables **231**

Python Module Index **233**

Index **235**

`aerospike` is a package which provides a Python client for Aerospike database clusters. The Python client is a CPython module, built on the Aerospike C client.

- *aerospike* - the module containing the Client, Query, and Scan Classes.
- *Scan Class — Scan* is a class built to handle scan operations of entire sets.
- *Query Class — Query* is a class built to handle queries over secondary indexes.
- *aerospike.predicates* is a submodule containing predicate helpers for use with the Query class.
- *aerospike.predexp* is a submodule containing predicate expression helpers for use with the Query class.
- *aerospike.exception* is a submodule containing the exception hierarchy for AerospikeError and its subclasses.
- *aerospike_helpers* is a helper package for bin operations (list, map, bitwise, etc.) and aerospike expressions.
- *GeoJSON Class — GeoJSON* is a class to handle GeoJSON type data.
- *Data_Mapping — Python Data Mappings* How Python types map to Aerospike Server types.

See also:

The Python Client Manual for a quick guide.

CHAPTER

ONE

CONTENT

1.1 aerospike — Aerospike Client for Python

The Aerospike client enables you to build an application in Python with an Aerospike cluster as its database. The client manages the connections to the cluster and handles the transactions performed against it.

Data Model

At the top is the `namespace`, a container that has one set of policy rules for all its data, and is similar to the *database* concept in an RDBMS, only distributed across the cluster. A namespace is subdivided into `sets`, similar to *tables*.

Pairs of key-value data called `bins` make up `records`, similar to *columns* of a *row* in a standard RDBMS. Aerospike is schema-less, meaning that you do not need to define your bins in advance.

Records are uniquely identified by their key, and record metadata is contained in an in-memory primary index.

See also:

Architecture Overview and Aerospike Data Model for more information about Aerospike.

1.1.1 Methods

`aerospike.client(`*config*`)`

Creates a new instance of the Client class. This client can *connect()* to the cluster and perform operations against it, such as *put()* and *get()* records.

This is a wrapper function which calls the constructor for the *Client* class. The client may also be constructed by calling the constructor directly.

Parameters **config** (*dict*) – the client's configuration.

aggedright

- **hosts a required list of (address, port, [tls-name]) tuples identifying a node (or multiple nodes) in the cluste**

  The client will connect to the first available node in the list, the *seed node*, and will learn about the cluster and partition map from it. If tls-name is specified, it must match the tls-name specified in the node's server configuration file and match the server's CA certificate.

  Note: TLS usage requires Aerospike Enterprise Edition

- **lua an optional dict containing the paths to two types of Lua**

  system_path

  The location of the system modules such as `aerospike.lua`

Default: `/usr/local/aerospike/lua`

user_path

The location of the user's record and stream UDFs .
Default: `./`

- **policies a `dict` of policies**

  read (`dict`)

  A dictionary containing *Read Policies*.

  write (`dict`)

  A dictionary containing *Write Policies*.

  apply (`dict`)

  A dictionary containing *Apply Policies*.

  operate (`dict`)

  A dictionary containing *Operate Policies*.

  remove (`dict`)

  A dictionary containing *Remove Policies*.

  query (`dict`)

  A dictionary containing *Query Policies*.

  scan (`dict`)

  A dictionary containing *Scan Policies*.

  batch (`dict`)

  A dictionary containing *Batch Policies*.

  total_timeout default connection timeout in milliseconds

  Deprecated: set this individually in the *Policies* dictionaries.

  auth_mode

  A value of *Auth Mode Constants* defining how the authentication mode with the server, such as *`aerospike.AUTH_INTERNAL`*.
  Default: *`aerospike.AUTH_INTERNAL`*

  login_timeout_ms (`int`)

  Representing the node login timeout in milliseconds.
  Default: `5000`.

  key default key policy

  Deprecated: set this individually in the *Policies* dictionaries.

  exists default exists policy

  Deprecated: set in the *Write Policies* dictionary

  max_retries representing the number of times to retry a tran

  Deprecated: set this individually in the *Policies* dictionaries.

  replica default replica policy

  Deprecated: set this in one or all of the other policies' *Read Policies*, *Write Policies*, *Apply Policies*, *Operate Policies*, *Remove Policies* dictionaries.

  commit_level default commit level policy

  Deprecated: set this as needed individually in the *Write Policies*, *Apply Policies*, *Operate Policies*, *Remove Policies* dictionaries.

- **shm a `dict` with optional shared-memory cluster tending para**

  Shared-memory cluster tending is on if the `dict` is provided. If multiple clients are instantiated talking to the same cluster the *shm* cluster-tending should be used.

  max_nodes (`int`)

  Maximum number of nodes allowed. Pad so new nodes can be added without configuration changes
  Default: `16`

  max_namespaces (`int`)

  Similarly pad
  Default: `8`

  takeover_threshold_sec (`int`)

Take over tending if the cluster hasn't been checked for this many seconds

Default: `30`

shm_key

Explicitly set the shm key for this client.

If **use_shared_connection** is not set, or set to `False`, the user must provide a value for this field in order for shared memory to work correctly.

If , and only if, **use_shared_connection** is set to `True`, the key will be implicitly evaluated per unique hostname, and can be inspected with *shm_key()* .

It is still possible to specify a key when using **use_shared_connection** = *True*.

default: `0xA8000000`

- **use_shared_connection** (`bool`)

  Indicating whether this instance should share its connection to the Aerospike cluster with other client instances in the same process.

  Default: `False`

- **tls** a `dict` of optional TLS configuration parameters.

  Note: TLS usage requires Aerospike Enterprise Edition

  ---

  enable (`bool`)

  Indicating whether tls should be enabled or not.

  Default: `False`

  cafile (`str`)

  Path to a trusted CA certificate file. By default TLS will use system standard trusted CA certificates

  capath (`str`)

  Path to a directory of trusted certificates. See the OpenSSL SSL_CTX_load_verify_locations manual page for more information about the format of the directory.

  protocols (`str`)

  Specifies enabled protocols. This format is the same as Apache's SSLProtocol documented at https://httpd.apache.org/docs/current/mod/mod_ssl.html#sslprotocol .

  If not specified the client will use "-all +TLSv1.2".

  cipher_suite (`str`)

  Specifies enabled cipher suites. The format is the same as OpenSSL's Cipher List Format documented at https://www.openssl.org/docs/manmaster/apps/ciphers.html .

  If not specified the OpenSSL default cipher suite described in the ciphers documentation will be used. If you are not sure what cipher suite to select this option is best left unspecified

  keyfile (`str`)

  Path to the client's key for mutual authentication. By default mutual authentication is disabled.

  keyfile_pw (`str`)

  Decryption password for the client's key for mutual authentication. By default the key is assumed not to be encrypted.

  cert_blacklist (`str`)

  Path to a certificate blacklist file. The file should contain one line for each blacklisted certificate. Each line starts with the certificate serial number expressed in hex. Each entry may optionally specify the issuer name of the certificate (serial numbers are only required to be unique per issuer). Example records: 867EC87482B2 /C=US/ST=CA/O=Acme/OU=Engineering/CN=Test

Chain CA E2D4B0E570F9EF8E885C065899886461

certfile (`str`)

Path to the client's certificate chain file for mutual authentication. By default mutual authentication is disabled.

crl_check (`bool`)

Enable CRL checking for the certificate chain leaf certificate. An error occurs if a suitable CRL cannot be found. By default CRL checking is disabled.

crl_check_all (`bool`)

Enable CRL checking for the entire certificate chain. An error occurs if a suitable CRL cannot be found. By default CRL checking is disabled.

log_session_info (`bool`)

Log session information for each connection.

for_login_only (`bool`)

Log session information for each connection. Use TLS connections only for login authentication. All other communication with the server will be done with non-TLS connections.

Default: `False` (Use TLS connections for all communication with server.)

- **send_bool_as an optional** `int` **that configures the client to write Python booleans as PY_BYTES_BLOB, inte**

One of the *Send Bool Constants* constant values.

Example: `{"send_bool_as", aerospike.aerospike.PY_BYTES}`

See `Data_Mapping` for more information.

Default: aerospike.PY_BYTES

- **serialization an optional instance-level** `tuple()` **of (serializer, deserializer).**

Takes precedence over a class serializer registered with *set_serializer()*.

- **thread_pool_size** (`int`)

Number of threads in the pool that is used in batch/scan/query commands.

Default: `16`

- **max_socket_idle** (`int`)

Maximum socket idle time in seconds. Connection pools will discard sockets that have been idle longer than the maximum. The value is limited to 24 hours (86400). It's important to set this value to a few seconds less than the server's proto-fd-idle-ms (default 60000 milliseconds, or 1 minute), so the client does not attempt to use a socket that has already been reaped by the server.

Default: `0` seconds (disabled) for non-TLS connections, 55 seconds for TLS connections

- **max_conns_per_node** (`int`)

Maximum number of pipeline connections allowed for each node

- **tend_interval** (`int`)

Polling interval in milliseconds for tending the cluster

Default: `1000`

- **compression_threshold** (`int`)

Compress data for transmission if the object size is greater than a given number of bytes

Default: `0`, meaning 'never compress'

Deprecated, set this in the 'write' policy dictionary.

- **cluster_name** (`str`)

Only server nodes matching this name will be used when determining the cluster name.

- **rack_id** (`int`)

Rack id where this client instance resides.

In order to enable this functionality, the *rack_aware* needs to be set to true, the *Read Policies* *replica* needs to be set to *POLICY_REPLICA_PREFER_RACK*.

The server rack configuration must also be configured.

Default: `0`

- **rack_aware** (`bool`)

  Track server rack data. This is useful when directing read operations to run on the same rack as the client.

  This is useful to lower cloud provider costs when nodes are distributed across different availability zones (represented as racks).

  In order to enable this functionality, the *rack_id* needs to be set to local rack, the *read policy replica* needs to be set to *POLICY_REPLICA_PREFER_RACK*. The server rack configuration must also be configured.

  Default: `False`

- **use_services_alternate** (`bool`)

  Flag to signify if "services-alternate" should be used instead of "services"

  Default: `False`

- **connect_timeout** (`int`)

  Initial host connection timeout in milliseconds. The timeout when opening a connection to the server host for the first time.

  Default: `1000`.

Returns an instance of the *aerospike.Client* class.

See also:

Shared Memory and Per-Transaction Consistency Guarantees.

```
import aerospike

# configure the client to first connect to a cluster node at 127.0.0.1
# the client will learn about the other nodes in the cluster from the
# seed node.
# in this configuration shared-memory cluster tending is turned on,
# which is appropriate for a multi-process context, such as a webserver
config = {
    'hosts':    [ ('127.0.0.1', 3000) ],
    'policies': {'read': {total_timeout': 1000}},
    'shm':      { }}
client = aerospike.client(config)
```

Changed in version 2.0.0.

```
import aerospike
import sys

# NOTE: Use of TLS Requires Aerospike Enterprise Server Version >= 3.11 and Python␣
↪Client version 2.1.0 or greater
# To view Instructions for server configuration for TLS see https://www.aerospike.
↪com/docs/guide/security/tls.html
tls_name = "some-server-tls-name"
tls_ip = "127.0.0.1"
tls_port = 4333

# If tls-name is specified, it must match the tls-name specified in the node's␣
↪server configuration file
# and match the server's CA certificate.
```

(continues on next page)

(continued from previous page)

```
tls_host_tuple = (tls_ip, tls_port, tls_name)
hosts = [tls_host_tuple]

# Example configuration which will use TLS with the specifed cafile
tls_config = {
    "cafile": "/path/to/cacert.pem",
    "enable": True
}

client = aerospike.client({
    "hosts": hosts,
    "tls": tls_config
})
try:
    client.connect()
except Exception as e:
    print(e)
    print("Failed to connect")
    sys.exit()

key = ('test', 'demo', 1)
client.put(key, {'aerospike': 'aerospike'})
print(client.get(key))
```

aerospike.**null**()

A type for distinguishing a server-side null from a Python `None`. Replaces the constant `aerospike.null`.

Returns a type representing the server-side type `as_null`.

New in version 2.0.1.

aerospike.**CDTWildcard**()

A type representing a wildcard object. This type may only be used as a comparison value in operations. It may not be stored in the database.

Returns a type representing a wildcard value.

```
import aerospike
from aerospike_helpers.operations import list_operations as list_ops

client = aerospike.client({'hosts': [('localhost', 3000)]}).connect()
key = 'test', 'demo', 1

#  get all values of the form [1, ...] from a list of lists.
#  For example if list is [[1, 2, 3], [2, 3, 4], [1, 'a']], this operation will match
#  [1, 2, 3] and [1, 'a']
operations = [list_ops.list_get_by_value('list_bin', [1, aerospike.CDTWildcard()],␣
↪aerospike.LIST_RETURN_VALUE)]
_, _, bins = client.operate(key, operations)
```

New in version 3.5.0.

Note: This requires Aerospike Server 4.3.1.3 or greater

aerospike.**CDTInfinite**()

A type representing an infinte value. This type may only be used as a comparison value in operations. It may not be stored in the database.

Returns a type representing an infinite value.

```
import aerospike
from aerospike_helpers.operations import list_operations as list_ops

client = aerospike.client({'hosts': [('localhost', 3000)]}).connect()
key = 'test', 'demo', 1

#  get all values of the form [1, ...] from a list of lists.
#  For example if list is [[1, 2, 3], [2, 3, 4], [1, 'a']], this operation will match
#  [1, 2, 3] and [1, 'a']
operations = [list_ops.list_get_by_value_range('list_bin', aerospike.LIST_RETURN_
↪VALUE, [1],  [1, aerospike.CDTInfinite()])]
_, _, bins = client.operate(key, operations)
```

New in version 3.5.0.

Note: This requires Aerospike Server 4.3.1.3 or greater

---

aerospike.**calc_digest**(*ns*, *set*, *key*) → bytearray

Calculate the digest of a particular key. See: *Key Tuple*.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **key** (str, int or bytearray) – the primary key identifier of the record within the set.

Returns a RIPEMD-160 digest of the input tuple.

Return type bytearray

```
import aerospike
import pprint

digest = aerospike.calc_digest("test", "demo", 1 )
pp.pprint(digest)
```

Serialization

Note: By default, the *aerospike.Client* maps the supported types int, str, float, bytearray, list, dict to matching aerospike server types (int, string, double, bytes, list, map). When an unsupported type is encountered, the module uses cPickle to serialize and deserialize the data, storing it into *as_bytes* of type 'Python' (AS_BYTES_PYTHON).

The functions *set_serializer()* and *set_deserializer()* allow for user-defined functions to handle serialization, instead. The serialized data is stored as 'Generic' *as_bytes* of type (AS_BYTES_BLOB). The *serialization* config param of *aerospike.client()* registers an instance-level pair of functions that handle serialization.

---

aerospike.**set_serializer**(*callback*)

Register a user-defined serializer available to all *aerospike.Client* instances.

Parameters **callback** (*callable*) – the function to invoke for serialization.

See also:

To use this function with *put()* the argument to *serializer* should be *aerospike.SERIALIZER_USER*.

```
import aerospike
import json

def my_serializer(val):
    return json.dumps(val)

aerospike.set_serializer(my_serializer)
```

New in version 1.0.39.

aerospike.**set_deserializer**(*callback*)

Register a user-defined deserializer available to all *aerospike.Client* instances. Once registered, all read methods (such as *get()*) will run bins containing 'Generic' *as_bytes* of type (AS_BYTES_BLOB) through this deserializer.

Parameters **callback** (*callable*) – the function to invoke for deserialization.

aerospike.**unset_serializers**()

Deregister the user-defined de/serializer available from *aerospike.Client* instances.

New in version 1.0.53.

Note: Serialization Examples

The following example shows the three modes of serialization - built-in, class-level user functions, instance-level user functions:

```
from __future__ import print_function
import aerospike
import marshal
import json

def go_marshal(val):
    return marshal.dumps(val)

def demarshal(val):
    return marshal.loads(val)

def jsonize(val):
    return json.dumps(val)

def dejsonize(val):
    return json.loads(val)

aerospike.set_serializer(go_marshal)
aerospike.set_deserializer(demarshal)
config = {'hosts':[('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()
```

(continues on next page)

(continued from previous page)

```
config['serialization'] = (jsonize,dejsonize)
client2 = aerospike.client(config).connect()

for i in xrange(1, 4):
    try:
        client.remove(('test', 'demo', 'foo' + i))
    except:
        pass

bin_ = {'t': (1, 2, 3)} # tuple is an unsupported type
print("Use the built-in serialization (cPickle)")
client.put(('test','demo','foo1'), bin_)
(key, meta, bins) = client.get(('test','demo','foo1'))
print(bins)

print("Use the class-level user-defined serialization (marshal)")
client.put(('test','demo','foo2'), bin_, serializer=aerospike.SERIALIZER_USER)
(key, meta, bins) = client.get(('test','demo','foo2'))
print(bins)

print("Use the instance-level user-defined serialization (json)")
client2.put(('test','demo','foo3'), bin_, serializer=aerospike.SERIALIZER_USER)
# notice that json-encoding a tuple produces a list
(key, meta, bins) = client2.get(('test','demo','foo3'))
print(bins)
client.close()
```

The expected output is:

```
Use the built-in serialization (cPickle)
{'i': 321, 't': (1, 2, 3)}
Use the class-level user-defined serialization (marshal)
{'i': 321, 't': (1, 2, 3)}
Use the instance-level user-defined serialization (json)
{'i': 321, 't': [1, 2, 3]}
```

While AQL shows the records as having the following structure:

```
aql> select i,t from test.demo where PK='foo1'
+-----+-------------------------------------------------+
| i   | t                                               |
+-----+-------------------------------------------------+
| 321 | 28 49 31 0A 49 32 0A 49 33 0A 74 70 31 0A 2E |
+-----+-------------------------------------------------+
1 row in set (0.000 secs)

aql> select i,t from test.demo where PK='foo2'
+-----+-------------------------------------------------------------------+
| i   | t                                                                 |
+-----+-------------------------------------------------------------------+
| 321 | 28 03 00 00 00 69 01 00 00 00 69 02 00 00 00 69 03 00 00 00 |
+-----+-------------------------------------------------------------------+
```

(continues on next page)

(continued from previous page)

```
1 row in set (0.000 secs)

aql> select i,t from test.demo where PK='foo3'
+-----+----------------------------+
| i   | t                          |
+-----+----------------------------+
| 321 | 5B 31 2C 20 32 2C 20 33 5D |
+-----+----------------------------+
1 row in set (0.000 secs)
```

Logging

aerospike.**set_log_handler**(*callback*)

Set a user-defined function as the log handler for all aerospike objects. The *callback* is invoked whenever a log event passing the logging level threshold is encountered.

Parameters **callback** (*callable*) – the function used as the logging handler.

Note: The callback function must have the five parameters (level, func, path, line, msg)

```
from __future__ import print_function
import aerospike

def as_logger(level, func, path, line, msg):
def as_logger(level, func, myfile, line, msg):
    print("**", myfile, line, func, ':: ', msg, "**")

aerospike.set_log_level(aerospike.LOG_LEVEL_DEBUG)
aerospike.set_log_handler(as_logger)
```

aerospike.**set_log_level**(*log_level*)

Declare the logging level threshold for the log handler.

Parameters **log_level** (*int*) – one of the *Log Level* constant values.

Geospatial

aerospike.**geodata**([*geo_data*])

Helper for creating an instance of the *GeoJSON* class. Used to wrap a geospatial object, such as a point, polygon or circle.

Parameters **geo_data** (*dict*) – a `dict` representing the geospatial data.

Returns an instance of the *aerospike.GeoJSON* class.

```
import aerospike

# Create GeoJSON point using WGS84 coordinates.
latitude = 45.920278
longitude = 63.342222
```

(continues on next page)

(continued from previous page)

```
loc = aerospike.geodata({'type': 'Point',
                         'coordinates': [longitude, latitude]})
```

New in version 1.0.54.

aerospike.**geojson**([*geojson_str*])
Helper for creating an instance of the *GeoJSON* class from a raw GeoJSON `str`.

Parameters **geojson_str** (*dict*) – a `str` of raw GeoJSON.

Returns an instance of the *aerospike.GeoJSON* class.

```
import aerospike

# Create GeoJSON point using WGS84 coordinates.
loc = aerospike.geojson('{"type": "Point", "coordinates": [-80.604333, 28.608389]}')
```

New in version 1.0.54.

1.1.2 Operators

Operators for the multi-ops method *operate()*.

Note: Starting version 3.4.0, it is highly recommended to use the *aerospike_helpers.operations package* to create the arguments for *operate()* and *operate_ordered()* Old style operatiors are deprecated. The docs for old style operators were removed in client 6.0.0.

1.1.3 Policy Options

1.1.3.1 Commit Level Policy Options

Specifies the number of replicas required to be successfully committed before returning success in a write operation to provide the desired consistency guarantee.

aerospike.**POLICY_COMMIT_LEVEL_ALL**
Return succcess only after successfully committing all replicas

aerospike.**POLICY_COMMIT_LEVEL_MASTER**
Return succcess after successfully committing the master replica

1.1.3.2 AP Read Mode Policy Options

Read policy for AP (availability) namespaces.

aerospike.**POLICY_READ_MODE_AP_ONE**
Involve single node in the read operation.

aerospike.**POLICY_READ_MODE_AP_ALL**
Involve all duplicates in the read operation.

New in version 3.7.0.

1.1.3.3 SC Read Mode Policy Options

Read policy for SC (strong consistency) namespaces.

aerospike.**POLICY_READ_MODE_SC_SESSION**
: Ensures this client will only see an increasing sequence of record versions. Server only reads from master. This is the default.

aerospike.**POLICY_READ_MODE_SC_LINEARIZE**
: Ensures ALL clients will only see an increasing sequence of record versions. Server only reads from master.

aerospike.**POLICY_READ_MODE_SC_ALLOW_REPLICA**
: Server may read from master or any full (non-migrating) replica. Increasing sequence of record versions is not guaranteed.

aerospike.**POLICY_READ_MODE_SC_ALLOW_UNAVAILABLE**
: Server may read from master or any full (non-migrating) replica or from unavailable partitions. Increasing sequence of record versions is not guaranteed.

New in version 3.7.0.

1.1.3.4 Existence Policy Options

Specifies the behavior for writing the record depending whether or not it exists.

aerospike.**POLICY_EXISTS_CREATE**
: Create a record, ONLY if it doesn't exist

aerospike.**POLICY_EXISTS_CREATE_OR_REPLACE**
: Completely replace a record if it exists, otherwise create it

aerospike.**POLICY_EXISTS_IGNORE**
: Write the record, regardless of existence. (i.e. create or update)

aerospike.**POLICY_EXISTS_REPLACE**
: Completely replace a record, ONLY if it exists

aerospike.**POLICY_EXISTS_UPDATE**
: Update a record, ONLY if it exists

1.1.3.5 Generation Policy Options

Specifies the behavior of record modifications with regard to the generation value.

aerospike.**POLICY_GEN_IGNORE**
: Write a record, regardless of generation

aerospike.**POLICY_GEN_EQ**
: Write a record, ONLY if generations are equal

aerospike.**POLICY_GEN_GT**
: Write a record, ONLY if local generation is greater-than remote generation

1.1.3.6 Key Policy Options

Specifies the behavior for whether keys or digests should be sent to the cluster.

aerospike.**POLICY_KEY_DIGEST**
: Calculate the digest on the client-side and send it to the server

aerospike.**POLICY_KEY_SEND**
: Send the key in addition to the digest. This policy causes a write operation to store the key on the server

1.1.3.7 Replica Options

Specifies which partition replica to read from.

aerospike.**POLICY_REPLICA_SEQUENCE**
: Always try node containing master partition first. If connection fails and *retry_on_timeout* is true, try node containing prole partition. Currently restricted to master and one prole.

aerospike.**POLICY_REPLICA_MASTER**
: Read from the partition master replica node

aerospike.**POLICY_REPLICA_ANY**
: Distribute reads across nodes containing key's master and replicated partition in round-robin fashion. Currently restricted to master and one prole.

aerospike.**POLICY_REPLICA_PREFER_RACK**
: Try node on the same rack as the client first. If there are no nodes on the same rack, use POLICY_REPLICA_SEQUENCE instead.

  rack_aware and **rack_id** must be set in the config argument of the client constructor in order to enable this functionality

1.1.3.8 Retry Policy Options

Specifies the behavior of failed operations.

aerospike.**POLICY_RETRY_NONE**
: Only attempt an operation once

aerospike.**POLICY_RETRY_ONCE**
: If an operation fails, attempt the operation one more time

1.1.4 Constants

1.1.4.1 TTL Constants

Specifies the TTL constants.

aerospike.**TTL_NAMESPACE_DEFAULT**
: Use the namespace default TTL.

aerospike.**TTL_NEVER_EXPIRE**
: Set TTL to never expire.

aerospike.**TTL_DONT_UPDATE**
: Do not change the current TTL of the record.

1.1.4.2 Auth Mode Constants

Specifies the type of authentication to be used when communicating with the server.

aerospike.**AUTH_INTERNAL**
: Use internal authentication only. Hashed password is stored on the server. Do not send clear password. This is the default.

aerospike.**AUTH_EXTERNAL**
: Use external authentication (like LDAP). Specific external authentication is configured on server. If TLS defined, send clear password on node login via TLS. Throw exception if TLS is not defined.

aerospike.**AUTH_EXTERNAL_INSECURE**
: Use external authentication (like LDAP). Specific external authentication is configured on server. Send clear password on node login whether or not TLS is defined. This mode should only be used for testing purposes because it is not secure authentication.

1.1.4.3 Scan Constants

aerospike.**SCAN_PRIORITY**
: Deprecated since version 3.10.0: Scan priority has been replaced by the records_per_second policy see *Scan Policies*. Scan priority will be removed in a coming release.

aerospike.**SCAN_STATUS_ABORTED**
: Deprecated since version 1.0.50: used by *scan_info()*

aerospike.**SCAN_STATUS_COMPLETED**
: Deprecated since version 1.0.50: used by *scan_info()*

aerospike.**SCAN_STATUS_INPROGRESS**
: Deprecated since version 1.0.50: used by *scan_info()*

aerospike.**SCAN_STATUS_UNDEF**
: Deprecated since version 1.0.50: used by *scan_info()*

New in version 1.0.39.

1.1.4.4 Job Constants

aerospike.**JOB_SCAN**
: Scan job type argument for the module parameter of *job_info()*

aerospike.**JOB_QUERY**
: Query job type argument for the module parameter of *job_info()*

1.1.4.5 Job Statuses

aerospike.**JOB_STATUS_UNDEF**

aerospike.**JOB_STATUS_INPROGRESS**

aerospike.**JOB_STATUS_COMPLETED**

New in version 1.0.50.

1.1.4.6 Serialization Constants

aerospike.**SERIALIZER_PYTHON**
Use the cPickle serializer to handle unsupported types (default)

aerospike.**SERIALIZER_USER**
Use a user-defined serializer to handle unsupported types. Must have been registered for the aerospike class or configured for the Client object

aerospike.**SERIALIZER_NONE**
Do not serialize bins whose data type is unsupported

New in version 1.0.47.

1.1.4.7 Send Bool Constants

Specifies how the Python client will write Python booleans.

aerospike.**PY_BYTES**
Write Python Booleans as PY_BYTES_BLOBs.

aerospike.**INTEGER**
Write Python Booleans as integers.

aerospike.**AS_BOOL**
Write Python Booleans as as_bools.

1.1.4.8 List Write Flags

Flags used by list write flag.

aerospike.**LIST_WRITE_DEFAULT**
Default. Allow duplicate values and insertions at any index.

aerospike.**LIST_WRITE_ADD_UNIQUE**
Only add unique values.

aerospike.**LIST_WRITE_INSERT_BOUNDED**
Enforce list boundaries when inserting. Do not allow values to be inserted at index outside current list boundaries.

Note: Requires server version >= 4.3.0

---

aerospike.**LIST_WRITE_NO_FAIL**
Do not raise error if a list item fails due to write flag constraints (always succeed).

Note: Requires server version >= 4.3.0

---

aerospike.**LIST_WRITE_PARTIAL**
Allow other valid list items to be committed if a list item fails due to write flag constraints.

1.1.4.9 List Return Types

Return types used by various list operations.

aerospike.**LIST_RETURN_NONE**
Do not return any value.

aerospike.**LIST_RETURN_INDEX**
Return key index order.

aerospike.**LIST_RETURN_REVERSE_INDEX**
Return reverse key order.

aerospike.**LIST_RETURN_RANK**
Return value order.

aerospike.**LIST_RETURN_REVERSE_RANK**
Return reverse value order.

aerospike.**LIST_RETURN_COUNT**
Return count of items selected.

aerospike.**LIST_RETURN_VALUE**
Return value for single key read and value list for range read.

1.1.4.10 List Order

Flags used by list order.

aerospike.**LIST_UNORDERED**
List is not ordered. This is the default.

aerospike.**LIST_ORDERED**
Ordered list.

1.1.4.11 List Sort Flags

Flags used by list sort.

aerospike.**LIST_SORT_DEFAULT**
Default. Preserve duplicates when sorting the list.

aerospike.**LIST_SORT_DROP_DUPLICATES**
Drop duplicate values when sorting the list.

1.1.4.12 Map Write Flag

Flags used by map write flag.

Note: Requires server version >= 4.3.0

---

aerospike.**MAP_WRITE_FLAGS_DEFAULT**
Default. Allow create or update.

aerospike.**MAP_WRITE_FLAGS_CREATE_ONLY**
If the key already exists, the item will be denied. If the key does not exist, a new item will be created.

aerospike.**MAP_WRITE_FLAGS_UPDATE_ONLY**
If the key already exists, the item will be overwritten. If the key does not exist, the item will be denied.

aerospike.**MAP_WRITE_FLAGS_NO_FAIL**
Do not raise error if a map item is denied due to write flag constraints (always succeed).

aerospike.**MAP_WRITE_FLAGS_PARTIAL**
Allow other valid map items to be committed if a map item is denied due to write flag constraints.

1.1.4.13 Map Write Mode

Flags used by map *write mode*.

Note: This should only be used for Server version < 4.3.0

aerospike.**MAP_UPDATE**
Default. Allow create or update.

aerospike.**MAP_CREATE_ONLY**
If the key already exists, the item will be denied. If the key does not exist, a new item will be created.

aerospike.**MAP_UPDATE_ONLY**
If the key already exists, the item will be overwritten. If the key does not exist, the item will be denied.

1.1.4.14 Map Order

Flags used by map order.

aerospike.**MAP_UNORDERED**
Map is not ordered. This is the default.

aerospike.**MAP_KEY_ORDERED**
Order map by key.

aerospike.**MAP_KEY_VALUE_ORDERED**
Order map by key, then value.

1.1.4.15 Map Return Types

Return types used by various map operations.

aerospike.**MAP_RETURN_NONE**
Do not return any value.

aerospike.**MAP_RETURN_INDEX**
Return key index order.

aerospike.**MAP_RETURN_REVERSE_INDEX**
Return reverse key order.

aerospike.**MAP_RETURN_RANK**
Return value order.

aerospike.**MAP_RETURN_REVERSE_RANK**
Return reserve value order.

aerospike.**MAP_RETURN_COUNT**
Return count of items selected.

aerospike.**MAP_RETURN_KEY**
Return key for single key read and key list for range read.

aerospike.**MAP_RETURN_VALUE**
Return value for single key read and value list for range read.

aerospike.**MAP_RETURN_KEY_VALUE**
Return key/value items. Note that key/value pairs will be returned as a list of tuples (i.e. [(key1, value1), (key2, value2)])

1.1.4.16 Bitwise Write Flags

aerospike.**BIT_WRITE_DEFAULT**
Allow create or update (default).

aerospike.**BIT_WRITE_CREATE_ONLY**
If bin already exists the operation is denied. Otherwise the bin is created.

aerospike.**BIT_WRITE_UPDATE_ONLY**
If bin does not exist the operation is denied. Otherwise the bin is updated.

aerospike.**BIT_WRITE_NO_FAIL**
Do not raise error if operation failed.

aerospike.**BIT_WRITE_PARTIAL**
Allow other valid operations to be committed if this operation is denied due to flag constraints. i.e. If the number of bytes from the offset to the end of the existing Bytes bin is less than the specified number of bytes, then only apply operations from the offset to the end.

New in version 3.9.0.

1.1.4.17 Bitwise Resize Flags

aerospike.**BIT_RESIZE_DEFAULT**
Add/remove bytes from the end (default).

aerospike.**BIT_RESIZE_FROM_FRONT**
Add/remove bytes from the front.

aerospike.**BIT_RESIZE_GROW_ONLY**
Only allow the bitmap size to increase.

aerospike.**BIT_RESIZE_SHRINK_ONLY**
Only allow the bitmap size to decrease.

New in version 3.9.0.

1.1.4.18 Bitwise Overflow

aerospike.**BIT_OVERFLOW_FAIL**
Operation will fail on overflow/underflow.

aerospike.**BIT_OVERFLOW_SATURATE**
If add or subtract ops overflow/underflow, set to max/min value. Example: MAXINT + 1 = MAXINT.

aerospike.**BIT_OVERFLOW_WRAP**
If add or subtract ops overflow/underflow, wrap the value. Example: MAXINT + 1 = MININT.

New in version 3.9.0.

1.1.4.19 HyperLogLog Write Flags

aerospike.**HLL_WRITE_DEFAULT**
Default. Allow create or update.

aerospike.**HLL_WRITE_CREATE_ONLY**
If the bin already exists, the operation will be denied. If the bin does not exist, a new bin will be created.

aerospike.**HLL_WRITE_UPDATE_ONLY**
If the bin already exists, the bin will be overwritten. If the bin does not exist, the operation will be denied.

aerospike.**HLL_WRITE_NO_FAIL**
Do not raise error if operation is denied.

aerospike.**HLL_WRITE_ALLOW_FOLD**
Allow the resulting set to be the minimum of provided index bits. For intersect_counts and similarity, allow the usage of less precise HLL algorithms when minhash bits of all participating sets do not match.

New in version 3.11.0.

1.1.4.20 Expression Write Flags

Flags used by expression_write.

aerospike.**EXP_WRITE_DEFAULT**
Default. Allow create or update.

aerospike.**EXP_WRITE_CREATE_ONLY**
If bin does not exist, a new bin will be created. If bin exists, the operation will be denied. If bin exists, fail with BinExistsError when EXP_WRITE_POLICY_NO_FAIL is not set.

aerospike.**EXP_WRITE_UPDATE_ONLY**
If bin exists, the bin will be overwritten. If bin does not exist, the operation will be denied. If bin does not exist, fail with BinNotFound when EXP_WRITE_POLICY_NO_FAIL is not set.

aerospike.**EXP_WRITE_ALLOW_DELETE**
If expression results in nil value, then delete the bin. Otherwise, return OpNotApplicable when EXP_WRITE_POLICY_NO_FAIL is not set.

aerospike.**EXP_WRITE_POLICY_NO_FAIL**
Do not raise error if operation is denied.

aerospike.**EXP_WRITE_EVAL_NO_FAIL**
Ignore failures caused by the expression resolving to unknown or a non-bin type.

1.1.4.21 Expression Read Flags

Flags used by expression_read

aerospike.**EXP_READ_DEFAULT**
Default.

aerospike.**EXP_READ_EVAL_NO_FAIL**
Ignore failures caused by the expression resolving to unknown or a non-bin type.

1.1.4.22 Bin Types

aerospike.**AS_BYTES_UNDEF**
(int): 0

aerospike.**AS_BYTES_INTEGER**
(int): 1

aerospike.**AS_BYTES_DOUBLE**
(int): 2

aerospike.**AS_BYTES_STRING**
(int): 3

aerospike.**AS_BYTES_BLOB**
(int): 4

aerospike.**AS_BYTES_JAVA**
(int): 7

aerospike.**AS_BYTES_CSHARP**
(int): 8

aerospike.**AS_BYTES_PYTHON**
(int): 9

aerospike.**AS_BYTES_RUBY**
(int): 10

aerospike.**AS_BYTES_PHP**
(int): 11

aerospike.**AS_BYTES_ERLANG**
(int): 12

aerospike.**AS_BYTES_HLL**
(int): 18

aerospike.**AS_BYTES_MAP**
(int): 19

aerospike.**AS_BYTES_LIST**
(int): 20

aerospike.**AS_BYTES_GEOJSON**
(int): 23

aerospike.**AS_BYTES_TYPE_MAX**
(int): 24

1.1.4.23 Miscellaneous

aerospike.**__version__**
A `str` containing the module's version.

New in version 1.0.54.

aerospike.**null**
A value for distinguishing a server-side null from a Python `None`.

Deprecated since version 2.0.1: use the function *aerospike.null()* instead.

aerospike.**UDF_TYPE_LUA**
UDF type is LUA (which is the only UDF type).

aerospike.**INDEX_STRING**
An index whose values are of the aerospike string data type.

aerospike.**INDEX_NUMERIC**
An index whose values are of the aerospike integer data type.

aerospike.**INDEX_GEO2DSPHERE**
An index whose values are of the aerospike GetJSON data type.

See also:

Data Types.

aerospike.**INDEX_TYPE_LIST**
Index a bin whose contents is an aerospike list.

aerospike.**INDEX_TYPE_MAPKEYS**
Index the keys of a bin whose contents is an aerospike map.

aerospike.**INDEX_TYPE_MAPVALUES**
Index the values of a bin whose contents is an aerospike map.

1.1.4.24 Log Level

aerospike.**LOG_LEVEL_TRACE**

aerospike.**LOG_LEVEL_DEBUG**

aerospike.**LOG_LEVEL_INFO**

aerospike.**LOG_LEVEL_WARN**

aerospike.**LOG_LEVEL_ERROR**

aerospike.**LOG_LEVEL_OFF**

1.1.4.25 Privileges

Permission codes define the type of permission granted for a user's role.

aerospike.**PRIV_READ**
The user is granted read access.

aerospike.**PRIV_WRITE**
The user is granted write access.

aerospike.**PRIV_READ_WRITE**
The user is granted read and write access.

aerospike.**PRIV_READ_WRITE_UDF**
The user is granted read and write access, and the ability to invoke UDFs.

aerospike.**PRIV_SYS_ADMIN**
The user is granted the ability to perform system administration operations. Global scope only.

aerospike.**PRIV_USER_ADMIN**
The user is granted the ability to perform user administration operations. Global scope only.

aerospike.**PRIV_DATA_ADMIN**
User can perform systems administration functions on a database that do not involve user administration. Examples include setting dynamic server configuration. Global scope only.

1.1.4.26 Regex Flag Values

Flags used for the *predexp.string_regex* function.

aerospike.**REGEX_NONE**
Use default behavior.

aerospike.**REGEX_ICASE**
Do not differentiate case.

aerospike.**REGEX_EXTENDED**
Use POSIX Extended Regular Expression syntax when interpreting regex.

aerospike.**REGEX_NOSUB**
Do not report position of matches.

aerospike.**REGEX_NEWLINE**
Match-any-character operators don't match a newline.

1.2 Client Class — `Client`

1.2.1 `Client`

The client connects through a seed node (the address of a single node) to an Aerospike database cluster. From the seed node, the client learns of the other nodes and establishes connections to them. It also gets the partition map of the cluster, which is how it knows where every record actually lives.

The client handles the connections, including re-establishing them ahead of executing an operation. It keeps track of changes to the cluster through a cluster-tending thread.

See also:

Client Architecture and Data Distribution.

Example:

```
    from __future__ import print_function
    # import the module
    import aerospike
    from aerospike import exception as ex
    import sys

    # Configure the client
    config = {
        'hosts': [ ('127.0.0.1', 3000) ]
    }

    # Optionally set policies for various method types
    write_policies = {'total_timeout': 2000, 'max_retries': 0}
    read_policies = {'total_timeout': 1500, 'max_retries': 1}
    policies = {'write': write_policies, 'read': read_policies}
    config['policies'] = policies

    # Create a client and connect it to the cluster
    try:
        client = aerospike.client(config).connect()
    except ex.ClientError as e:
        print("Error: {0} [{1}]".format(e.msg, e.code))
        sys.exit(1)

    # Records are addressable via a tuple of (namespace, set, primary key)
    key = ('test', 'demo', 'foo')

    try:
        # Write a record
        client.put(key, {
            'name': 'John Doe',
            'age': 32
        })
    except ex.RecordError as e:
        print("Error: {0} [{1}]".format(e.msg, e.code))

    # Read a record
    (key, meta, record) = client.get(key)

    # Close the connection to the Aerospike cluster
    client.close()


.. index::
    single: Connection

.. _aerospike_connection_operations:
```

1.2.1.1 Connection

class aerospike.**Client**

connect([*username*, *password*])

Connect to the cluster. The optional *username* and *password* only apply when connecting to the Enterprise Edition of Aerospike.

Parameters

- **username** (*str*) – a defined user with roles in the cluster. See *admin_create_user()*.
- **password** (*str*) – the password will be hashed by the client using bcrypt.

Raises *ClientError*, for example when a connection cannot be established to a seed node (any single node in the cluster from which the client learns of the other nodes).

Note: Python client 5.0.0 and up will fail to connect to Aerospike server 4.8.x or older. If you see the error "-10, 'Failed to connect'", please make sure you are using server 4.9 or later.

See also:

Security features article.

is_connected()

Tests the connections between the client and the nodes of the cluster. If the result is `False`, the client will require another call to *connect()*.

Return type `bool`

Changed in version 2.0.0.

close()

Close all connections to the cluster. It is recommended to explicitly call this method when the program is done communicating with the cluster.

1.2.1.2 Key Tuple

key

The key tuple, which is sent and returned by various operations, has the structure

```
(namespace, set, primary key[, the record's RIPEMD-160 digest])
```

aggedright

- *namespace* the `str` name of the namespace, which must be preconfigured on the cluster.
- *set* the `str` name of the set. Will be created automatically if it does not exist.
- *primary key* the value by which the client-side application identifies the record, which can be of type `str`, `int` or `bytearray`.
- *digest* the first three parts of the tuple get hashed through RIPEMD-160, and the digest used by the clients and cluster nodes to locate the record. A key tuple is also valid if it has the digest part filled and the primary key part set to `None`.

```
>>> client = aerospike.client(config).connect()
>>> client.put(('test','demo','oof'), {'id':0, 'a':1})
>>> (key, meta, bins) = client.get(('test','demo','oof'))
>>> key
('test', 'demo', None, bytearray(b'\ti\xcb\xb9\xb6V#V\xecI#\xealu\x05\x00H\x98\
↪xe4='))
>>> (key2, meta2, bins2) = client.get(key)
>>> bins2
{'a': 1, 'id': 0}
>>> client.close()
```

See also:

Data Model: Keys and Digests.

1.2.1.3 Record Tuple

record

The record tuple `(key, meta, bins)` which is returned by various read operations.

aggedright

- *key* the *Key Tuple*.
- *meta* a `dict` containing `{'gen' : genration value, 'ttl': ttl value}`.
- *bins* a `dict` containing bin-name/bin-value pairs.

See also:

Data Model: Record.

1.2.2 Operations

1.2.2.1 Record Operations

class aerospike.**Client**

put(*key*, *bins*[, *meta*[, *policy*[, *serializer*]]])

Write a record with a given *key* to the cluster.

Parameters

- **key** (*tuple*) – a *Key Tuple* tuple associated with the record.
- **bins** (*dict*) – a `dict` of bin-name / bin-value pairs.
- **meta** (*dict*) – optional record metadata to be set, with field `'ttl'` set to `int` number of seconds or one of the *TTL Constants*, and `'gen'` set to `int` generation number to compare.
- **policy** (*dict*) – optional *Write Policies*.
- **serializer** – optionally override the serialization mode of the client with one of the *Serialization Constants*. To use a class-level user-defined serialization function registered with *aerospike.set_serializer()* use *aerospike.SERIALIZER_USER*.

Raises a subclass of *AerospikeError*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex

config = {
    'hosts': [ ('127.0.0.1', 3000) ],
    'total_timeout': 1500
}
client = aerospike.client(config).connect()
try:
    key = ('test', 'demo', 1)
    bins = {
        'l': [ "qwertyuiop", 1, bytearray("asd;as[d'as;d", "utf-8") ],
        'm': { "key": "asd';q;'1';" },
        'i': 1234,
        'f': 3.14159265359,
        's': '!@#@#$QSDAsd;as'
    }
    client.put(key, bins,
               policy={'key': aerospike.POLICY_KEY_SEND},
               meta={'ttl':180})
    # adding a bin
    client.put(key, {'smiley': u"\ud83d\ude04"})
    # removing a bin
    client.put(key, {'i': aerospike.null()})
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Note: Version >= 3.10.0 Supports predicate expressions for record operations see *predexp*. Requires server version >= 4.7.0.

```
from __future__ import print_function
import aerospike
from aerospike import predexp
from aerospike import exception as ex
import sys

config = {"hosts": [("127.0.0.1", 3000)]}
client = aerospike.client(config).connect()

try:
    keys = [("test", "demo", 1), ("test", "demo", 2), ("test", "demo", 3)]
    records = [{"number": 1}, {"number": 2}, {"number": 3}]
    for i in range(3):
        client.put(keys[i], records[i])

    preds = [  # check that the record has a value < 2 bin 'name'
        predexp.integer_bin("number"),
        predexp.integer_value(2),
```

(continues on next page)

(continued from previous page)

```
        predexp.integer_less(),
    ]
    records = []

    for i in range(3):
        try:
            records.append(client.get(keys[i], policy={"predexp": preds}))
        except ex.FilteredOut as e:
            print("Error: {0} [{1}]".format(e.msg, e.code))

    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
# the get only returns records that match the preds
# otherwise, an error is returned
# EXPECTED OUTPUT:
# Error: 127.0.0.1:3000 AEROSPIKE_FILTERED_OUT [27]
# Error: 127.0.0.1:3000 AEROSPIKE_FILTERED_OUT [27]
# [(('test', 'demo', 1, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\
↪x9a\x1e\xac\xc4F\xc8')), {'gen': 8, 'ttl': 2592000}, {'charges': [10, 20, 14],
↪'name': 'John', 'number': 1})]
```

Note: Using Generation Policy

The generation policy allows a record to be written only when the generation is a specific value. In the following example, we only want to write the record if no change has occurred since *exists()* was called.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [ ('127.0.0.1',3000)]}
client = aerospike.client(config).connect()

try:
    (key, meta) = client.exists(('test','test','key1'))
    print(meta)
    print('============')
    client.put(('test','test','key1'), {'id':1,'a':2},
        meta={'gen': 33},
        policy={'gen':aerospike.POLICY_GEN_EQ})
    print('Record written.')
except ex.RecordGenerationError:
    print("put() failed due to generation policy mismatch")
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
client.close()
```

exists(*key[, policy]) -> (key, meta*)

Check if a record with a given *key* exists in the cluster and return the record as a `tuple()` consisting of *key* and *meta*. If the record does not exist the *meta* data will be `None`.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **policy** (*dict*) – optional *Read Policies*.

Return type `tuple()` (key, meta)

Raises a subclass of *AerospikeError*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    # assuming a record with such a key exists in the cluster
    key = ('test', 'demo', 1)
    (key, meta) = client.exists(key)
    print(key)
    print('---------------------------')
    print(meta)
except ex.RecordNotFound:
    print("Record not found:", key)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Changed in version 2.0.3.

get(*key[, policy]) -> (key, meta, bins*)

Read a record with a given *key*, and return the record as a `tuple()` consisting of *key*, *meta* and *bins*.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **policy** (*dict*) – optional *Read Policies*.

Returns a *Record Tuple*. See *Unicode Handling*.

Raises *RecordNotFound*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

```

(continues on next page)

(continued from previous page)

```
config = {'hosts': [('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

try:
    # assuming a record with such a key exists in the cluster
    key = ('test', 'demo', 1)
    (key, meta, bins) = client.get(key)
    print(key)
    print('---------------------------')
    print(meta)
    print('---------------------------')
    print(bins)
except ex.RecordNotFound:
    print("Record not found:", key)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Warning: The client has been changed to raise a *RecordNotFound* exception when *get()* does not find the record. Code that used to check for `meta != None` should be modified.

Changed in version 2.0.0.

select(*key, bins[, policy]) -> (key, meta, bins*)

Read a record with a given *key*, and return the record as a `tuple()` consisting of *key*, *meta* and *bins*, with the specified bins projected. Prior to Aerospike server 3.6.0, if a selected bin does not exist its value will be `None`. Starting with 3.6.0, if a bin does not exist it will not be present in the returned *Record Tuple*.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **bins** (*list*) – a list of bin names to select from the record.
- **policy** (*dict*) – optional *Read Policies*.

Returns a *Record Tuple*. See *Unicode Handling*.

Raises *RecordNotFound*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    # assuming a record with such a key exists in the cluster
    key = ('test', 'demo', 1)
    (key, meta, bins) = client.select(key, ['name'])
```

(continues on next page)

(continued from previous page)

```
    print("name: ", bins.get('name'))
except ex.RecordNotFound:
    print("Record not found:", key)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Warning: The client has been changed to raise a *RecordNotFound* exception when *select()* does not find the record. Code that used to check for `meta != None` should be modified.

Changed in version 2.0.0.

touch(*key*[, *val=0*[, *meta*[, *policy*]]])

Touch the given record, setting its time-to-live and incrementing its generation.

Parameters

- **key** (*tuple*) – a *Key Tuple* tuple associated with the record.
- **val** (*int*) – the optional ttl in seconds, with `0` resolving to the default value in the server config.
- **meta** (*dict*) – optional record metadata to be set.
- **policy** (*dict*) – optional *Operate Policies*.

Raises a subclass of *AerospikeError*.

See also:

Record TTL and Evictions and FAQ.

```
import aerospike

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

key = ('test', 'demo', 1)
client.touch(key, 120, policy={'total_timeout': 100})
client.close()
```

remove(*key[meta, policy]*)

Remove a record matching the *key* from the cluster.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **meta** (*dict*) – Optional dictonary allowing a user to specify the expected generation of the record.
- **policy** (*dict*) – optional *Remove Policies*. May be passed as a keyword argument.

Raises a subclass of *AerospikeError*.

```
import aerospike

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

key = ('test', 'demo', 1)
client.remove(key, meta={'gen': 5}, policy={'gen': aerospike.POLICY_GEN_EQ})
client.close()
```

get_key_digest(*ns*, *set*, *key*) → bytearray

Calculate the digest of a particular key. See: *Key Tuple*.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **key** (str or int) – the primary key identifier of the record within the set.

Returns a RIPEMD-160 digest of the input tuple.

Return type bytearray

```
import aerospike
import pprint

pp = pprint.PrettyPrinter(indent=2)
config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

digest = client.get_key_digest("test", "demo", 1 )
pp.pprint(digest)
key = ('test', 'demo', None, digest)
(key, meta, bins) = client.get(key)
pp.pprint(bins)
client.close()
```

Deprecated since version 2.0.1: use the function *aerospike.calc_digest()* instead.

Removing a Bin

remove_bin(*key*, *list*[, *meta*[, *policy*]])

Remove a list of bins from a record with a given *key*. Equivalent to setting those bins to *aerospike.null()* with a *put()*.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **list** (*list*) – the bins names to be removed from the record.
- **meta** (*dict*) – optional record metadata to be set, with field 'ttl' set to int number of seconds or one of the *TTL Constants*, and 'gen' set to int generation number to compare.
- **policy** (*dict*) – optional *Write Policies*.

Raises a subclass of *AerospikeError*.

```
import aerospike

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

key = ('test', 'demo', 1)
meta = { 'ttl': 3600 }
client.remove_bin(key, ['name', 'age'], meta, {'retry': aerospike.POLICY_RETRY_
↪ONCE})
client.close()
```

1.2.2.2 Batch Operations

class aerospike.**Client**

get_many(*keys*[, *policy*]) → [key, meta, bins]

Batch-read multiple records, and return them as a `list`. Any record that does not exist will have a `None` value for metadata and bins in the record tuple.

Parameters

- **keys** (*list*) – a list of *Key Tuple*.
- **policy** (*dict*) – optional *Batch Policies*.

Returns a `list` of *Record Tuple*.

Raises a *ClientError* if the batch is too big.

See also:

More information about the Batch Index interface new to Aerospike server >= 3.6.0.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    # assume the fourth key has no matching record
    keys = [
      ('test', 'demo', '1'),
      ('test', 'demo', '2'),
      ('test', 'demo', '3'),
      ('test', 'demo', '4')
    ]
    records = client.get_many(keys)
    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
```

(continues on next page)

(continued from previous page)

```
finally:
    client.close()
```

Note: We expect to see something like:

```
[
  (('test', 'demo', '1', bytearray(b'ev\xb4\x88\x8c\xcf\x92\x9c \x0bo\xbd\x90\
↪xd0\x9d\xf3\xf6\xd1\x0c\xf3')), {'gen': 1, 'ttl': 2592000}, {'age': 1, 'name
↪': u'Name1'}),
  (('test', 'demo', '2', bytearray(b'n\xcd7p\x88\xdcF\xe1\xd6\x0e\x05\xfb\xcbs\
↪xa68I\xf0T\xfd')), {'gen': 1, 'ttl': 2592000}, {'age': 2, 'name': u'Name2'}),
  (('test', 'demo', '3', bytearray(b'\x9f\xf2\xe3\xf3\xc0\xc1\xc3q\xb5$n\xf8\
↪xccV\xa9\xed\xd91a\x86')), {'gen': 1, 'ttl': 2592000}, {'age': 3, 'name': u
↪'Name3'}),
  (('test', 'demo', '4', bytearray(b'\x8eu\x19\xbe\xe0(\xda ^\xfa\x8ca\x93s\
↪xe8\xb3%\xa8]\x8b')), None, None)
]
```

Note: Version >= 3.10.0 Supports predicate expressions for batch operations see *predexp*. Requires server version >= 4.7.0

```
from __future__ import print_function
import aerospike
from aerospike import predexp
from aerospike import exception as ex
import sys

config = {"hosts": [("127.0.0.1", 3000)]}
client = aerospike.client(config).connect()

try:
    keys = [("test", "demo", 1), ("test", "demo", 2), ("test", "demo", 3)]
    records = [{"number": 1}, {"number": 2}, {"number": 3}]
    for i in range(3):
        client.put(keys[i], records[i])

    preds = [  # check that the record has a value less than 2 in bin 'name'
        predexp.integer_bin("number"),
        predexp.integer_value(2),
        predexp.integer_less(),
    ]
    records = client.get_many(keys, policy={"predexp": preds})
    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
# the get_many only returns the records that matched the preds
```

(continues on next page)

(continued from previous page)

```
# EXPECTED OUTPUT:
# [
#   (('test', 'demo', 1, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\
↪x9a\x1e\xac\xc4F\xc8')), {'gen': 8, 'ttl': 2592000}, {'charges': [10, 20, 14],
↪'name': 'John', 'number': 1}),
#   (('test', 'demo', 2, bytearray(b'\xaejQ_7\xdeJ\xda\xccD\x96\xe2\xda\x1f\xea\
↪x84\x8c:\x92p')), None, None),
#   ('test', 'demo', 3, bytearray(b'\xb1\xa5`g\xf6\xd4\xa8\xa4D9\xd3\xafb\xbf\
↪xf8ha\x01\x94\xcd')), None, None)
# ]
```

Warning: The return type changed to `list` starting with version 1.0.50.

exists_many(*keys*[, *policy*]) → [key, meta]

Batch-read metadata for multiple keys, and return it as a `list`. Any record that does not exist will have a `None` value for metadata in the result tuple.

Parameters

- **keys** (*list*) – a list of *Key Tuple*.
- **policy** (*dict*) – optional *Batch Policies*.

Returns a `list` of (key, metadata) `tuple()`.

See also:

More information about the Batch Index interface new to Aerospike server >= 3.6.0.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    # assume the fourth key has no matching record
    keys = [
      ('test', 'demo', '1'),
      ('test', 'demo', '2'),
      ('test', 'demo', '3'),
      ('test', 'demo', '4')
    ]
    records = client.exists_many(keys)
    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Note: We expect to see something like:

```
[
   (('test', 'demo', '1', bytearray(b'ev\xb4\x88\x8c\xcf\x92\x9c \x0bo\xbd\x90\
↪xd0\x9d\xf3\xf6\xd1\x0c\xf3')), {'gen': 2, 'ttl': 2592000}),
   (('test', 'demo', '2', bytearray(b'n\xcd7p\x88\xdcF\xe1\xd6\x0e\x05\xfb\
↪xcbs\xa68I\xf0T\xfd')), {'gen': 7, 'ttl': 1337}),
   (('test', 'demo', '3', bytearray(b'\x9f\xf2\xe3\xf3\xc0\xc1\xc3q\xb5$n\xf8\
↪xccV\xa9\xed\xd91a\x86')), {'gen': 9, 'ttl': 543}),
   (('test', 'demo', '4', bytearray(b'\x8eu\x19\xbe\xe0(\xda ^\xfa\x8ca\x93s\
↪xe8\xb3%\xa8]\x8b')), None)
]
```

Warning: The return type changed to `list` starting with version 1.0.50.

select_many(*keys*, *bins*[, *policy*]) → [(key, meta, bins), ...]}

Batch-read multiple records, and return them as a `list`. Any record that does not exist will have a `None` value for metadata and bins in the record tuple. The *bins* will be filtered as specified.

Parameters

- **keys** (*list*) – a list of *Key Tuple*.
- **bins** (*list*) – the bin names to select from the matching records.
- **policy** (*dict*) – optional *Batch Policies*.

Returns a `list` of *Record Tuple*.

See also:

More information about the Batch Index interface new to Aerospike server >= 3.6.0.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    # assume the fourth key has no matching record
    keys = [
      ('test', 'demo', None, bytearray(b'ev\xb4\x88\x8c\xcf\x92\x9c \x0bo\xbd\
↪x90\xd0\x9d\xf3\xf6\xd1\x0c\xf3'),
      ('test', 'demo', None, bytearray(b'n\xcd7p\x88\xdcF\xe1\xd6\x0e\x05\xfb\
↪xcbs\xa68I\xf0T\xfd'),
      ('test', 'demo', None, bytearray(b'\x9f\xf2\xe3\xf3\xc0\xc1\xc3q\xb5$n\
↪xf8\xccV\xa9\xed\xd91a\x86'),
      ('test', 'demo', None, bytearray(b'\x8eu\x19\xbe\xe0(\xda ^\xfa\x8ca\
↪x93s\xe8\xb3%\xa8]\x8b')
    ]
```

(continues on next page)

(continued from previous page)

```
    records = client.select_many(keys, [u'name'])
    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Note: We expect to see something like:

```
[
  (('test', 'demo', None, bytearray(b'ev\xb4\x88\x8c\xcf\x92\x9c \x0bo\xbd\x90\
↪xd0\x9d\xf3\xf6\xd1\x0c\xf3'), {'gen': 1, 'ttl': 2592000}, {'name': u'Name1'}
↪),
  (('test', 'demo', None, bytearray(b'n\xcd7p\x88\xdcF\xe1\xd6\x0e\x05\xfb\
↪xcbs\xa68I\xf0T\xfd'), {'gen': 1, 'ttl': 2592000}, {'name': u'Name2'}),
  (('test', 'demo', None, bytearray(b'\x9f\xf2\xe3\xf3\xc0\xc1\xc3q\xb5$n\xf8\
↪xccV\xa9\xed\xd91a\x86'), {'gen': 1, 'ttl': 2592000}, {'name': u'Name3'}),
  (('test', 'demo', None, bytearray(b'\x8eu\x19\xbe\xe0(\xda ^\xfa\x8ca\x93s\
↪xe8\xb3%\xa8]\x8b'), None, None)
]
```

Warning: The return type changed to `list` starting with version 1.0.50.

1.2.2.3 String Operations

class aerospike.**Client**

Note: Please see *aerospike_helpers.operations.operations* for the new way to use string operations.

append(*key*, *bin*, *val*[, *meta*[, *policy*]])

Append the string *val* to the string value in *bin*.

Parameters

- **key** (*tuple*) – a *Key Tuple* tuple associated with the record.
- **bin** (*str*) – the name of the bin.
- **val** (*str*) – the string to append to the value of *bin*.
- **meta** (*dict*) – optional record metadata to be set, with field `'ttl'` set to `int` number of seconds or one of the *TTL Constants*, and `'gen'` set to `int` generation number to compare.
- **policy** (*dict*) – optional *Operate Policies*.

Raises a subclass of *AerospikeError*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    key = ('test', 'demo', 1)
    client.append(key, 'name', ' jr.', policy={'total_timeout': 1200})
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

prepend(*key*, *bin*, *val*[, *meta*[, *policy*]])

Prepend the string value in *bin* with the string *val*.

Parameters

- **key** (*tuple*) – a *Key Tuple* tuple associated with the record.
- **bin** (*str*) – the name of the bin.
- **val** (*str*) – the string to prepend to the value of *bin*.
- **meta** (*dict*) – optional record metadata to be set, with field `'ttl'` set to `int` number of seconds or one of the *TTL Constants*, and `'gen'` set to `int` generation number to compare.
- **policy** (*dict*) – optional *Operate Policies*.

Raises a subclass of *AerospikeError*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    key = ('test', 'demo', 1)
    client.prepend(key, 'name', 'Dr. ', policy={'total_timeout': 1200})
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

1.2.2.4 Numeric Operations

class aerospike.**Client**

Note: Please see *aerospike_helpers.operations.operations* for the new way to use numeric operations using the operate command.

---

increment(*key*, *bin*, *offset*[, *meta*[, *policy*]])

Increment the integer value in *bin* by the integer *val*.

Parameters

- **key** (*tuple*) – a *Key Tuple* tuple associated with the record.
- **bin** (*str*) – the name of the bin.
- **offset** (int or float) – the value by which to increment the value in *bin*.
- **meta** (*dict*) – optional record metadata to be set, with field 'ttl' set to int number of seconds or one of the *TTL Constants*, and 'gen' set to int generation number to compare.
- **policy** (*dict*) – optional *Operate Policies*. Note: the exists policy option may not be: aerospike.POLICY_EXISTS_CREATE_OR_REPLACE nor aerospike.POLICY_EXISTS_REPLACE

Raises a subclass of *AerospikeError*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    client.put(('test', 'cats', 'mr. peppy'), {'breed':'persian'}, policy={
↪'exists': aerospike.POLICY_EXISTS_CREATE_OR_REPLACE})
    (key, meta, bins) = client.get(('test', 'cats', 'mr. peppy'))
    print("Before:", bins, "\n")
    client.increment(key, 'lives', -1)
    (key, meta, bins) = client.get(key)
    print("After:", bins, "\n")
    client.increment(key, 'lives', -1)
    (key, meta, bins) = client.get(key)
    print("Poor Kitty:", bins, "\n")
    print(bins)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

1.2.2.5 List Operations

class aerospike.**Client**

Note: Please see *aerospike_helpers.operations.list_operations* for the new way to use list operations. Old style list operations are deprecated. The docs for old style list operations were removed in client 6.0.0. The code supporting these methods will be removed in a coming release.

1.2.2.6 Map Operations

class aerospike.**Client**

Note: Please see *aerospike_helpers.operations.map_operations* for the new way to use map operations. Old style map operations are deprecated. The docs for old style map operations were removed in client 6.0.0. The code supporting these methods will be removed in a coming release.

1.2.2.7 Multi-Ops (Operate)

class aerospike.**Client**

operate(*key, list[, meta[, policy]]*) -> *(key, meta, bins)*

Perform multiple bin operations on a record with a given *key*, In Aerospike server versions prior to 3.6.0, non-existent bins being read will have a `None` value. Starting with 3.6.0 non-existent bins will not be present in the returned *Record Tuple*. The returned record tuple will only contain one element per bin, even if multiple operations were performed on the bin.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **list** (*list*) – a `list` of one or more bin operations, each structured as the `dict` `{'bin': bin name, 'op': aerospike.OPERATOR_* [, 'val': value]}`. See *aerospike_helpers.operations package*.
- **meta** (*dict*) – optional record metadata to be set, with field `'ttl'` set to `int` number of seconds or one of the *TTL Constants*, and `'gen'` set to `int` generation number to compare.
- **policy** (*dict*) – optional *Operate Policies*.

Returns a *Record Tuple*. See *Unicode Handling*.

Raises a subclass of *AerospikeError*.

Note: Version >= 3.10.0 Supports predicate expressions for Multi-Ops see *predexp*. Requires server version >= 4.7.0.

```
from __future__ import print_function
import aerospike
```

(continues on next page)

(continued from previous page)

```
from aerospike import predexp
from aerospike_helpers.operations import list_operations, operations
from aerospike import exception as ex
import sys

config = {"hosts": [("127.0.0.1", 3000)]}
client = aerospike.client(config).connect()

try:
    unique_id = 1
    key = ("test", "demo", unique_id)
    client.put(key, {"name": "John", "charges": [10, 20, 14]})

    ops = [list_operations.list_append("charges", 25)]

    preds = [  # check that the record has value 'Kim' in bin 'name'
        predexp.string_bin("name"),
        predexp.string_value("Kim"),
        predexp.string_equal(),
    ]

    # Because the record's name bin is 'John' and not 'Kim',
    # client.operate() will fail with AEROSPIKE_FILTERED_OUT and the
    # operations will not be applied.
    try:
        client.operate(key, ops, policy={"predexp": preds})
    except ex.FilteredOut as e:
        print("Error: {0} [{1}]".format(e.msg, e.code))

    record = client.get(key)
    print(record)

    # This client.operate() will succeed because the name bin is 'John'.
    preds = [  # check that the record has value 'John' in bin 'name'
        predexp.string_bin("name"),
        predexp.string_value("John"),
        predexp.string_equal(),
    ]

    client.operate(key, ops, policy={"predexp": preds})

    record = client.get(key)
    print(record)

except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
# Error: 127.0.0.1:3000 AEROSPIKE_FILTERED_OUT [27]
# (('test', 'demo', None, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\
↪x9a\x1e\xac\xc4F\xc8')), {'ttl': 2592000, 'gen': 23}, {'number': 1, 'name': 'John',
↪'charges': [10, 20, 14]})
```

(continues on next page)

(continued from previous page)

```
# (('test', 'demo', None, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\
↪x9a\x1e\xac\xc4F\xc8')), {'ttl': 2592000, 'gen': 24}, {'number': 1, 'name': 'John',
↪'charges': [10, 20, 14, 25]})
```

Note: In version *2.1.3* the return format of certain bin entries for this method, **only in cases when a map operation specifying a `return_type` is used**, has changed. Bin entries for map operations using "return_type" of aerospike.MAP_RETURN_KEY_VALUE will now return a bin value of a list of keys and corresponding values, rather than a list of key/value tuples. See the following code block for details.

```
# pre 2.1.3 formatting of key/value bin value
[('key1', 'val1'), ('key2', 'val2'), ('key3', 'val3')]

# >= 2.1.3 formatting
['key1', 'val1', 'key2', 'val2', 'key3', 'val3']
```

Note: *operate()* can now have multiple write operations on a single bin.

```
from __future__ import print_function
import aerospike
from aerospike_helpers.operations import operations as op_helpers
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    key = ('test', 'demo', 1)
    client.put(key, {'age': 25, 'career': 'delivery boy'})
    ops = [
    op_helpers.increment("age", 1000),
    op_helpers.write("name", "J."),
    op_helpers.prepend("name", "Phillip "),
    op_helpers.append("name", " Fry"),
    op_helpers.read("name"),
    op_helpers.read("career"),
    op_helpers.read("age")
    ]
    (key, meta, bins) = client.operate(key, ops, {'ttl':360}, {'total_timeout
↪':500})

    print(key)
    print('---------------------------')
    print(meta)
    print('---------------------------')
    print(bins) # will display all bins selected by OPERATOR_READ operations
except ex.AerospikeError as e:
```

(continues on next page)

(continued from previous page)

```
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Note: OPERATOR_TOUCH should only ever combine with OPERATOR_READ, for example to implement LRU expiry on the records of a set.

Warning: Having *val* associated with OPERATOR_TOUCH is deprecated. Use the meta *ttl* field instead.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    key = ('test', 'demo', 1)
    ops = [
        {
          "op" : aerospike.OPERATOR_TOUCH,
        },
        {
          "op" : aerospike.OPERATOR_READ,
          "bin": "name"
        }
    ]
    (key, meta, bins) = client.operate(key, ops, {'ttl':1800})
    print("Touched the record for {0}, extending its ttl by 30m".format(bins))
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Changed in version 2.1.3.

operate_ordered(*key, list[, meta[, policy]]*) -> *(key, meta, bins)*

Perform multiple bin operations on a record with the results being returned as a list of (bin-name, result) tuples. The order of the elements in the list will correspond to the order of the operations from the input parameters.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **list** (*list*) – a list of one or more bin operations, each structured as the dict {'bin': bin name, 'op': aerospike.OPERATOR_* [, 'val': value]}. See *aerospike_helpers.operations package*.

- **meta** (*dict*) – optional record metadata to be set, with field `'ttl'` set to `int` number of seconds or one of the *TTL Constants*, and `'gen'` set to `int` generation number to compare.
- **policy** (*dict*) – optional *Operate Policies*.

Returns a *Record Tuple*. See *Unicode Handling*.

Raises a subclass of *AerospikeError*.

Note: In version *2.1.3* the return format of bin entries for this method, **only in cases when a map operation specifying a `return_type` is used**, has changed. Map operations using "return_type" of aerospike.MAP_RETURN_KEY_VALUE will now return a bin value of a list of keys and corresponding values, rather than a list of key/value tuples. See the following code block for details. In addition, some operations which did not return a value in versions <= 2.1.2 will now return a response.

```
# pre 2.1.3 formatting of key/value bin value
[('key1', 'val1'), ('key2', 'val2'), ('key3', 'val3')]

# >= 2.1.3 formatting
['key1', 'val1', 'key2', 'val2', 'key3', 'val3']
```

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
from aerospike_helpers.operations import operations as op_helpers
import sys

config = { 'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

try:
    key = ('test', 'demo', 1)
    policy = {
        'total_timeout': 1000,
        'key': aerospike.POLICY_KEY_SEND,
        'commit_level': aerospike.POLICY_COMMIT_LEVEL_MASTER
    }

    llist = [
        op_helpers.append("name", "aa"),
        op_helpers.read("name"),
        op_helpers.increment("age", 3),
        op_helpers.read("age")
    ]

    client.operate_ordered(key, llist, {}, policy)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
```

Changed in version 2.1.3.

1.2.2.8 Scan and Query

class aerospike.**Client**

scan(*namespace*[, *set*]) → *Scan*

Return a *aerospike.Scan* object to be used for executing scans over a specified *set* (which can be omitted or None) in a *namespace*. A scan with a None set returns all the records in the namespace.

Parameters

- **namespace** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – optional specified set name, otherwise the entire *namespace* will be scanned.

Returns an *aerospike.Scan* class.

query(*namespace*[, *set*]) → *Query*

Return a *aerospike.Query* object to be used for executing queries over a specified *set* (which can be omitted or None) in a *namespace*. A query with a None set returns records which are **not in any named set**. This is different than the meaning of a None set in a scan.

Parameters

- **namespace** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – optional specified set name, otherwise the records which are not part of any *set* will be queried (**Note**: this is different from not providing the *set* in *scan()*).

Returns an *aerospike.Query* class.

1.2.2.9 User Defined Functions

class aerospike.**Client**

udf_put(*filename*[, *udf_type=aerospike.UDF_TYPE_LUA*[, *policy*]])

Register a UDF module with the cluster.

Parameters

- **filename** (*str*) – the path to the UDF module to be registered with the cluster.
- **udf_type** (*int*) – *aerospike.UDF_TYPE_LUA*.
- **policy** (*dict*) – currently **timeout** in milliseconds is the available policy.

Raises a subclass of *AerospikeError*.

Note: Register the UDF module and copy it to the Lua 'user_path', a directory that should contain a copy of the modules registered with the cluster.

```
config = {
    'hosts': [ ('127.0.0.1', 3000)],
    'lua': { 'user_path': '/path/to/lua/user_path'}}
client = aerospike.client(config).connect()
client.udf_put('/path/to/my_module.lua')
client.close()
```

---

udf_remove(*module*[, *policy*])
Remove a previously registered UDF module from the cluster.

Parameters

- **module** (*str*) – the UDF module to be deregistered from the cluster.
- **policy** (*dict*) – currently **timeout** in milliseconds is the available policy.

Raises a subclass of *AerospikeError*.

```
client.udf_remove('my_module.lua')
```

udf_list([*policy*]) →]
Return the list of UDF modules registered with the cluster.

Parameters **policy** (*dict*) – currently **timeout** in milliseconds is the available policy.

Return type list

Raises a subclass of *AerospikeError*.

```
from __future__ import print_function
import aerospike

config = {'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()
print(client.udf_list())
client.close()
```

Note: We expect to see something like:

```
[{'content': bytearray(b''),
  'hash': bytearray(b'195e39ceb51c110950bd'),
  'name': 'my_udf1.lua',
  'type': 0},
 {'content': bytearray(b''),
  'hash': bytearray(b'8a2528e8475271877b3b'),
  'name': 'stream_udf.lua',
  'type': 0},
 {'content': bytearray(b''),
  'hash': bytearray(b'362ea79c8b64857701c2'),
  'name': 'aggregate_udf.lua',
  'type': 0},
 {'content': bytearray(b''),
  'hash': bytearray(b'635f47081431379baa4b'),
  'name': 'module.lua',
  'type': 0}]
```

---

udf_get(*module*[, *language=aerospike.UDF_TYPE_LUA*[, *policy*]]) → str
Return the content of a UDF module which is registered with the cluster.

Parameters

- **module** (*str*) – the UDF module to read from the cluster.
- **udf_type** (*int*) – *aerospike.UDF_TYPE_LUA*

- **policy** (*dict*) – currently **timeout** in milliseconds is the available policy.

Return type str

Raises a subclass of *AerospikeError*.

apply(*key*, *module*, *function*, *args*[, *policy*])

Apply a registered (see *udf_put()*) record UDF to a particular record.

Parameters

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **module** (*str*) – the name of the UDF module.
- **function** (*str*) – the name of the UDF to apply to the record identified by *key*.
- **args** (*list*) – the arguments to the UDF.
- **policy** (*dict*) – optional *Apply Policies*.

Returns the value optionally returned by the UDF, one of str,int, float, bytearray, list, dict.

Raises a subclass of *AerospikeError*.

See also:

Record UDF and Developing Record UDFs.

Note: Version >= 3.10.0 Supports predicate expressions for apply, scan_apply, and query_apply see *predexp*. Requires server version 4.7.0.

```
from __future__ import print_function
import aerospike
from aerospike import predexp
from aerospike import exception as ex
import sys

config = {"hosts": [("127.0.0.1", 3000)]}
client = aerospike.client(config).connect()

# register udf
try:
    client.udf_put("/path/to/my_udf.lua")
except ex.FilteredOut as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    client.close()
    sys.exit(1)


# put records and apply udf
try:
    keys = [("test", "demo", 1), ("test", "demo", 2), ("test", "demo", 3)]
    records = [{"number": 1}, {"number": 2}, {"number": 3}]
    for i in range(3):
        client.put(keys[i], records[i])

    preds = [  # check that the record has value < 2 or == 3 in bin 'name'
```

(continues on next page)

(continued from previous page)

```
        predexp.integer_bin("number"),
        predexp.integer_value(2),
        predexp.integer_less(),
        predexp.integer_bin("number"),
        predexp.integer_value(3),
        predexp.integer_equal(),
        predexp.predexp_or(2),
    ]

    policy = {"predexp": preds}

    client.scan_apply("test", None, "my_udf", "my_udf", ["number", 10], policy)
    records = client.get_many(keys)

    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
# the udf has only modified the records that matched the preds
# EXPECTED OUTPUT:
# [
#   (('test', 'demo', 1, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\x9a\
↪x1e\xac\xc4F\xc8')), {'gen': 2, 'ttl': 2591999}, {'number': 11}),
#   (('test', 'demo', 2, bytearray(b'\xaejQ_7\xdeJ\xda\xccD\x96\xe2\xda\x1f\xea\x84\
↪x8c:\x92p')), {'gen': 12, 'ttl': 2591999}, {'number': 2}),
#   (('test', 'demo', 3, bytearray(b'\xb1\xa5`g\xf6\xd4\xa8\xa4D9\xd3\xafb\xbf\xf8ha\
↪x01\x94\xcd')), {'gen': 13, 'ttl': 2591999}, {'number': 13})
# ]
```

```
# contents of my_udf.lua
function my_udf(rec, bin, offset)
    info("my transform: %s", tostring(record.digest(rec)))
    rec[bin] = rec[bin] + offset
    aerospike:update(rec)
end
```

---

scan_apply(*ns*, *set*, *module*, *function*[, *args*[, *policy*[, *options*]]]) → int

Initiate a scan and apply a record UDF to each record matched by the scan. This method blocks until the scan is complete.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name. Should be `None` if the entire namespace is to be scanned.
- **module** (*str*) – the name of the UDF module.
- **function** (*str*) – the name of the UDF to apply to the records matched by the scan.
- **args** (*list*) – the arguments to the UDF.
- **policy** (*dict*) – optional *Scan Policies*.

- **options** (*dict*) – the *Scan Options* that will apply to the scan.

Return type int

Returns a job ID that can be used with *job_info()* to check the status of the aerospike.JOB_SCAN.

Raises a subclass of *AerospikeError*.

See also:

Record UDF and Developing Record UDFs.

query_apply(*ns*, *set*, *predicate*, *module*, *function*[, *args*[, *policy*]]) → int

Initiate a query and apply a record UDF to each record matched by the query. This method blocks until the query is completed.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name. Should be None if you want to query records in the *ns* which are in no set.
- **predicate** (*tuple*) – the tuple() produced by one of the *aerospike.predicates* methods.
- **module** (*str*) – the name of the UDF module.
- **function** (*str*) – the name of the UDF to apply to the records matched by the query.
- **args** (*list*) – the arguments to the UDF.
- **policy** (*dict*) – optional *Write Policies*.

Return type int

Returns a job ID that can be used with *job_info()* to check the status of the aerospike.JOB_QUERY.

Raises a subclass of *AerospikeError*.

See also:

Record UDF and Developing Record UDFs.

job_info(*job_id*, *module*[, *policy*]) → dict

Return the status of a job running in the background.

Parameters

- **job_id** (*int*) – the job ID returned by *scan_apply()* and *query_apply()*.
- **module** – one of *Job Constants*.

Returns a dict with keys *status*, *records_read*, and *progress_pct*. The value of *status* is one of *Job Statuses*.

Raises a subclass of *AerospikeError*.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import time

config = {'hosts': [ ('127.0.0.1', 3000)]}
```

(continues on next page)

(continued from previous page)

```
client = aerospike.client(config).connect()
try:
    # run the UDF 'add_val' in Lua module 'simple' on the records of test.demo
    job_id = client.scan_apply('test', 'demo', 'simple', 'add_val', ['age', 1])
    while True:
        time.sleep(0.25)
        response = client.job_info(job_id, aerospike.JOB_SCAN)
        if response['status'] == aerospike.JOB_STATUS_COMPLETED:
            break
    print("Job ", str(job_id), " completed")
    print("Progress percentage : ", response['progress_pct'])
    print("Number of scanned records : ", response['records_read'])
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
client.close()
```

scan_info(*scan_id*) → dict

Return the status of a scan running in the background.

Parameters **scan_id** (*int*) – the scan ID returned by *scan_apply()*.

Returns a `dict` with keys *status*, *records_scanned*, and *progress_pct*. The value of *status* is one of *Job Statuses*.

Raises a subclass of *AerospikeError*.

Deprecated since version 1.0.50: Use *job_info()* instead.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex

config = {'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()
try:
    scan_id = client.scan_apply('test', 'demo', 'simple', 'add_val', ['age',␣
↪1])
    while True:
        response = client.scan_info(scan_id)
        if response['status'] == aerospike.SCAN_STATUS_COMPLETED or \
           response['status'] == aerospike.SCAN_STATUS_ABORTED:
            break
    if response['status'] == aerospike.SCAN_STATUS_COMPLETED:
        print("Background scan successful")
        print("Progress percentage : ", response['progress_pct'])
        print("Number of scanned records : ", response['records_scanned'])
        print("Background scan status : ", "SCAN_STATUS_COMPLETED")
    else:
        print("Scan_apply failed")
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
client.close()
```

1.2.2.10 Info Operations

class aerospike.**Client**

get_node_names() →]

Return the list of hosts present in a connected cluster including node names.

Returns a `list` of node info dictionaries.

Raises a subclass of *AerospikeError*.

```
import aerospike

config = {'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

nodes = client.get_node_names()
print(nodes)
client.close()
```

Note: We expect to see something like:

```
[{'address': '1.1.1.1', 'port': 3000, 'node_name': 'BCER199932C'}, {'address':
↪'1.1.1.1', 'port': 3010, 'node_name': 'ADFFE7782CD'}]
```

Changed in version 6.0.0.

get_nodes() →]

Return the list of hosts present in a connected cluster.

Returns a `list` of node address tuples.

Raises a subclass of *AerospikeError*.

```
import aerospike

config = {'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

nodes = client.get_nodes()
print(nodes)
client.close()
```

Note: We expect to see something like:

```
[('127.0.0.1', 3000), ('127.0.0.1', 3010)]
```

Changed in version 3.0.0.

Warning: In versions < 3.0.0 `get_nodes` will not work when using TLS

info(*command*[, *hosts*[, *policy*]]) → {}

Deprecated since version 3.0.0: Use *info_single_node()* to send a request to a single node, or *info_all()* to send a request to the entire cluster. Sending requests to specific nodes can be better handled with a simple Python function such as:

```
def info_to_host_list(client, request, hosts, policy=None):
    output = {}
    for host in hosts:
        try:
            response = client.info_node(request, host, policy)
            output[host] = response
        except Exception as e:
            #  Handle the error gracefully here
            output[host] = e
    return output
```

Send an info *command* to all nodes in the cluster and filter responses to only include nodes specified in a *hosts* list.

Parameters

- **command** (*str*) – the info command.
- **hosts** (*list*) – a `list` containing an *address*, *port* `tuple()`. Example: `[('127.0.0.1', 3000)]`
- **policy** (*dict*) – optional *Info Policies*.

Return type `dict`

Raises a subclass of *AerospikeError*.

See also:

Info Command Reference.

```
import aerospike

config = {'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

response = client.info(command)
client.close()
```

Note: We expect to see something like:

```
{'BB9581F41290C00': (None, '127.0.0.1:3000\n'), 'BC3581F41290C00': (None, '127.
↪0.0.1:3010\n')}
```

Changed in version 3.0.0.

info_all(*command[, policy]]*) → {}

Send an info *command* to all nodes in the cluster to which the client is connected. If any of the individual requests fail, this will raise an exception.

Parameters

- **command** (*str*) – the info command.

- **policy** (*dict*) – optional *Info Policies*.

Return type dict

Raises a subclass of *AerospikeError*.

See also:

Info Command Reference.

```
import aerospike

config = {'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect()

response = client.info_all(command)
client.close()
```

Note: We expect to see something like:

```
{'BB9581F41290C00': (None, '127.0.0.1:3000\n'), 'BC3581F41290C00': (None, '127.
↪0.0.1:3010\n')}
```

New in version 3.0.0.

info_node(*command*, *host*[, *policy*]) → str

Deprecated since version 6.0.0: Use *info_single_node()* to send a request to a single node, or *info_all()* to send a request to the entire cluster.

Send an info *command* to a single node specified by *host*.

Parameters

- **command** (*str*) – the info command.
- **host** (*tuple*) – a tuple() containing an *address*, *port* , optional *tls-name* . Example: `('127.0.0.1', 3000)` or when using TLS `('127.0.0.1', 4333, 'server-tls-name')`. In order to send an info request when TLS is enabled, the *tls-name* must be present.
- **policy** (*dict*) – optional *Info Policies*.

Return type str

Raises a subclass of *AerospikeError*.

See also:

Info Command Reference.

Changed in version 3.0.0.

Warning: for client versions < 3.0.0 `info_node` will not work when using TLS

info_single_node(*command*, *host*[, *policy*]) → str

Send an info *command* to a single node specified by *host name*.

Parameters

- **command** (*str*) – the info command.
- **host** (*tuple*) – a `str` containing a node name. Example: 'BCER199932C'
- **policy** (*dict*) – optional *Info Policies*.

Return type `str`

Raises a subclass of *AerospikeError*.

Note: Use *get_node_names()* as an easy way to get host IP to node name mappings.

See also:

Info Command Reference.

info_random_node(*command*[, *policy*]) → str

Send an info *command* to a single random node.

Parameters

- **command** (*str*) – the info command.
- **policy** (*dict*) – optional *Info Policies*.

Return type `str`

Raises a subclass of *AerospikeError*.

See also:

Info Command Reference.

Changed in version 6.0.0.

set_xdr_filter(*data_center*, *namespace*, *expression_filter*[, *policy*]) → str

Set the cluster's xdr filter using an Aerospike expression. The cluster's current filter can be removed by setting expression_filter to None.

Parameters

- **data_center** (*str*) – The data center to apply the filter to.
- **namespace** (*str*) – The namespace to apply the filter to.
- **expression_filter** (*expression*) – The filter to set. See expressions at *aerospike_helpers*.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

See also:

xdr-set-filter Info Command Reference.

Changed in version 5.0.0.

Warning: Requires Aerospike server version >= 5.3.

shm_key() → int

Expose the value of the shm_key for this client if shared-memory cluster tending is enabled,

Return type `int` or `None`

truncate(*namespace*, *set*, *nanos*[, *policy*])

Remove records in specified namespace/set efficiently. This method is many orders of magnitude faster than deleting records one at a time. See Truncate command reference.

Note: Requires Aerospike Server versions >= 3.12

This asynchronous server call may return before the truncation is complete. The user can still write new records after the server returns because new records will have last update times greater than the truncate cutoff (set at the time of truncate call)

Parameters

- **namespace** (*str*) – The namespace on which the truncation operation should be performed.
- **set** (*str*) – The set to truncate. Pass in `None` to indicate that all records in the namespace should be truncated.
- **nanos** (*long*) – A cutoff threshold indicating that records last updated before the threshold will be removed.Units are in nanoseconds since unix epoch (1970-01-01). A value of `0` indicates that all records in the set should be truncated regardless of update time. The value must not be in the future.
- **policy** (*dict*) – Optional *Info Policies*

Return type Status indicating the success of the operation.

Raises a subclass of *AerospikeError*.

```
import aerospike
import time

client = aerospike.client({'hosts': [('localhost', 3000)]}).connect()

# Store 10 items in the database
for i in range(10):
    key = ('test', 'truncate', i)
    record = {'item': i}
    client.put(key, record)

time.sleep(2)
current_time = time.time()
# Convert the current time to nanoseconds since epoch
threshold_ns = int(current_time * 10 ** 9)

time.sleep(2)  # Make sure some time passes before next round of additions

# Store another 10 items into the database
for i in range(10, 20):
    key = ('test', 'truncate', i)
    record = {'item': i}
    client.put(key, record)

# Store a record in the 'test' namespace without a set
key = ('test', None, 'no set')
record = ({'item': 'no set'})
```

(continues on next page)

(continued from previous page)

```
client.put(key, record)

# Remove all items created before the threshold time
# The first 10 records we added will be removed by this call.
# The second 10 will remain.
client.truncate('test', 'truncate', threshold_ns)


# Remove all records from test/truncate.
# After this the record with key ('test', None, 'no set') still exists
client.truncate('test', 'truncate', 0)

# Remove all records from the test namespace
client.truncate('test', None, 0)

client.close()
```

1.2.2.11 Index Operations

class aerospike.**Client**

index_string_create(*ns*, *set*, *bin*, *index_name*[, *policy*])
Create a string index with *index_name* on the *bin* in the specified *ns*, *set*.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **bin** (*str*) – the name of bin the secondary index is built on.
- **index_name** (*str*) – the name of the index.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

index_integer_create(*ns*, *set*, *bin*, *index_name*[, *policy*])
Create an integer index with *index_name* on the *bin* in the specified *ns*, *set*.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **bin** (*str*) – the name of bin the secondary index is built on.
- **index_name** (*str*) – the name of the index.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

index_list_create(*ns*, *set*, *bin*, *index_datatype*, *index_name*[, *policy*])
Create an index named *index_name* for numeric, string or GeoJSON values (as defined by *index_datatype*) on records of the specified *ns*, *set* whose *bin* is a list.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **bin** (*str*) – the name of bin the secondary index is built on.
- **index_datatype** – Possible values are `aerospike.INDEX_STRING`, `aerospike.INDEX_NUMERIC` and `aerospike.INDEX_GEO2DSPHERE`.
- **index_name** (*str*) – the name of the index.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

Note: Requires server version >= 3.8.0

---

index_map_keys_create(*ns*, *set*, *bin*, *index_datatype*, *index_name*[, *policy*])

Create an index named *index_name* for numeric, string or GeoJSON values (as defined by *index_datatype*) on records of the specified *ns*, *set* whose *bin* is a map. The index will include the keys of the map.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **bin** (*str*) – the name of bin the secondary index is built on.
- **index_datatype** – Possible values are `aerospike.INDEX_STRING`, `aerospike.INDEX_NUMERIC` and `aerospike.INDEX_GEO2DSPHERE`.
- **index_name** (*str*) – the name of the index.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

Note: Requires server version >= 3.8.0

---

index_map_values_create(*ns*, *set*, *bin*, *index_datatype*, *index_name*[, *policy*])

Create an index named *index_name* for numeric, string or GeoJSON values (as defined by *index_datatype*) on records of the specified *ns*, *set* whose *bin* is a map. The index will include the values of the map.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **bin** (*str*) – the name of bin the secondary index is built on.
- **index_datatype** – Possible values are `aerospike.INDEX_STRING`, `aerospike.INDEX_NUMERIC` and `aerospike.INDEX_GEO2DSPHERE`.
- **index_name** (*str*) – the name of the index.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

Note: Requires server version >= 3.8.0

```
import aerospike

client = aerospike.client({ 'hosts': [ ('127.0.0.1', 3000)]}).connect()

# assume the bin fav_movies in the set test.demo bin should contain
# a dict { (str) _title_ : (int) _times_viewed_ }
# create a secondary index for string values of test.demo records whose 'fav_
↪movies' bin is a map
client.index_map_keys_create('test', 'demo', 'fav_movies', aerospike.INDEX_
↪STRING, 'demo_fav_movies_titles_idx')
# create a secondary index for integer values of test.demo records whose 'fav_
↪movies' bin is a map
client.index_map_values_create('test', 'demo', 'fav_movies', aerospike.INDEX_
↪NUMERIC, 'demo_fav_movies_views_idx')
client.close()
```

index_geo2dsphere_create(*ns*, *set*, *bin*, *index_name*[, *policy*])

Create a geospatial 2D spherical index with *index_name* on the *bin* in the specified *ns*, *set*.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **set** (*str*) – the set name.
- **bin** (*str*) – the name of bin the secondary index is built on.
- **index_name** (*str*) – the name of the index.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

See also:

aerospike.GeoJSON, *aerospike.predicates*

Note: Requires server version >= 3.7.0

```
import aerospike

client = aerospike.client({ 'hosts': [ ('127.0.0.1', 3000)]}).connect()
client.index_geo2dsphere_create('test', 'pads', 'loc', 'pads_loc_geo')
client.close()
```

index_remove(*ns*, *index_name*[, *policy*])

Remove the index with *index_name* from the namespace.

Parameters

- **ns** (*str*) – the namespace in the aerospike cluster.
- **index_name** (*str*) – the name of the index.
- **policy** (*dict*) – optional *Info Policies*.

Raises a subclass of *AerospikeError*.

1.2.2.12 Admin Operations

class aerospike.**Client**

Note: The admin methods implement the security features of the Enterprise Edition of Aerospike. These methods will raise a *SecurityNotSupported* when the client is connected to a Community Edition cluster (see *aerospike.exception*).

A user is validated by the client against the server whenever a connection is established through the use of a username and password (passwords hashed using bcrypt). When security is enabled, each operation is validated against the user's roles. Users are assigned roles, which are collections of *Privilege Objects*.

```
import aerospike
from aerospike import exception as ex
import time

config = {'hosts': [('127.0.0.1', 3000)] }
client = aerospike.client(config).connect('ipji', 'life is good')

try:
    dev_privileges = [{'code': aerospike.PRIV_READ}, {'code': aerospike.PRIV_READ_
↪WRITE}]
    client.admin_create_role('dev_role', dev_privileges)
    client.admin_grant_privileges('dev_role', [{'code': aerospike.PRIV_READ_WRITE_
↪UDF}])
    client.admin_create_user('dev', 'you young whatchacallit... idiot', ['dev_role
↪'])
    time.sleep(1)
    print(client.admin_query_user('dev'))
    print(admin_query_users())
except ex.AdminError as e:
    print("Error [{0}]: {1}".format(e.code, e.msg))
client.close()
```

See also:

Security features article.

---

admin_create_role(*role*, *privileges*[, *policy*[, *whitelist*[, *read_quota*[, *write_quota*]]]])

Create a custom, named *role* containing a `list` of *privileges*, optional whitelist, and quotas.

Parameters

- **role** (*str*) – The name of the role.
- **privileges** (*list*) – A list of *Privilege Objects*.
- **policy** (*dict*) – Optional *Admin Policies*.
- **whitelist** (*list*) – A list of whitelist IP addresses that can contain wildcards, for example 10.1.2.0/24.
- **read_quota** (*int*) – Maximum reads per second limit, pass in zero for no limit.

- **write_quota** (*int*) – Maximum write per second limit, pass in zero for no limit.

Raises One of the *AdminError* subclasses.

admin_set_whitelist(*role*, *whitelist*[, *policy*])
Add *whitelist* to a *role*.

Parameters

- **role** (*str*) – The name of the role.
- **whitelist** (*list*) – List of IP strings the role is allowed to connect to. Setting whitlist to None will clear the whitelist for that role.
- **policy** (*dict*) – Optional *Admin Policies*.

Raises One of the *AdminError* subclasses.

admin_set_quotas(*role*[, *read_quota*[, *write_quota*[, *policy*]]])
Add *quotas* to a *role*.

Parameters

- **role** (*str*) – the name of the role.
- **read_quota** (*int*) – Maximum reads per second limit, pass in zero for no limit.
- **write_quota** (*int*) – Maximum write per second limit, pass in zero for no limit.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_drop_role(*role*[, *policy*])
Drop a custom *role*.

Parameters

- **role** (*str*) – the name of the role.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_grant_privileges(*role*, *privileges*[, *policy*])
Add *privileges* to a *role*.

Parameters

- **role** (*str*) – the name of the role.
- **privileges** (*list*) – a list of *Privilege Objects*.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_revoke_privileges(*role*, *privileges*[, *policy*])
Remove *privileges* from a *role*.

Parameters

- **role** (*str*) – the name of the role.
- **privileges** (*list*) – a list of *Privilege Objects*.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_get_role(*role*[, *policy*]) →]
Get the dict of privileges, whitelist, and quotas associated with a *role*.

Parameters

- **role** (*str*) – the name of the role.
- **policy** (*dict*) – optional *Admin Policies*.

Returns a *Privilege Objects*.

Raises one of the *AdminError* subclasses.

admin_get_roles([*policy*]) → {}
Get all named roles and their attributes.

Parameters **policy** (*dict*) – optional *Admin Policies*.

Returns a dict of *Privilege Objects* keyed by role name.

Raises one of the *AdminError* subclasses.

admin_query_role(*role*[, *policy*]) →]
Get the list of privileges associated with a *role*.

Parameters

- **role** (*str*) – the name of the role.
- **policy** (*dict*) – optional *Admin Policies*.

Returns a list of *Privilege Objects*.

Raises one of the *AdminError* subclasses.

admin_query_roles([*policy*]) → {}
Get all named roles and their privileges.

Parameters **policy** (*dict*) – optional *Admin Policies*.

Returns a dict of *Privilege Objects* keyed by role name.

Raises one of the *AdminError* subclasses.

admin_create_user(*username*, *password*, *roles*[, *policy*])
Create a user with a specified *username* and grant it *roles*.

Parameters

- **username** (*str*) – the username to be added to the aerospike cluster.
- **password** (*str*) – the password associated with the given username.
- **roles** (*list*) – the list of role names assigned to the user.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_drop_user(*username*[, *policy*])
Drop the user with a specified *username* from the cluster.

Parameters

- **username** (*str*) – the username to be dropped from the aerospike cluster.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_change_password(*username*, *password*[, *policy*])
Change the *password* of the user *username*. This operation can only be performed by that same user.

Parameters

- **username** (*str*) – the username.
- **password** (*str*) – the password associated with the given username.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_set_password(*username*, *password*[, *policy*])
Set the *password* of the user *username* by a user administrator.

Parameters

- **username** (*str*) – the username to be added to the aerospike cluster.
- **password** (*str*) – the password associated with the given username.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_grant_roles(*username*, *roles*[, *policy*])
Add *roles* to the user *username*.

Parameters

- **username** (*str*) – the username to be granted the roles.
- **roles** (*list*) – a list of role names.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_revoke_roles(*username*, *roles*[, *policy*])
Remove *roles* from the user *username*.

Parameters

- **username** (*str*) – the username to have the roles revoked.
- **roles** (*list*) – a list of role names.
- **policy** (*dict*) – optional *Admin Policies*.

Raises one of the *AdminError* subclasses.

admin_query_user(*username*[, *policy*]) →]
Return the list of roles granted to the specified user *username*.

Parameters

- **username** (*str*) – the username to query for.
- **policy** (*dict*) – optional *Admin Policies*.

Returns a `list` of role names.

Raises one of the *AdminError* subclasses.

admin_query_users([*policy*]) → {}
Return the `dict` of users, with their roles keyed by username.

Parameters **policy** (*dict*) – optional *Admin Policies*.

Returns a `dict` of roles keyed by username.

Raises one of the *AdminError* subclasses.

1.2.3 Policies

1.2.3.1 Write Policies

policy

A `dict` of optional write policies, which are applicable to *put()*, *query_apply()*. *remove_bin()*.

aggedright

- **max_retries** (`int`)
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`.

  Default: `0`

  Warning: Database writes that are not idempotent (such as "add") should not be retried because the write operation may be performed multiple times if the client timed out previous transaction attempts. It's important to use a distinct write policy for non-idempotent writes, which sets max_retries = *0*;

- **sleep_between_retries** (`int`)
  Milliseconds to sleep between retries. Enter `0` to skip sleep.

  Default: `0`
- **socket_timeout** (`int`)
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not `0` and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both `socket_timeout` and `total_timeout` are non-zero and `socket_timeout` > `total_timeout`, then `socket_timeout` will be set to `total_timeout`. If `socket_timeout` is `0`, there will be no socket idle limit.

  Default: `0`
- **total_timeout** (`int`)
  Total transaction timeout in milliseconds.

  The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

  If `total_timeout` is not `0` and `total_timeout` is reached before the transaction completes, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`. If `total_timeout` is `0`, there will be no total time limit.

  Default: `1000`
- **compress** (`bool`)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

  This option will increase cpu and memory usage (for extra compressed buffers), but decrease the size of data sent over the network.

  Default: `False`
- **key**

One of the *Key Policy Options* values such as *aerospike.POLICY_KEY_DIGEST*

Default: *aerospike.POLICY_KEY_DIGEST*

- **exists**
  One of the *Existence Policy Options* values such as *aerospike.POLICY_EXISTS_CREATE*

  Default: *aerospike.POLICY_EXISTS_IGNORE*
- **gen**
  One of the *Generation Policy Options* values such as *aerospike.POLICY_GEN_IGNORE*

  Default: *aerospike.POLICY_GEN_IGNORE*
- **commit_level**
  One of the *Commit Level Policy Options* values such as *aerospike.POLICY_COMMIT_LEVEL_ALL*

  Default: *aerospike.POLICY_COMMIT_LEVEL_ALL*
- **durable_delete** (`bool`)
  Perform durable delete

  Default: `False`
- **expressions** `list`
  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

  Note: Requires Aerospike server version >= 5.2.

1.2.3.2 Read Policies

`policy`
A `dict` of optional read policies, which are applicable to *get()*, *exists()*, *select()*.

aggedright

- **max_retries** (`int`)
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`.

  Default: `2`
- **sleep_between_retries** (`int`)
  Milliseconds to sleep between retries. Enter `0` to skip sleep.

  Default: `0`
- **socket_timeout** (`int`)
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not `0` and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both `socket_timeout` and `total_timeout` are non-zero and `socket_timeout` > `total_timeout`, then `socket_timeout` will be set to `total_timeout`. If `socket_timeout` is `0`, there will be no socket idle limit.

  Default: `0`
- **total_timeout** (`int`)
  Total transaction timeout in milliseconds.

  The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

  If `total_timeout` is not `0` and `total_timeout` is reached before the transaction completes, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`. If `total_timeout` is `0`, there will be no total time limit.

Default: `1000`

- **compress** (`bool`)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

  This option will increase cpu and memory usage (for extra compressed buffers), but decrease the size of data sent over the network.

  Default: `False`
- **deserialize** (`bool`)
  Should raw bytes representing a list or map be deserialized to a list or dictionary. Set to *False* for backup programs that just need access to raw bytes.
  Default: `True`
- **key**
  One of the *Key Policy Options* values such as *aerospike.POLICY_KEY_DIGEST*

  Default: *aerospike.POLICY_KEY_DIGEST*
- **read_mode_ap**
  One of the *AP Read Mode Policy Options* values such as `aerospike.AS_POLICY_READ_MODE_AP_ONE`

  Default: `aerospike.AS_POLICY_READ_MODE_AP_ONE`
  New in version 3.7.0.
- **read_mode_sc**
  One of the *SC Read Mode Policy Options* values such as *aerospike.POLICY_READ_MODE_SC_SESSION*

  Default: *aerospike.POLICY_READ_MODE_SC_SESSION*
  New in version 3.7.0.
- **replica**
  One of the *Replica Options* values such as *aerospike.POLICY_REPLICA_MASTER*

  Default: `aerospike.POLICY_REPLICA_SEQUENCE`
- **expressions** `list`
  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

  Note: Requires Aerospike server version >= 5.2.

1.2.3.3 Operate Policies

policy
A `dict` of optional operate policies, which are applicable to *append()*, *prepend()*, *increment()*, *operate()*, and atomic list and map operations.

aggedright

- **max_retries** (`int`)
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`.

  Default: `0`

  Warning: Database writes that are not idempotent (such as "add") should not be retried because the write operation may be performed multiple times if the client timed out previous transaction attempts. It's important to use a distinct write policy for non-idempotent writes, which sets max_retries = *0*;
- **sleep_between_retries** (`int`)
  Milliseconds to sleep between retries. Enter `0` to skip sleep.

Default: `0`

- **socket_timeout** (`int`)
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not `0` and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both `socket_timeout` and `total_timeout` are non-zero and `socket_timeout` > `total_timeout`, then `socket_timeout` will be set to `total_timeout`. If `socket_timeout` is `0`, there will be no socket idle limit.

  Default: `0`
- **total_timeout** (`int`)
  Total transaction timeout in milliseconds.

  The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

  If `total_timeout` is not `0` and `total_timeout` is reached before the transaction completes, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`. If `total_timeout` is `0`, there will be no total time limit.

  Default: `1000`
- **compress** (`bool`)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

  This option will increase cpu and memory usage (for extra compressed buffers), but decrease the size of data sent over the network.

  Default: `False`
- **key**
  One of the *Key Policy Options* values such as *aerospike.POLICY_KEY_DIGEST*

  Default: *aerospike.POLICY_KEY_DIGEST*
- **gen**
  One of the *Generation Policy Options* values such as *aerospike.POLICY_GEN_IGNORE*

  Default: *aerospike.POLICY_GEN_IGNORE*
- **replica**
  One of the *Replica Options* values such as *aerospike.POLICY_REPLICA_MASTER*

  Default: *aerospike.POLICY_REPLICA_SEQUENCE*
- **commit_level**
  One of the *Commit Level Policy Options* values such as *aerospike.POLICY_COMMIT_LEVEL_ALL*

  Default: *aerospike.POLICY_COMMIT_LEVEL_ALL*
- **read_mode_ap**
  One of the *AP Read Mode Policy Options* values such as `aerospike.AS_POLICY_READ_MODE_AP_ONE`

  Default: `aerospike.AS_POLICY_READ_MODE_AP_ONE`
  New in version 3.7.0.
- **read_mode_sc**
  One of the *SC Read Mode Policy Options* values such as *aerospike.POLICY_READ_MODE_SC_SESSION*

  Default: *aerospike.POLICY_READ_MODE_SC_SESSION*
  New in version 3.7.0.
- **exists**
  One of the *Existence Policy Options* values such as *aerospike.POLICY_EXISTS_CREATE*

Default: *aerospike.POLICY_EXISTS_IGNORE*

- **durable_delete** (bool)
  Perform durable delete

  Default: False
- **expressions** list
  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

  Note: Requires Aerospike server version >= 5.2.

1.2.3.4 Apply Policies

policy
A dict of optional apply policies, which are applicable to *apply()*.

aggedright

- **max_retries** (int)
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error AEROSPIKE_ERR_TIMEOUT.

  Default: 0

  Warning: Database writes that are not idempotent (such as "add") should not be retried because the write operation may be performed multiple times if the client timed out previous transaction attempts. It's important to use a distinct write policy for non-idempotent writes, which sets max_retries = *0*;
- **sleep_between_retries** (int)
  Milliseconds to sleep between retries. Enter 0 to skip sleep.

  Default: 0
- **socket_timeout** (int)
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not 0 and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both socket_timeout and total_timeout are non-zero and socket_timeout > total_timeout, then socket_timeout will be set to total_timeout. If socket_timeout is 0, there will be no socket idle limit.

  Default: 0
- **total_timeout** (int)
  Total transaction timeout in milliseconds.

  The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

  If total_timeout is not 0 and total_timeout is reached before the transaction completes, the transaction will return error AEROSPIKE_ERR_TIMEOUT. If total_timeout is 0, there will be no total time limit.

  Default: 1000
- **compress** (bool)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

  This option will increase cpu and memory

usage (for extra compressed buffers), but decrease the size of data sent over the network.

Default: `False`

- **key**
  One of the *Key Policy Options* values such as *aerospike.POLICY_KEY_DIGEST*

  Default: *aerospike.POLICY_KEY_DIGEST*
- **replica**
  One of the *Replica Options* values such as *aerospike.POLICY_REPLICA_MASTER*

  Default: *aerospike.POLICY_REPLICA_SEQUENCE*
- **gen**
  One of the *Generation Policy Options* values such as *aerospike.POLICY_GEN_IGNORE*

  Default: *aerospike.POLICY_GEN_IGNORE*
- **commit_level**
  One of the *Commit Level Policy Options* values such as *aerospike.POLICY_COMMIT_LEVEL_ALL*

  Default: *aerospike.POLICY_COMMIT_LEVEL_ALL*
- **durable_delete** (`bool`)
  Perform durable delete

  Default: `False`
- **expressions** `list`
  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

  Note: Requires Aerospike server version >= 5.2.

1.2.3.5 Remove Policies

policy
A `dict` of optional remove policies, which are applicable to *remove()*.

aggedright

- **max_retries** (`int`)
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`.

  Default: `0`

  Warning: Database writes that are not idempotent (such as "add") should not be retried because the write operation may be performed multiple times if the client timed out previous transaction attempts. It's important to use a distinct write policy for non-idempotent writes, which sets max_retries = *0*;
- **sleep_between_retries** (`int`)
  Milliseconds to sleep between retries. Enter `0` to skip sleep.
  Default: `0`
- **socket_timeout** (`int`)
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not `0` and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both `socket_timeout` and `total_timeout` are non-zero and `socket_timeout` > `total_timeout`, then `socket_timeout` will be set to `total_timeout`. If `socket_timeout` is `0`, there will be no socket idle limit.

  Default: `0`
- **total_timeout** (`int`)

Total transaction timeout in milliseconds.

The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

If `total_timeout` is not `0` and `total_timeout` is reached before the transaction completes, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`. If `total_timeout` is `0`, there will be no total time limit.

Default: `1000`

- **compress** (`bool`)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

  This option will increase cpu and memory usage (for extra compressed buffers), but decrease the size of data sent over the network.

  Default: `False`
- **key**
  One of the *Key Policy Options* values such as *aerospike.POLICY_KEY_DIGEST*

  Default: *aerospike.POLICY_KEY_DIGEST*
- **commit_level**
  One of the *Commit Level Policy Options* values such as *aerospike.POLICY_COMMIT_LEVEL_ALL*

  Default: *aerospike.POLICY_COMMIT_LEVEL_ALL*
- **gen**
  One of the *Generation Policy Options* values such as *aerospike.POLICY_GEN_IGNORE*

  Default: *aerospike.POLICY_GEN_IGNORE*
- **durable_delete** (`bool`)
  Perform durable delete

  Default: `False`

  Note: Requires Enterprise server version >= 3.10

- **replica**
  One of the *Replica Options* values such as *aerospike.POLICY_REPLICA_MASTER*

  Default: `aerospike.POLICY_REPLICA_SEQUENCE`
- **expressions** `list`
  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

  Note: Requires Aerospike server version >= 5.2.

1.2.3.6 Batch Policies

policy
A `dict` of optional batch policies, which are applicable to *get_many()*, *exists_many()* and *select_many()*.

aggedright

- **max_retries** (`int`)
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`.

  Default: `2`

- **sleep_between_retries** (int)
  Milliseconds to sleep between retries. Enter 0 to skip sleep.

  Default: 0
- **socket_timeout** (int)
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not 0 and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both socket_timeout and total_timeout are non-zero and socket_timeout > total_timeout, then socket_timeout will be set to total_timeout. If socket_timeout is 0, there will be no socket idle limit.

  Default: 0
- **total_timeout** (int)
  Total transaction timeout in milliseconds.

  The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

  If total_timeout is not 0 and total_timeout is reached before the transaction completes, the transaction will return error AEROSPIKE_ERR_TIMEOUT. If total_timeout is 0, there will be no total time limit.

  Default: 1000
- **compress** (bool)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

  This option will increase cpu and memory usage (for extra compressed buffers), but decrease the size of data sent over the network.

  Default: False
- **read_mode_ap**
  One of the *AP Read Mode Policy Options* values such as aerospike.AS_POLICY_READ_MODE_AP_ONE

  Default: aerospike.AS_POLICY_READ_MODE_AP_ONE
  New in version 3.7.0.
- **read_mode_sc**
  One of the *SC Read Mode Policy Options* values such as *aerospike.POLICY_READ_MODE_SC_SESSION*

  Default: *aerospike.POLICY_READ_MODE_SC_SESSION*
  New in version 3.7.0.
- **replica**
  One of the *Replica Options* values such as *aerospike.POLICY_REPLICA_MASTER*

  Default: *aerospike.POLICY_REPLICA_SEQUENCE*
- **concurrent** (bool)
  Determine if batch commands to each server are run in parallel threads.

  Default False
- **allow_inline** (bool)
  Allow batch to be processed immediately in the server's receiving thread when the server deems it to be appropriate. If *False*, the batch will always be processed in separate transaction threads. This field is only relevant for the new batch index protocol.

  Default True
- **send_set_name** (bool)
  Send set name field to server for every key in the batch for batch index protocol. This is only necessary when authentication is enabled and security roles are defined on a per set basis.

  Default: False

- **deserialize** (bool)
  Should raw bytes be deserialized to as_list or as_map. Set to *False* for backup programs that just need access to raw bytes.

  Default: True

- **predexp** list
  A list of *aerospike.predexp* used as a predicate filter for record, bin, batch, and record UDF operations.

  Default: None

1.2.3.7 Info Policies

policy
A dict of optional info policies, which are applicable to *info_all()*, *info_single_node()*, *info_random_node()* and index operations.

aggedright

- **timeout** (int)
  Read timeout in milliseconds

1.2.3.8 Admin Policies

policy
A dict of optional admin policies, which are applicable to admin (security) operations.

aggedright

- **timeout** (int)
  Admin operation timeout in milliseconds

1.2.3.9 List Policies

policy
A dict of optional list policies, which are applicable to list operations.

aggedright

- **write_flags**
  Write flags for the operation.
  One of the *List Write Flags* values such as *aerospike.LIST_WRITE_DEFAULT*

  Default: *aerospike.LIST_WRITE_DEFAULT*

  Values should be or'd together: aerospike.LIST_WRITE_ADD_UNIQUE | aerospike.LIST_WRITE_INSERT_BOUNDED
- **list_order**
  Ordering to maintain for the list.
  One of *List Order*, such as *aerospike.LIST_ORDERED*

  Default: *aerospike.LIST_UNORDERED*

Example:

```
list_policy = {
    "write_flags": aerospike.LIST_WRITE_ADD_UNIQUE | aerospike.LIST_WRITE_
↪INSERT_BOUNDED,
    "list_order": aerospike.LIST_ORDERED
}
```

1.2.3.10 Map Policies

policy
A dict of optional map policies, which are applicable to map operations. Only one of map_write_mode or map_write_flags should be provided. map_write_mode should be used for Aerospike Server versions < *4.3.0* and map_write_flags should be used for Aerospike server versions greater than or equal to *4.3.0* .

aggedright

- **map_write_mode**
  Write mode for the map operation.
  One of the *Map Write Mode* values such as *aerospike.MAP_UPDATE*

  Default: *aerospike.MAP_UPDATE*

  Note: This should only be used for Server version < 4.3.0.

- **map_write_flags**
  Write flags for the map operation.
  One of the *Map Write Flag* values such as *aerospike.MAP_WRITE_FLAGS_DEFAULT*

  Default: *aerospike.MAP_WRITE_FLAGS_DEFAULT*

  Values should be or'd together:
  aerospike.LIST_WRITE_ADD_UNIQUE | aerospike.LIST_WRITE_INSERT_BOUNDED

  Note: This is only valid for Aerospike Server versions >= 4.3.0.

- **map_order**
  Ordering to maintain for the map entries.
  One of *Map Order*, such as *aerospike.MAP_KEY_ORDERED*

  Default: *aerospike.MAP_UNORDERED*

Example:

```
# Server >= 4.3.0
map_policy = {
    'map_order': aerospike.MAP_UNORDERED,
    'map_write_flags': aerospike.MAP_WRITE_FLAGS_CREATE_ONLY
}

# Server < 4.3.0
map_policy = {
    'map_order': aerospike.MAP_UNORDERED,
    'map_write_mode': aerospike.MAP_CREATE_ONLY
}
```

1.2.3.11 Bit Policies

policy
A dict of optional bit policies, which are applicable to bit operations.

Note: Requires server version >= 4.6.0

aggedright

- **bit_write_flags**
  Write flags for the bit operation.
  One of the *Bitwise Write Flags* values such as *aerospike.BIT_WRITE_DEFAULT*

  Default: *aerospike.BIT_WRITE_DEFAULT*

Example:

```
bit_policy = {
    'bit_write_flags': aerospike.BIT_WRITE_UPDATE_ONLY
}
```

1.2.3.12 HyperLogLog Policies

policy
A dict of optional HyperLogLog policies, which are applicable to bit operations.

Note: Requires server version >= 4.9.0

aggedright

- **flags**
  Write flags for the HLL operation.
  One of the *HyperLogLog Write Flags* values such as *aerospike.HLL_WRITE_DEFAULT*

  Default: *aerospike.HLL_WRITE_DEFAULT*

Example:

```
HLL_policy = {
    'flags': aerospike.HLL_WRITE_UPDATE_ONLY
}
```

1.2.4 Misc

1.2.4.1 Role Objects

Role Dictionary
A dict describing attributes associated with a specific role.

aggedright

- **privileges** A list of *Privilege Objects*.
- **whitelist** A list of IP address strings.
- **read_quota** A int representing the allowed read transactions per second.
- **write_quota** A int representing the allowed write transactions per second.

Example:

```
{'code': aerospike.PRIV_READ, 'ns': 'test', 'set': 'demo'}
```

1.2.4.2 Privilege Objects

privilege
A `dict` describing a privilege associated with a specific role.

aggedright

- **code** one of the *Privileges* values such as *aerospike.PRIV_READ*
- **ns** optional namespace, to which the privilege applies, otherwise the privilege applies globally.
- **set** optional set within the *ns*, to which the privilege applies, otherwise to the entire namespace.

Example:

```
{'code': aerospike.PRIV_READ, 'ns': 'test', 'set': 'demo'}
```

1.2.4.3 Unicode Handling

Both `str` and `unicode()` values are converted by the client into UTF-8 encoded strings for storage on the aerospike server. Read methods such as *get()*, *query()*, *scan()* and *operate()* will return that data as UTF-8 encoded `str` values. To get a `unicode()` you will need to manually decode.

Warning: Prior to release 1.0.43 read operations always returned strings as `unicode()`.

```
>>> client.put(key, { 'name': 'Dr. Zeta Alphabeta', 'age': 47})
>>> (key, meta, record) = client.get(key)
>>> type(record['name'])
<type 'str'>
>>> record['name']
'Dr. Zeta Alphabeta'
>>> client.put(key, { 'name': unichr(0x2603), 'age': 21})
>>> (key, meta, record) = client.get(key)
>>> type(record['name'])
<type 'str'>
>>> record['name']
'\xe2\x98\x83'
>>> print(record['name'])

>>> name = record['name'].decode('utf-8')
>>> type(name)
<type 'unicode'>
>>> name
u'\u2603'
>>> print(name)
```

1.3 Scan Class — Scan

1.3.1 Scan

The Scan object is used to return all the records in a specified set (which can be ommitted or None). A Scan with a None set returns all the records in the namespace.

The scan is invoked using `foreach()`, `results()`, or `execute_background()`. The bins returned can be filtered using `select()`.

See also:

Scans and Managing Scans.

1.3.1.1 Scan Methods

class aerospike.**Scan**

select(*bin1*[, *bin2*[, *bin3*..]])
Set a filter on the record bins resulting from *results()* or *foreach()*. If a selected bin does not exist in a record it will not appear in the *bins* portion of that record tuple.

apply(*module*, *function*[, *arguments*])
Apply a record UDF to each record found by the scan UDF.

Parameters

- **module** (*str*) – the name of the Lua module.
- **function** (*str*) – the name of the Lua function within the *module*.
- **arguments** (*list*) – optional arguments to pass to the *function*. NOTE: these arguments must be types supported by Aerospike See: supported data types. If you need to use an unsuported type, (e.g. set or tuple) you can use a serializer such as pickle first.

Returns one of the supported types, int, str, float (double), list, dict (map), bytearray (bytes).

See also:

Developing Record UDFs

add_ops(*ops*)
Add a list of write ops to the scan. When used with *Scan.execute_background()* the scan will perform the write ops on any records found. If no predicate is attached to the scan it will apply ops to all the records in the specified set. See *aerospike_helpers* for available ops.

Parameters **ops** – *list* A list of write operations generated by the aerospike_helpers e.g. list_operations, map_operations, etc.

Note: Requires server version >= 4.7.0.

```
import aerospike
from aerospike_helpers.operations import list_operations
from aerospike_helpers.operations import operations
```

(continues on next page)

(continued from previous page)

```
scan = client.scan('test', 'demo')

ops =  [
    operations.append(test_bin, 'val_to_append'),
    list_operations.list_remove_by_index(test_bin, list_index_to_remove,␣
↪aerospike.LIST_RETURN_NONE)
]
scan.add_ops(ops)

id = scan.execute_background()
client.close()
```

For a more comprehensive example, see using a list of write ops with *Query.execute_background()* .

results(*[policy[, nodename]]) -> list of (key, meta, bins*)

Buffer the records resulting from the scan, and return them as a `list` of records.

Parameters

- **policy** (*dict*) – optional *Scan Policies*.
- **nodename** (*str*) – optional Node ID of node used to limit the scan to a single node.

Returns a `list` of *Record Tuple*.

```
import aerospike
import pprint

pp = pprint.PrettyPrinter(indent=2)
config = { 'hosts': [ ('127.0.0.1',3000)]}
client = aerospike.client(config).connect()

client.put(('test','test','key1'), {'id':1,'a':1},
    policy={'key':aerospike.POLICY_KEY_SEND})
client.put(('test','test','key2'), {'id':2,'b':2},
    policy={'key':aerospike.POLICY_KEY_SEND})

scan = client.scan('test', 'test')
scan.select('id','a','zzz')
res = scan.results()
pp.pprint(res)
client.close()
```

Note: We expect to see:

```
[ ( ( 'test',
      'test',
      u'key2',
      bytearray(b'\xb2\x18\n\xd4\xce\xd8\xba:\x96s\xf5\x9ba\xf1j\xa7t\xeem\x01
↪')),
    { 'gen': 52, 'ttl': 2592000},
    { 'id': 2}),
  ( ( 'test',
      'test',
```

(continues on next page)

(continued from previous page)

```
    u'key1',
    bytearray(b'\x1cJ\xce\xa7\xd4Vj\xef+\xdf@W\xa5\xd8o\x8d:\xc9\xf4\xde')),
  { 'gen': 52, 'ttl': 2592000},
  { 'a': 1, 'id': 1})]
```

Note: Python client versions >= 3.10.0 Supports predicate expressions for results, foreach, and execute_background see *predexp*. Requires server version >= 4.7.0.

```
from __future__ import print_function
import aerospike
from aerospike import predexp
from aerospike import exception as ex
import sys
import time

config = { 'hosts': [('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

# register udf
try:
    client.udf_put('/path/to/my_udf.lua')
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    client.close()
    sys.exit(1)

# put records and run scan
try:
    keys = [('test', 'demo', 1), ('test', 'demo', 2), ('test', 'demo', 3)]
    records = [{'number': 1}, {'number': 2}, {'number': 3}]
    for i in range(3):
        client.put(keys[i], records[i])

    scan = client.scan('test', 'demo')

    preds = [ # check that the record has value < 2 or value == 3 in bin 'name'
        predexp.integer_bin('number'),
        predexp.integer_value(2),
        predexp.integer_less(),
        predexp.integer_bin('number'),
        predexp.integer_value(3),
        predexp.integer_equal(),
        predexp.predexp_or(2)
    ]

    policy = {
        'predexp': preds
    }

    records = scan.results(policy)
```

(continues on next page)

(continued from previous page)

```
    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
# the scan only returns records that match the predexp
# EXPECTED OUTPUT:
# [
#   (('test', 'demo', 1, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\
↪x9a\x1e\xac\xc4F\xc8')), {'gen': 2, 'ttl': 2591999}, {'number': 1}),
#   (('test', 'demo', 3, bytearray(b'\xb1\xa5`g\xf6\xd4\xa8\xa4D9\xd3\xafb\xbf\
↪xf8ha\x01\x94\xcd')), {'gen': 13, 'ttl': 2591999}, {'number': 3})
# ]
```

```
# contents of my_udf.lua
function my_udf(rec, bin, offset)
    info("my transform: %s", tostring(record.digest(rec)))
    rec[bin] = rec[bin] + offset
    aerospike:update(rec)
end
```

---

foreach(*callback*[, *policy*[, *options*[, *nodename*]]])

Invoke the *callback* function for each of the records streaming back from the scan.

Parameters

- **callback** (*callable*) – the function to invoke for each record.
- **policy** (*dict*) – optional *Scan Policies*.
- **options** (*dict*) – the *Scan Options* that will apply to the scan.
- **nodename** (*str*) – optional Node ID of node used to limit the scan to a single node.

Note: A *Record Tuple* is passed as the argument to the callback function.

```
import aerospike
import pprint

pp = pprint.PrettyPrinter(indent=2)
config = { 'hosts': [ ('127.0.0.1',3000)]}
client = aerospike.client(config).connect()

client.put(('test','test','key1'), {'id':1,'a':1},
    policy={'key':aerospike.POLICY_KEY_SEND})
client.put(('test','test','key2'), {'id':2,'b':2},
    policy={'key':aerospike.POLICY_KEY_SEND})

def show_key(record):
    key, meta, bins = record
    print(key)
```

(continues on next page)

(continued from previous page)

```
scan = client.scan('test', 'test')
scan_opts = {
  'concurrent': True,
  'nobins': True
}
scan.foreach(show_key, options=scan_opts)
client.close()
```

Note: We expect to see:

```
('test', 'test', u'key2', bytearray(b'\xb2\x18\n\xd4\xce\xd8\xba:\x96s\xf5\
↪x9ba\xf1j\xa7t\xeem\x01'))
('test', 'test', u'key1', bytearray(b'\x1cJ\xce\xa7\xd4Vj\xef+\xdf@W\xa5\xd8o\
↪x8d:\xc9\xf4\xde'))
```

Note: To stop the stream return `False` from the callback function.

```
from __future__ import print_function
import aerospike

config = { 'hosts': [ ('127.0.0.1',3000)]}
client = aerospike.client(config).connect()

def limit(lim, result):
    c = [0] # integers are immutable so a list (mutable) is used for the␣
↪counter
    def key_add(record):
        key, metadata, bins = record
        if c[0] < lim:
            result.append(key)
            c[0] = c[0] + 1
        else:
            return False
    return key_add

scan = client.scan('test','user')
keys = []
scan.foreach(limit(100, keys))
print(len(keys)) # this will be 100 if the number of matching records > 100
client.close()
```

execute_background([*policy*])

Execute a record UDF on records found by the scan in the background. This method returns before the scan has completed. A UDF can be added to the scan with *Scan.apply()*.

Parameters **policy** (*dict*) – optional *Write Policies*.

Returns a job ID that can be used with *aerospike.Client.job_info()* to track the status of the `aerospike.JOB_SCAN`, as it runs in the background.

Note: Python client version 3.10.0 implemented scan execute_background.

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
import sys
import time

config = {"hosts": [("127.0.0.1", 3000)]}
client = aerospike.client(config).connect()

# register udf
try:
    client.udf_put("/path/to/my_udf.lua")
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    client.close()
    sys.exit(1)

# put records and apply udf
try:
    keys = [("test", "demo", 1), ("test", "demo", 2), ("test", "demo", 3)]
    records = [{"number": 1}, {"number": 2}, {"number": 3}]
    for i in range(3):
        client.put(keys[i], records[i])

    scan = client.scan("test", "demo")
    scan.apply("my_udf", "my_udf", ["number", 10])
    job_id = scan.execute_background()

    # wait for job to finish
    while True:
        response = client.job_info(job_id, aerospike.JOB_SCAN)
        if response["status"] != aerospike.JOB_STATUS_INPROGRESS:
            break
        time.sleep(0.25)

    records = client.get_many(keys)
    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
# EXPECTED OUTPUT:
# [
#   (('test', 'demo', 1, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\
↪x9a\x1e\xac\xc4F\xc8')), {'gen': 2, 'ttl': 2591999}, {'number': 11}),
#   (('test', 'demo', 2, bytearray(b'\xaejQ_7\xdeJ\xda\xccD\x96\xe2\xda\x1f\xea\
↪x84\x8c:\x92p')), {'gen': 12, 'ttl': 2591999}, {'number': 12}),
#   (('test', 'demo', 3, bytearray(b'\xb1\xa5`g\xf6\xd4\xa8\xa4D9\xd3\xafb\xbf\
↪xf8ha\x01\x94\xcd')), {'gen': 13, 'ttl': 2591999}, {'number': 13})
# ]
```

```
# contents of my_udf.lua
function my_udf(rec, bin, offset)
    info("my transform: %s", tostring(record.digest(rec)))
    rec[bin] = rec[bin] + offset
    aerospike:update(rec)
end
```

1.3.1.2 Scan Policies

policy

A `dict` of optional scan policies which are applicable to *`Scan.results()`* and *`Scan.foreach()`*. See *Policy Options*.

aggedright

- **max_retries** `int`
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`.

  Default: `0`

  > **Warning:** Database writes that are not idempotent (such as "add") should not be retried because the write operation may be performed multiple times if the client timed out previous transaction attempts. It's important to use a distinct write policy for non-idempotent writes which sets max_retries = *0*;

- **sleep_between_retries** `int`
  Milliseconds to sleep between retries. Enter `0` to skip sleep.

  Default: `0`

- **socket_timeout** `int`
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not `0` and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both `socket_timeout` and `total_timeout` are non-zero and `socket_timeout` > `total_timeout`, then `socket_timeout` will be set to `total_timeout`. If `socket_timeout` is `0`, there will be no socket idle limit.

  Default: `30000`.

- **total_timeout** `int`
  Total transaction timeout in milliseconds.

  The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

  If `total_timeout` is not `0` and `total_timeout` is reached before the transaction completes, the transaction will return error `AEROSPIKE_ERR_TIMEOUT`. If `total_timeout` is `0`, there will be no total time limit.

  Default: `0`

- **compress** (`bool`)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

This option will increase cpu and memory usage (for extra compressed buffers), but decrease the size of data sent over the network.

Default: `False`

- **fail_on_cluster_change** `bool`
  Deprecated in 6.0.0. No longer has any effect..
  Abort the scan if the cluster is not in a stable state. Only used for server versions < 4.9.

  Default: `False`
- **durable_delete** `bool`
  Perform durable delete (requires Enterprise server version >= 3.10)
  If the transaction results in a record deletion, leave a tombstone for the record.

  Default: `False`
- **records_per_second** `int`
  Limit the scan to process records at records_per_second.

  Requires server version >= 4.7.0.

  Default: `0` (no limit).
- **expressions** `list`
  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: `None`

  Note: Requires Aerospike server version >= 5.2.
- **max_records** `int`
  Approximate number of records to return to client.
  This number is divided by the number of nodes involved in the scan.
  The actual number of records returned may be less than max_records if node record counts are small and unbalanced across nodes.

  Default: `0` (No Limit).

1.3.1.3 Scan Options

options
A `dict` of optional scan options which are applicable to *Scan.foreach()*.

aggedright

- **priority**
  Deprecated in 6.0.0. Scan priority will be removed in a coming release.
  Scan priority has been replaced by the records_per_second policy see *Scan Policies*.
- **nobins** `bool`
  Whether to return the *bins* portion of the *Record Tuple*.

  Default `False`.
- **concurrent** `bool`
  Whether to run the scan concurrently on all nodes of the cluster.

  Default `False`.
- **percent** `int`
  Deprecated in version 6.0.0, will be removed in a coming release.
  No longer available with server 5.6+.
  Use scan policy max_records instead.
  Percentage of records to return from the scan.

  Default `100`.

New in version 1.0.39.

1.4 Query Class — Query

1.4.1 Query

The query object created by calling *aerospike.Client.query()* is used for executing queries over a secondary index of a specified set (which can be omitted or None). For queries, the None set contains those records which are not part of any named set.

The query can (optionally) be assigned one of the following

- One of the *predicates* (*between()* or *equals()*) using *where()*.
- A list of *predexp* using *predexp()*

A query without a predicate will match all the records in the given set, similar to a *Scan*.

The query is invoked using *foreach()*, *results()*, or *execute_background()* The bins returned can be filtered by using `select()`.

If a list of write operations is added to the query with *add_ops()*, they will be applied to each record processed by the query. See available write operations at See *aerospike_helpers*

Finally, a stream UDF may be applied with *apply()*. It will aggregate results out of the records streaming back from the query.

See also:

Queries and Managing Queries.

1.4.1.1 Query Methods

class aerospike.**Query**

select(*bin1*[, *bin2*[, *bin3..*]])
Set a filter on the record bins resulting from *results()* or *foreach()*. If a selected bin does not exist in a record it will not appear in the *bins* portion of that record tuple.

where(*predicate*)
Set a where *predicate* for the query, without which the query will behave similar to *aerospike.Scan*. The predicate is produced by one of the *aerospike.predicates* methods *equals()* and *between()*.

Parameters **predicate** (*tuple*) – the tuple() produced by one of the *aerospike.predicates* methods.

Note: Currently, you can assign at most one predicate to the query.

results(*[,policy [, options]]) -> list of (key, meta, bins*)
Buffer the records resulting from the query, and return them as a `list` of records.

Parameters

- **policy** (*dict*) – optional *Query Policies*.
- **options** (*dict*) – optional *Query Options*.

Returns a `list` of *Record Tuple*.

```
import aerospike
from aerospike import predicates as p
import pprint

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

pp = pprint.PrettyPrinter(indent=2)
query = client.query('test', 'demo')
query.select('name', 'age') # matched records return with the values of these␣
↪bins
# assuming there is a secondary index on the 'age' bin of test.demo
query.where(p.equals('age', 40))
records = query.results( {'total_timeout':2000})
pp.pprint(records)
client.close()
```

Note: Queries require a secondary index to exist on the *bin* being queried.

Note: Python client version >= 3.10.0 Supports predicate expressions for results, foreach, and execute_background see *predexp*. Requires server versions >= 4.7.0.

```
from __future__ import print_function
import aerospike
from aerospike import predexp
from aerospike import exception as ex
import sys
import time

config = {"hosts": [("127.0.0.1", 3000)]}
client = aerospike.client(config).connect()

# register udf
try:
    client.udf_put(
        "/path/to/my_udf.lua"
    )
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    client.close()
    sys.exit(1)

# put records and apply udf
try:
    keys = [("test", "demo", 1), ("test", "demo", 2), ("test", "demo", 3)]
    records = [{"number": 1}, {"number": 2}, {"number": 3}]
    for i in range(3):
        client.put(keys[i], records[i])

    try:
```

(continues on next page)

(continued from previous page)

```
        client.index_integer_create("test", "demo", "number", "test_demo_number_idx
↪")
    except ex.IndexFoundError:
        pass

    query = client.query("test", "demo")
    query.apply("my_udf", "my_udf", ["number", 10])
    job_id = query.execute_background()

    # wait for job to finish
    while True:
        response = client.job_info(job_id, aerospike.JOB_SCAN)
        print(response)
        if response["status"] != aerospike.JOB_STATUS_INPROGRESS:
            break
        time.sleep(0.25)

    records = client.get_many(keys)
    print(records)
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()
# EXPECTED OUTPUT:
# [
#   (('test', 'demo', 1, bytearray(b'\xb7\xf4\xb88\x89\xe2\xdag\xdeh>\x1d\xf6\x91\x9a\
↪x1e\xac\xc4F\xc8')), {'gen': 2, 'ttl': 2591999}, {'number': 11}),
#   (('test', 'demo', 2, bytearray(b'\xaejQ_7\xdeJ\xda\xccD\x96\xe2\xda\x1f\xea\x84\
↪x8c:\x92p')), {'gen': 12, 'ttl': 2591999}, {'number': 12}),
#   (('test', 'demo', 3, bytearray(b'\xb1\xa5`g\xf6\xd4\xa8\xa4D9\xd3\xafb\xbf\xf8ha\
↪x01\x94\xcd')), {'gen': 13, 'ttl': 2591999}, {'number': 13})
# ]
```

```
# contents of my_udf.lua
function my_udf(rec, bin, offset)
    info("my transform: %s", tostring(record.digest(rec)))
    rec[bin] = rec[bin] + offset
    aerospike:update(rec)
end
```

Note: For a similar example using .results() see *aerospike.Scan.results()*.

---

foreach(*callback*[, *policy*[, *options*]])

Invoke the *callback* function for each of the records streaming back from the query.

Parameters

- **callback** (*callable*) – the function to invoke for each record.
- **policy** (*dict*) – optional *Query Policies*.

- **options** (*dict*) – optional *Query Options*.

Note: A *Record Tuple* is passed as the argument to the callback function.

```
import aerospike
from aerospike import predicates as p
import pprint

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

pp = pprint.PrettyPrinter(indent=2)
query = client.query('test', 'demo')
query.select('name', 'age') # matched records return with the values of these␣
↪bins
# assuming there is a secondary index on the 'age' bin of test.demo
query.where(p.between('age', 20, 30))
names = []
def matched_names(record):
    key, metadata, bins = record
    pp.pprint(bins)
    names.append(bins['name'])

query.foreach(matched_names, {'total_timeout':2000})
pp.pprint(names)
client.close()
```

Note: To stop the stream return `False` from the callback function.

```
from __future__ import print_function
import aerospike
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1',3000)]}
client = aerospike.client(config).connect()

def limit(lim, result):
    c = [0] # integers are immutable so a list (mutable) is used for the␣
↪counter
    def key_add(record):
        key, metadata, bins = record
        if c[0] < lim:
            result.append(key)
            c[0] = c[0] + 1
        else:
            return False
    return key_add

query = client.query('test','user')
query.where(p.between('age', 20, 30))
keys = []
```

(continues on next page)

(continued from previous page)

```
query.foreach(limit(100, keys))
print(len(keys)) # this will be 100 if the number of matching records > 100
client.close()
```

---

apply(*module*, *function*[, *arguments*])

Aggregate the *results()* using a stream UDF. If no predicate is attached to the *Query* the stream UDF will aggregate over all the records in the specified set.

Parameters

- **module** (*str*) – the name of the Lua module.
- **function** (*str*) – the name of the Lua function within the *module*.
- **arguments** (*list*) – optional arguments to pass to the *function*. NOTE: these arguments must be types supported by Aerospike See: supported data types. If you need to use an unsuported type, (e.g. set or tuple) you can use a serializer like pickle first.

Returns one of the supported types, `int`, `str`, `float` (double), `list`, `dict` (map), `bytearray` (bytes).

See also:

Developing Stream UDFs

Note: Assume we registered the following Lua module with the cluster as **stream_udf.lua** using *aerospike.Client.udf_put()*.

```
local function having_ge_threshold(bin_having, ge_threshold)
    return function(rec)
        debug("group_count::thresh_filter: %s >  %s ?", tostring(rec[bin_
↪having]), tostring(ge_threshold))
        if rec[bin_having] < ge_threshold then
            return false
        end
        return true
    end
end

local function count(group_by_bin)
  return function(group, rec)
    if rec[group_by_bin] then
      local bin_name = rec[group_by_bin]
      group[bin_name] = (group[bin_name] or 0) + 1
      debug("group_count::count: bin %s has value %s which has the count of %s
↪", tostring(bin_name), tostring(group[bin_name]))
    end
    return group
  end
end

local function add_values(val1, val2)
  return val1 + val2
end
```

(continues on next page)

(continued from previous page)

```
local function reduce_groups(a, b)
  return map.merge(a, b, add_values)
end

function group_count(stream, group_by_bin, bin_having, ge_threshold)
  if bin_having and ge_threshold then
    local myfilter = having_ge_threshold(bin_having, ge_threshold)
    return stream : filter(myfilter) : aggregate(map{}, count(group_by_bin)) :␣
↪reduce(reduce_groups)
  else
    return stream : aggregate(map{}, count(group_by_bin)) : reduce(reduce_
↪groups)
  end
end
```

Find the first name distribution of users in their twenties using a query aggregation:

```
import aerospike
from aerospike import predicates as p
import pprint

config = {'hosts': [('127.0.0.1', 3000)],
          'lua': {'system_path':'/usr/local/aerospike/lua/',
                  'user_path':'/usr/local/aerospike/usr-lua/'}}
client = aerospike.client(config).connect()

pp = pprint.PrettyPrinter(indent=2)
query = client.query('test', 'users')
query.where(p.between('age', 20, 29))
query.apply('stream_udf', 'group_count', [ 'first_name' ])
names = query.results()
# we expect a dict (map) whose keys are names, each with a count value
pp.pprint(names)
client.close()
```

With stream UDFs, the final reduce steps (which ties the results from the reducers of the cluster nodes) executes on the client-side. Explicitly setting the Lua `user_path` in the config helps the client find the local copy of the module containing the stream UDF. The `system_path` is constructed when the Python package is installed, and contains system modules such as `aerospike.lua`, `as.lua`, and `stream_ops.lua`. The default value for the Lua `system_path` is `/usr/local/aerospike/lua`.

---

add_ops(*ops*)

Add a list of write ops to the query. When used with *Query.execute_background()* the query will perform the write ops on any records found. If no predicate is attached to the Query it will apply ops to all the records in the specified set.

Parameters **ops** – *list* A list of write operations generated by the aerospike_helpers e.g. list_operations, map_operations, etc.

Note: Requires server version >= 4.7.0.

```
import aerospike
from aerospike_helpers.operations import list_operations
from aerospike_helpers.operations import operations
query = client.query('test', 'demo')

ops = [
    operations.append(test_bin, 'val_to_append'),
    list_operations.list_remove_by_index(test_bin, list_index_to_remove,␣
↪aerospike.LIST_RETURN_NONE)
]
query.add_ops(ops)

id = query.execute_background()
client.close()
```

For a more comprehensive example, see using a list of write ops with *Query.execute_background()* .

predexp(*predicates*)

Set the predicate expression filters to be used by this query.

Parameters **predicates** – *list* A list of predicates generated by the *aerospike.predexp — Query Predicate Expressions* functions

```
import aerospike
from aerospike import predexp as predexp
query = client.query('test', 'demo')

predexps = [
    predexp.rec_device_size(),
    predexp.integer_value(65 * 1024),
    predexp.integer_greater()
]
query.predexp(predexps)

big_records = query.results()
client.close()
```

execute_background([*policy*])

Execute a record UDF or write operations on records found by the query in the background. This method returns before the query has completed. A UDF or a list of write operations must have been added to the query with *Query.apply()* or *Query.add_ops()* respectively.

Parameters **policy** (*dict*) – optional *Write Policies*.

Returns a job ID that can be used with *aerospike.Client.job_info()* to track the status of the aerospike.JOB_QUERY , as it runs in the background.

```
# Using a record UDF
import aerospike
query = client.query('test', 'demo')
query.apply('myudfs', 'myfunction', ['a', 1])
query_id = query.execute_background()
# This id can be used to monitor the progress of the background query
```

```
# Using a list of write ops.
from __future__ import print_function
import aerospike
from aerospike import predicates
from aerospike import exception as ex
from aerospike_helpers.operations import list_operations
import sys
import time

# Configure the client.
config = {"hosts": [("127.0.0.1", 3000)]}

# Create a client and connect it to the cluster.
try:
    client = aerospike.client(config).connect()
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

TEST_NS = "test"
TEST_SET = "demo"
nested_list = [{"name": "John", "id": 100}, {"name": "Bill", "id": 200}]
# Write the records.
try:
    keys = [(TEST_NS, TEST_SET, i) for i in range(5)]
    for i, key in enumerate(keys):
        client.put(key, {"account_number": i, "members": nested_list})
except ex.RecordError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))

# EXAMPLE 1: Append a new account member to all accounts.
try:
    new_member = {"name": "Cindy", "id": 300}

    ops = [list_operations.list_append("members", new_member)]

    query = client.query(TEST_NS, TEST_SET)
    query.add_ops(ops)
    id = query.execute_background()
    # allow for query to complete
    time.sleep(3)
    print("EXAMPLE 1")

    for i, key in enumerate(keys):
        _, _, bins = client.get(key)
        print(bins)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# EXAMPLE 2: Remove a member from a specific account using predicates.
try:
    # Add index to the records for use with predex.
```

(continues on next page)

(continued from previous page)

```
    client.index_integer_create(
        TEST_NS, TEST_SET, "account_number", "test_demo_account_number_idx"
    )

    ops = [
        list_operations.list_remove_by_index("members", 0, aerospike.LIST_
↪RETURN_NONE)
    ]

    query = client.query(TEST_NS, TEST_SET)
    number_predicate = predicates.equals("account_number", 3)
    query.where(number_predicate)
    query.add_ops(ops)
    id = query.execute_background()
    # allow for query to complete
    time.sleep(3)
    print("EXAMPLE 2")

    for i, key in enumerate(keys):
        _, _, bins = client.get(key)
        print(bins)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# Cleanup and close the connection to the Aerospike cluster.
for i, key in enumerate(keys):
    client.remove(key)
client.index_remove(TEST_NS, "test_demo_account_number_idx")
client.close()

"""
EXPECTED OUTPUT:
EXAMPLE 1
{'account_number': 0, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
{'account_number': 1, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
{'account_number': 2, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
{'account_number': 3, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
{'account_number': 4, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
EXAMPLE 2
{'account_number': 0, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
{'account_number': 1, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
{'account_number': 2, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
{'account_number': 3, 'members': [{'name': 'Bill', 'id': 200}, {'name': 'Cindy', 'id':␣
↪300}]}
```

(continues on next page)

(continued from previous page)

```
{'account_number': 4, 'members': [{'name': 'John', 'id': 100}, {'name': 'Bill', 'id': 200}
↪, {'name': 'Cindy', 'id': 300}]}
"""
```

1.4.1.2 Query Policies

policy

A dict of optional query policies which are applicable to *Query.results()* and *Query.foreach()*. See *Policy Options*.

aggedright

- **max_retries** int
  Maximum number of retries before aborting the current transaction. The initial attempt is not counted as a retry.

  If max_retries is exceeded, the transaction will return error AEROSPIKE_ERR_TIMEOUT.

  Default: 0

  Warning: : Database writes that are not idempotent (such as "add") should not be retried because the write operation may be performed multiple times if the client timed out previous transaction attempts. It's important to use a distinct write policy for non-idempotent writes which sets max_retries = *0*;

- **sleep_between_retries** int
  Milliseconds to sleep between retries. Enter 0 to skip sleep.

  Default: 0
- **socket_timeout** int
  Socket idle timeout in milliseconds when processing a database command.

  If socket_timeout is not 0 and the socket has been idle for at least socket_timeout, both max_retries and total_timeout are checked. If max_retries and total_timeout are not exceeded, the transaction is retried.

  If both socket_timeout and total_timeout are non-zero and socket_timeout > total_timeout, then socket_timeout will be set to total_timeout. If socket_timeout is 0, there will be no socket idle limit.

  Default: 30000.
- **total_timeout** int
  Total transaction timeout in milliseconds.

  The total_timeout is tracked on the client and sent to the server along with the transaction in the wire protocol. The client will most likely timeout first, but the server also has the capability to timeout the transaction.

  If total_timeout is not 0 total_timeout is reached before the transaction completes, the transaction will return error AEROSPIKE_ERR_TIMEOUT. If total_timeout is 0, there will be no total time limit.

  Default: 0
- **compress** (bool)
  Compress client requests and server responses.

  Use zlib compression on write or batch read commands when the command buffer size is greater than 128 bytes. In addition, tell the server to compress it's response on read commands. The server response compression threshold is also 128 bytes.

  This option will increase cpu and memory usage (for extra compressed buffers), but decrease the size of data sent over the network.

Default: False

- **deserialize** bool
  Should raw bytes representing a list or map be deserialized to a list or dictionary.
  Set to *False* for backup programs that just need access to raw bytes.

  Default: True
- **fail_on_cluster_change** bool
  Deprecated in 6.0.0. No longer has any effect..
  Terminate query if cluster is in migration state.

  Default False
- **expressions** list
  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

Note: Requires Aerospike server version >= 5.2.

1.4.1.3 Query Options

options
A dict of optional query options which are applicable to *Query.foreach()* and *Query.results()*.

aggedright

- **nobins** bool
  Whether to return the *bins* portion of the *Record Tuple*.

  Default False.

New in version 3.0.0.

1.5 aerospike.predicates — Query Predicates

aerospike.predicates.**between**(*bin*, *min*, *max*)
Represent a *bin* **BETWEEN** *min* **AND** *max* predicate.

Parameters

- **bin** (*str*) – the bin name.
- **min** (*int*) – the minimum value to be matched with the between operator.
- **max** (*int*) – the maximum value to be matched with the between operator.

Returns tuple() to be used in *aerospike.Query.where()*.

```
from __future__ import print_function
import aerospike
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()
query = client.query('test', 'demo')
query.where(p.between('age', 20, 30))
res = query.results()
print(res)
client.close()
```

aerospike.predicates.**equals**(*bin*, *val*)
Represent a *bin* = *val* predicate.

Parameters

- **bin** (*str*) – the bin name.
- **val** (str or int) – the value to be matched with an equals operator.

Returns tuple() to be used in *aerospike.Query.where()*.

```
from __future__ import print_function
import aerospike
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()
query = client.query('test', 'demo')
query.where(p.equals('name', 'that guy'))
res = query.results()
print(res)
client.close()
```

aerospike.predicates.**geo_within_geojson_region**(*bin*, *shape*[, *index_type*])
Predicate for finding any point in bin which is within the given shape. Requires a geo2dsphere index (*index_geo2dsphere_create()*) over a *bin* containing *GeoJSON* point data.

Parameters

- **bin** (*str*) – the bin name.
- **shape** (*str*) – the shape formatted as a GeoJSON string.
- **index_type** – Optional. Possible aerospike.INDEX_TYPE_* values are detailed in *Miscellaneous*.

Returns tuple() to be used in *aerospike.Query.where()*.

Note: Requires server version >= 3.7.0

```
from __future__ import print_function
import aerospike
from aerospike import GeoJSON
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

client.index_geo2dsphere_create('test', 'pads', 'loc', 'pads_loc_geo')
bins = {'pad_id': 1,
        'loc': aerospike.geojson('{"type":"Point", "coordinates":[-80.604333, 28.
↪608389]}')}
client.put(('test', 'pads', 'launchpad1'), bins)

# Create a search rectangle which matches screen boundaries:
# (from the bottom left corner counter-clockwise)
scrn = GeoJSON({ 'type': "Polygon",
```

(continues on next page)

(continued from previous page)

```
                'coordinates': [
                 [[-80.590000, 28.60000],
                  [-80.590000, 28.61800],
                  [-80.620000, 28.61800],
                  [-80.620000, 28.60000],
                  [-80.590000, 28.60000]]]})

# Find all points contained in the rectangle.
query = client.query('test', 'pads')
query.select('pad_id', 'loc')
query.where(p.geo_within_geojson_region('loc', scrn.dumps()))
records = query.results()
print(records)
client.close()
```

New in version 1.0.58.

aerospike.predicates.**geo_within_radius**(*bin*, *long*, *lat*, *radius_meters*[, *index_type*])

Predicate helper builds an AeroCircle GeoJSON shape, and returns a 'within GeoJSON region' predicate. Requires a geo2dsphere index (*index_geo2dsphere_create()*) over a *bin* containing *GeoJSON* point data.

Parameters

- **bin** (*str*) – the bin name.
- **long** (*float*) – the longitude of the center point of the AeroCircle.
- **lat** (*float*) – the latitude of the center point of the AeroCircle.
- **radius_meters** (*float*) – the radius length in meters of the AeroCircle.
- **index_type** – Optional. Possible `aerospike.INDEX_TYPE_*` values are detailed in *Miscellaneous*.

Returns `tuple()` to be used in *aerospike.Query.where()*.

Note: Requires server version >= 3.8.1

```
from __future__ import print_function
import aerospike
from aerospike import GeoJSON
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

client.index_geo2dsphere_create('test', 'pads', 'loc', 'pads_loc_geo')
bins = {'pad_id': 1,
        'loc': aerospike.geojson('{"type":"Point", "coordinates":[-80.604333, 28.
↪608389]}')}
client.put(('test', 'pads', 'launchpad1'), bins)

query = client.query('test', 'pads')
query.select('pad_id', 'loc')
query.where(p.geo_within_radius('loc', -80.605000, 28.60900, 400.0))
```

(continues on next page)

(continued from previous page)

```
records = query.results()
print(records)
client.close()
```

New in version 1.0.58.

aerospike.predicates.**geo_contains_geojson_point**(*bin*, *point*[, *index_type*])

Predicate for finding any regions in the bin which contain the given point. Requires a geo2dsphere index (*index_geo2dsphere_create()*) over a *bin* containing *GeoJSON* point data.

Parameters

- **bin** (*str*) – the bin name.
- **point** (*str*) – the point formatted as a GeoJSON string.
- **index_type** – Optional. Possible `aerospike.INDEX_TYPE_*` values are detailed in *Miscellaneous*.

Returns `tuple()` to be used in *aerospike.Query.where()*.

Note: Requires server version >= 3.7.0

```
from __future__ import print_function
import aerospike
from aerospike import GeoJSON
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

client.index_geo2dsphere_create('test', 'launch_centers', 'area', 'launch_area_geo')
rect = GeoJSON({ 'type': "Polygon",
                 'coordinates': [
                  [[-80.590000, 28.60000],
                   [-80.590000, 28.61800],
                   [-80.620000, 28.61800],
                   [-80.620000, 28.60000],
                   [-80.590000, 28.60000]]]})
bins = {'area': rect}
client.put(('test', 'launch_centers', 'kennedy space center'), bins)

# Find all geo regions containing a point
point = GeoJSON({'type': "Point",
                 'coordinates': [-80.604333, 28.608389]})
query = client.query('test', 'launch_centers')
query.where(p.geo_contains_geojson_point('area', point.dumps()))
records = query.results()
print(records)
client.close()
```

New in version 1.0.58.

aerospike.predicates.**geo_contains_point**(*bin*, *long*, *lat*[, *index_type*])

Predicate helper builds a GeoJSON point, and returns a 'contains GeoJSON point' predicate. Requires a

geo2dsphere index (*index_geo2dsphere_create()*) over a *bin* containing *GeoJSON* point data.

Parameters

- **bin** (*str*) – the bin name.
- **long** (*float*) – the longitude of the point.
- **lat** (*float*) – the latitude of the point.
- **index_type** – Optional. Possible `aerospike.INDEX_TYPE_*` values are detailed in *Miscellaneous*.

Returns `tuple()` to be used in *aerospike.Query.where()*.

Note: Requires server version >= 3.7.0

```
from __future__ import print_function
import aerospike
from aerospike import GeoJSON
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

client.index_geo2dsphere_create('test', 'launch_centers', 'area', 'launch_area_geo')
rect = GeoJSON({ 'type': "Polygon",
                 'coordinates': [
                  [[-80.590000, 28.60000],
                   [-80.590000, 28.61800],
                   [-80.620000, 28.61800],
                   [-80.620000, 28.60000],
                   [-80.590000, 28.60000]]]})
bins = {'area': rect}
client.put(('test', 'launch_centers', 'kennedy space center'), bins)

# Find all geo regions containing a point
query = client.query('test', 'launch_centers')
query.where(p.geo_contains_point('area', -80.604333, 28.608389))
records = query.results()
print(records)
client.close()
```

New in version 1.0.58.

`aerospike.predicates.`**`contains`**(*bin*, *index_type*, *val*)

Represent the predicate *bin* **CONTAINS** *val* for a bin with a complex (list or map) type.

Parameters

- **bin** (*str*) – the bin name.
- **index_type** – Possible `aerospike.INDEX_TYPE_*` values are detailed in *Miscellaneous*.
- **val** (`str` or `int`) – match records whose *bin* is an *index_type* (ex: list) containing *val*.

Returns `tuple()` to be used in *aerospike.Query.where()*.

Note: Requires server version >= 3.8.1

```
from __future__ import print_function
import aerospike
from aerospike import predicates as p

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

# assume the bin fav_movies in the set test.demo bin should contain
# a dict { (str) _title_ : (int) _times_viewed_ }
# create a secondary index for string values of test.demo records whose 'fav_movies'␣
↪bin is a map
client.index_map_keys_create('test', 'demo', 'fav_movies', aerospike.INDEX_STRING,
↪'demo_fav_movies_titles_idx')
# create a secondary index for integer values of test.demo records whose 'fav_movies'␣
↪bin is a map
client.index_map_values_create('test', 'demo', 'fav_movies', aerospike.INDEX_
↪NUMERIC, 'demo_fav_movies_views_idx')

client.put(('test','demo','Dr. Doom'), {'age':43, 'fav_movies': {'12 Monkeys': 1,
↪'Brasil': 2}})
client.put(('test','demo','The Hulk'), {'age':38, 'fav_movies': {'Blindness': 1,
↪'Eternal Sunshine': 2}})

query = client.query('test', 'demo')
query.where(p.contains('fav_movies', aerospike.INDEX_TYPE_MAPKEYS, '12 Monkeys'))
res = query.results()
print(res)
client.close()
```

aerospike.predicates.**range**(*bin*, *index_type*, *min*, *max*)

Represent the predicate *bin* **CONTAINS** values **BETWEEN** *min* **AND** *max* for a bin with a complex (list or map) type.

Parameters

- **bin** (*str*) – the bin name.
- **index_type** – Possible `aerospike.INDEX_TYPE_*` values are detailed in *Miscellaneous*.
- **min** (*int*) – the minimum value to be used for matching with the range operator.
- **max** (*int*) – the maximum value to be used for matching with the range operator.

Returns `tuple()` to be used in *aerospike.Query.where()*.

Note: Requires server version >= 3.8.1

```
from __future__ import print_function
import aerospike
from aerospike import predicates as p
```

(continues on next page)

(continued from previous page)

```
config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()

# create a secondary index for numeric values of test.demo records whose 'age' bin is␣
↪a list
client.index_list_create('test', 'demo', 'age', aerospike.INDEX_NUMERIC, 'demo_age_
↪nidx')

# query for records whose 'age' bin has a list with numeric values between 20 and 30
query = client.query('test', 'demo')
query.where(p.range('age', aerospike.INDEX_TYPE_LIST, 20, 30))
res = query.results()
print(res)
client.close()
```

1.6 aerospike.predexp — Query Predicate Expressions

The following methods allow a user to define a predicate expression filter. Predicate expression filters are applied on the query results on the server. Predicate expression filters may occur on any bin in the record.

aerospike.predexp.**predexp_and**(*nexpr*)

Create an AND logical predicate expression.

Parameters **nexpr** – int Number of expressions to combine with "and" . The value of nexpr must be between 1 and 65535.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "c" is between 11 and 20 inclusive:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("c"),
    predexp.integer_value(11),
    predexp.integer_greatereq(),
    predexp.integer_bin("c"),
    predexp.integer_value(20),
    predexp.integer_lesseq(),
    predexp.predexp_and(2)
]
```

aerospike.predexp.**predexp_or**(*nexpr*)

Create an Or logical predicate expression.

Parameters **nexpr** – int Number of expressions to combine with "or". The value of nexpr must be between 1 and 65535.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "pet" is "dog" or "cat"

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_bin("pet"),
    predexp.string_value("cat"),
    predexp.string_equal(),
    predexp.string_bin("pet"),
    predexp.string_value("dog"),
    predexp.string_equal(),
    predexp.predexp_or(2)
]
```

aerospike.predexp.**predexp_not**()

Create a not logical predicate expression which negates the previouse predicate expression on the stack.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "pet" is not "cat"

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_bin("pet"),
    predexp.string_value("cat"),
    predexp.string_equal(),
    predexp.predexp_not()
]
```

aerospike.predexp.**integer_bin**(*bin_name*)

Create an integer bin value predicate expression.

Parameters **bin_name** – str The name of the bin containing an integer.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "age" is 42

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("age"),
    predexp.integer_value(42),
    predexp.integer_equal()
]
```

aerospike.predexp.**string_bin**(*bin_name*)

Create a string bin value predicate expression.

Parameters **bin_name** – str The name of the bin containing a string.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "name" is "Bob".

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_bin("name"),
    predexp.string_value("Bob"),
```

(continues on next page)

(continued from previous page)

```
    predexp.string_equal()
]
```

aerospike.predexp.**geojson_bin**(*bin_name*)

Create a GeoJSON bin value predicate expression.

Parameters **bin_name** – str The name of the bin containing a GeoJSON value.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "location" is within a specified region.

```
from aerospike import predexp as predexp
geo_region = aerospike.GeoJSON(
    {"type": "AeroCircle", "coordinates": [[-122.0, 37.5], 1000]}).dumps()
predexps =  [
    predexp.geojson_bin("location"),
    predexp.geojson_value(geo_region),
    predexp.geojson_within()
]
```

aerospike.predexp.**list_bin**(*bin_name*)

Create a list bin value predicate expression.

Parameters **bin_name** – str The name of the bin containing a list.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the list in bin "names" contains an entry equal to "Alice"

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_var("list_entry"),
    predexp.string_value("Alice"),
    predexp.string_equal(),
    predexp.list_bin("names"),
    predexp.list_iterate_or("list_entry")
]
```

aerospike.predexp.**map_bin**(*bin_name*)

Create a map bin value predicate expression.

Parameters **bin_name** – str The name of the bin containing a map value.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the map in bin "pet_count" has an entry with a key equal to "Cat"

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_var("map_key"),
    predexp.string_value("Cat"),
    predexp.string_equal(),
    predexp.map_bin("pet_count"),
```

(continues on next page)

(continued from previous page)

```
    predexp.mapkey_iterate_or("map_key")
]
```

aerospike.predexp.**geojson_value**(*geo_value*)
Create a GeoJSON value predicate expression.

Parameters **bin_name** – str The geojson string.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "location" is within a specified region.

```
from aerospike import predexp as predexp
geo_region = aerospike.GeoJSON(
    {"type": "AeroCircle", "coordinates": [[-122.0, 37.5], 1000]}).dumps()
predexps =  [
    predexp.geojson_bin("location"),
    predexp.geojson_value(geo_region),
    predexp.geojson_within()
]
```

aerospike.predexp.**integer_value**(*int_value*)
Create an integer value predicate expression.

Parameters **bin_name** – int The integer value

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "age" is 42

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("age"),
    predexp.integer_value(42),
    predexp.integer_equal()
]
```

aerospike.predexp.**string_value**(*string_value*)
Create a string value predicate expression.

Parameters **bin_name** – str The string value.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records where the value of bin "name" is "Bob".

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_bin("name"),
    predexp.string_value("Bob"),
    predexp.string_equal()
]
```

aerospike.predexp.**integer_var**(*var_name*)
Create an integer iteration variable predicate expression.

Parameters **var_name** – str The name of the variable. This should match a value used when specifying the iteration.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example the following selects a record where the list in bin "numbers" contains an entry equal to 42

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_var("item"),
    predexp.integer_value(42),
    predexp.integer_equal(),
    predexp.list_bin("numbers"),
    predexp.list_iterate_or("item")
]
```

aerospike.predexp.**string_var**(*var_name*)

Create an string iteration variable predicate expression.

Parameters **var_name** – str The name of the variable. This should match a value used when specifying the iteration.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example the following selects a record where the list in bin "languages" contains an entry equal to "Python"

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_var("item"),
    predexp.string_value("Python"),
    predexp.string_equal(),
    predexp.list_bin("languages"),
    predexp.list_iterate_or("item")
]
```

aerospike.predexp.**geojson_var**(*var_name*)

Create an GeoJSON iteration variable predicate expression.

Parameters **var_name** – str The name of the variable. This should match a value used when specifying the iteration.

Returns tuple() to be used in *aerospike.Query.predexp()*.

aerospike.predexp.**list_iterate_or**(*var_name*)

Create an list iteration OR logical predicate expression.

Parameters **bin_name** – str The name of the iteration variable

Returns tuple() to be used in *aerospike.Query.predexp()*.

The list iteration expression pops two children off the expression stack. The left child (pushed earlier) must contain a logical subexpression containing one or more matching iteration variable expressions. The right child (pushed later) must specify a list bin. The list iteration traverses the list and repeatedly evaluates the subexpression substituting each list element's value into the matching iteration variable. The result of the iteration expression is a logical OR of all of the individual element evaluations.

If the list bin contains zero elements *list_iterate_or()* will evaluate to false.

For example, the following sequence of predicate expressions selects records where the list in bin "names" contains an entry equal to "Alice"

```
from aerospike import predexp as predexp
predexps = [
    predexp.string_var("list_entry"),
    predexp.string_value("Alice"),
    predexp.string_equal(),
    predexp.list_bin("names"),
    predexp.list_iterate_or("list_entry")
]
```

aerospike.predexp.**list_iterate_and**(*var_name*)

Create an list iteration And logical predicate expression.

Parameters **var_name** – str The name of the iteration variable

Returns tuple() to be used in *aerospike.Query.predexp()*.

The list iteration expression pops two children off the expression stack. The left child (pushed earlier) must contain a logical subexpression containing one or more matching iteration variable expressions. The right child (pushed later) must specify a list bin. The list iteration traverses the list and repeatedly evaluates the subexpression substituting each list element's value into the matching iteration variable. The result of the iteration expression is a logical AND of all of the individual element evaluations.

If the list bin contains zero elements *list_iterate_and()* will evaluate to true. This is useful when testing for exclusion (see example).

For example, the following sequence of predicate expressions selects records where the list in bin "names" contains no entries equal to "Bob".

```
from aerospike import predexp as predexp
predexps = [
    predexp.string_var("list_entry"),
    predexp.string_value("Bob"),
    predexp.string_equal(),
    predexp.predexp_not(),
    predexp.list_bin("names"),
    predexp.list_iterate_and("list_entry")
]
```

aerospike.predexp.**mapkey_iterate_or**(*var_name*)

Create an map key iteration Or logical predicate expression.

Parameters **var_name** – str The name of the iteration variable

Returns tuple() to be used in *aerospike.Query.predexp()*.

The mapkey iteration expression pops two children off the expression stack. The left child (pushed earlier) must contain a logical subexpression containing one or more matching iteration variable expressions. The right child (pushed later) must specify a map bin. The mapkey iteration traverses the map and repeatedly evaluates the subexpression substituting each map key value into The matching iteration variable. The result of the iteration expression is a logical OR of all of the individual element evaluations.

If the map bin contains zero elements *mapkey_iterate_or()* will return false. For example, the following sequence of predicate expressions selects records where the map in bin "pet_count" has an entry with a key equal to "Cat"

```
from aerospike import predexp as predexp
predexps = [
```

(continues on next page)

(continued from previous page)

```
    predexp.string_var("map_key"),
    predexp.string_value("Cat"),
    predexp.string_equal(),
    predexp.map_bin("pet_count"),
    predexp.mapkey_iterate_or("map_key")
]
```

aerospike.predexp.**mapkey_iterate_and**(*var_name*)

Create an map key iteration AND logical predicate expression.

Parameters **var_name** – str The name of the iteration variable

Returns tuple() to be used in *aerospike.Query.predexp()*.

The mapkey iteration expression pops two children off the expression stack. The left child (pushed earlier) must contain a logical subexpression containing one or more matching iteration variable expressions. The right child (pushed later) must specify a map bin. The mapkey iteration traverses the map and repeatedly evaluates the subexpression substituting each map key value into The matching iteration variable. The result of the iteration expression is a logical AND of all of the individual element evaluations.

If the map bin contains zero elements *mapkey_iterate_and()* will return true. This is useful when testing for exclusion (see example).

For example, the following sequence of predicate expressions selects records where the map in bin "pet_count" does not contain an entry with a key equal to "Cat".

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_var("map_key"),
    predexp.string_value("Cat"),
    predexp.string_equal(),
    predexp.predexp_not(),
    predexp.map_bin("pet_count"),
    predexp.mapkey_iterate_and("map_key")
]
```

aerospike.predexp.**mapval_iterate_or**(*var_name*)

Create an map value iteration Or logical predicate expression.

Parameters **var_name** – str The name of the iteration variable

Returns tuple() to be used in *aerospike.Query.predexp()*.

The mapval iteration expression pops two children off the expression stack. The left child (pushed earlier) must contain a logical subexpression containing one or more matching iteration variable expressions. The right child (pushed later) must specify a map bin. The mapval iteration traverses the map and repeatedly evaluates the subexpression substituting each map value into the matching iteration variable. The result of the iteration expression is a logical OR of all of the individual element evaluations.

If the map bin contains zero elements *mapval_iterate_or()* will return false.

For example, the following sequence of predicate expressions selects records where at least one of the values in the map in bin "pet_count" is 0

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_var("map_key"),
```

(continues on next page)

(continued from previous page)

```
    predexp.integer_value(0),
    predexp.integer_equal(),
    predexp.map_bin("pet_count"),
    predexp.mapval_iterate_or("map_key")
]
```

aerospike.predexp.**mapval_iterate_and**(*var_name*)

Create an map value iteration AND logical predicate expression.

Parameters **var_name** – str The name of the iteration variable

Returns tuple() to be used in *aerospike.Query.predexp()*.

The mapval iteration expression pops two children off the expression stack. The left child (pushed earlier) must contain a logical subexpression containing one or more matching iteration variable expressions. The right child (pushed later) must specify a map bin. The mapval iteration traverses the map and repeatedly evaluates the subexpression substituting each map value into the matching iteration variable. The result of the iteration expression is a logical AND of all of the individual element evaluations.

If the map bin contains zero elements *mapval_iterate_and()* will return true. This is useful when testing for exclusion (see example).

For example, the following sequence of predicate expressions selects records where none of the values in the map in bin "pet_count" is 0

```
from aerospike import predexp as predexp
predexps =  [
    predexp.string_var("map_key"),
    predexp.integer_value(0),
    predexp.integer_equal(),
    predexp.predexp_not(),
    predexp.map_bin("pet_count"),
    predexp.mapval_iterate_and("map_key")
]
```

aerospike.predexp.**rec_digest_modulo**(*mod*)

Create a digest modulo record metadata value predicate expression.

Parameters **mod** – int The value of this expression assumes the value of 4 bytes of the digest modulo this argument.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have `digest(key) % 3 == 1` :

```
from aerospike import predexp as predexp
predexps =  [
    predexp.rec_digest_modulo(3),
    predexp.integer_value(1),
    predexp.integer_equal()
]
```

aerospike.predexp.**rec_last_update**()

Create a last update record metadata value predicate expression. The record last update expression assumes the value of the number of nanoseconds since the unix epoch that the record was last updated.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have been updated after a timestamp:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.rec_last_update(),
    predexp.integer_value(timestamp_ns),
    predexp.integer_greater()
]
```

aerospike.predexp.**rec_void_time**()

Create a void time record metadata value predicate expression. The record void time expression assumes the value of the number of nanoseconds since the unix epoch when the record will expire. The special value of 0 means the record will not expire.

Returns `tuple()` to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have void time set to 0 (no expiration):

```
from aerospike import predexp as predexp
predexps =  [
    predexp.rec_void_time(),
    predexp.integer_value(0),
    predexp.integer_equal()
]
```

aerospike.predexp.**rec_device_size**()

Create a record device size metadata value predicate expression. The record device size expression assumes the value of the size in bytes that the record occupies on device storage. For non-persisted records, this value is 0.

Returns `tuple()` to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records whose device storage size is larger than 65K:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.rec_device_size(),
    predexp.integer_value(65 * 1024),
    predexp.integer_greater()
]
```

aerospike.predexp.**integer_equal**()

Create an integer comparison logical predicate expression. If the value of either of the child expressions is unknown because a specified bin does not exist or contains a value of the wrong type the result of the comparison is false. If a true outcome is desirable in this situation use the complimentary comparison and enclose in a logical NOT.

Returns `tuple()` to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" equal to 42:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("foo"),
    predexp.integer_value(42),
```

(continues on next page)

(continued from previous page)

```
    predexp.integer_equal()
]
```

aerospike.predexp.**integer_greater**()
Create an integer comparison logical predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" greater than 42:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("foo"),
    predexp.integer_value(42),
    predexp.integer_greater()
]
```

aerospike.predexp.**integer_greatereq**()
Create an integer comparison logical predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" greater than or equal to 42:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("foo"),
    predexp.integer_value(42),
    predexp.integer_greatereq()
]
```

aerospike.predexp.**integer_less**()
Create an integer comparison logical predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" less than 42:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("foo"),
    predexp.integer_value(42),
    predexp.integer_less()
]
```

aerospike.predexp.**integer_lesseq**()
Create an integer comparison logical predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" less than or equal to 42:

```
from aerospike import predexp as predexp
predexps =  [
    predexp.integer_bin("foo"),
```

(continues on next page)

(continued from previous page)

```
    predexp.integer_value(42),
    predexp.integer_lesseq()
]
```

aerospike.predexp.**integer_unequal**()

Create an integer comparison logical predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

This expression will evaluate to true if, and only if, both children of the expression exist, and are of type integer, and are not equal to each other. If this is not desired, utilize *aerospike.predexp.integer_equal()* in conjunction with *aerospike.predexp.predexp_not()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" not equal to 42:

```
from aerospike import predexp as predexp
predexps = [
    predexp.integer_bin("foo"),
    predexp.integer_value(42),
    predexp.integer_unequal()
]
```

aerospike.predexp.**string_equal**()

Create an integer comparison logical predicate expression. If the value of either of the child expressions is unknown because a specified bin does not exist or contains a value of the wrong type the result of the comparison is false. If a true outcome is desirable in this situation use the complimentary comparison and enclose in a logical NOT.

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" equal to "bar":

```
from aerospike import predexp as predexp
predexps = [
    predexp.string_bin("foo"),
    predexp.string_value("bar"),
    predexp.string_equal()
]
```

aerospike.predexp.**string_unequal**()

Create an integer comparison logical predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

This expression will evaluate to true if, and only if, both children of the expression exist, and are of type string, and are not equal to each other. If this is not desired, utilize *aerospike.predexp.string_equal()* in conjunction with *aerospike.predexp.predexp_not()*.

For example, the following sequence of predicate expressions selects records that have bin "foo" not equal to "bar":

```
from aerospike import predexp as predexp
predexps = [
    predexp.string_bin("foo"),
    predexp.string_value("bar"),
    predexp.string_unequal()
]
```

aerospike.predexp.**geojson_within**()
Create a Geojson within predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

```
from aerospike import predexp as predexp
predexps =  [
    predexp.geojson_bin("location"),
    predexp.geojson_value(my_geo_region),
    predexp.geojson_within()
]
```

aerospike.predexp.**geojson_contains**()
Create a Geojson contains predicate expression.

Returns tuple() to be used in *aerospike.Query.predexp()*.

```
from aerospike import predexp as predexp
predexps =  [
    predexp.geojson_bin("region"),
    predexp.geojson_value(my_geo_point),
    predexp.geojson_contains()
]
```

aerospike.predexp.**string_regex**(**flags*)
Create a string regex predicate. May be called without any arguments to specify default behavior.

Parameters **flags** – int *Regex Flag Values* Any, or none of the aerospike REGEX constants

Returns tuple() to be used in *aerospike.Query.predexp()*.

For example, the following sequence of predicate expressions selects records that have bin "hex" value ending in '1' or '2':

```
from aerospike import predexp as predexp
predexps = [
    predexp.string_bin('hex'),
    predexp.string_value('0x00.[12]'),
    predexp.string_regex(aerospike.REGEX_ICASE)
]
```

1.7 aerospike.exception — Aerospike Exceptions

```
from __future__ import print_function

import aerospike
from aerospike import exception as ex

try:
    config = { 'hosts': [ ('127.0.0.1', 3000)], 'policies': { 'total_timeout': 1200}}
    client = aerospike.client(config).connect()
    client.close()
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
```

New in version 1.0.44.

1.7.1 In Doubt Status

The in doubt status of a caught exception can be checked by looking at the 5th element of its *args* tuple

```
key = 'test', 'demo', 1
record = {'some': 'thing'}
try:
  client.put(key, record)
except AerospikeError as exc:
  print("The in doubt nature of the operation is: {}".format(exc.args[4])
```

New in version 3.0.1.

1.7.2 Exception Types

exception aerospike.exception.**AerospikeError**
The parent class of all exceptions raised by the Aerospike client, inherits from exceptions.Exception . *These attributes should be checked by executing `exc.args[i]* where i is the index of the attribute. For example to check *in_doubt*, run *exc.args[4]*

code
The associated status code.

msg
The human-readable error message.

file

line

in_doubt
True if it is possible that the operation succeeded.

exception aerospike.exception.**ClientError**
Exception class for client-side errors, often due to mis-configuration or misuse of the API methods. Subclass of *AerospikeError*.

exception aerospike.exception.**InvalidHostError**
Subclass of *ClientError*.

exception aerospike.exception.**ParamError**
The operation was not performed because of invalid parameters.

exception aerospike.exception.**ServerError**
The parent class for all errors returned from the cluster.

exception aerospike.exception.**InvalidRequest**
Protocol-level error. Subclass of *ServerError*.

exception aerospike.exception.**OpNotApplicable**
The operation cannot be applied to the current bin value on the server. Subclass of *ServerError*.

exception aerospike.exception.**FilteredOut**
The transaction was not performed because the predexp was false.

exception aerospike.exception.**ServerFull**
The server node is running out of memory and/or storage device space reserved for the specified namespace. Subclass of *ServerError*.

exception aerospike.exception.**AlwaysForbidden**
Operation not allowed in current configuration. Subclass of *ServerError*.

exception aerospike.exception.**UnsupportedFeature**
Encountered an unimplemented server feature. Subclass of *ServerError*.

exception aerospike.exception.**DeviceOverload**
The server node's storage device(s) can't keep up with the write load. Subclass of *ServerError*.

exception aerospike.exception.**NamespaceNotFound**
Namespace in request not found on server. Subclass of *ServerError*.

exception aerospike.exception.**ForbiddenError**
Operation not allowed at this time. Subclass of *ServerError*.

exception aerospike.exception.**ElementExistsError**
Raised when trying to alter a map key which already exists, when using a create_only policy.

Subclass of *ServerError*.

exception aerospike.exception.**ElementNotFoundError**
Raised when trying to alter a map key which does not exist, when using an update_only policy.

Subclass of *ServerError*.

exception aerospike.exception.**RecordError**
The parent class for record and bin exceptions exceptions associated with read and write operations. Subclass of *ServerError*.

key
The key identifying the record.

bin
Optionally the bin associated with the error.

exception aerospike.exception.**RecordKeyMismatch**
Record key sent with transaction did not match key stored on server. Subclass of *RecordError*.

exception aerospike.exception.**RecordNotFound**
Record does not exist in database. May be returned by read, or write with policy *aerospike.POLICY_EXISTS_UPDATE*. Subclass of *RecordError*.

exception aerospike.exception.**RecordGenerationError**
Generation of record in database does not satisfy write policy. Subclass of *RecordError*.

exception aerospike.exception.**RecordExistsError**
Record already exists. May be returned by write with policy *aerospike.POLICY_EXISTS_CREATE*. Subclass of *RecordError*.

exception aerospike.exception.**RecordBusy**
Too may concurrent requests for one record - a "hot-key" situation. Subclass of *RecordError*.

exception aerospike.exception.**RecordTooBig**
Record being (re-)written can't fit in a storage write block. Subclass of *RecordError*.

exception aerospike.exception.**BinNameError**
Length of bin name exceeds the limit of 14 characters. Subclass of *RecordError*.

exception aerospike.exception.**BinIncompatibleType**
Bin modification operation can't be done on an existing bin due to its value type (for example appending to an integer). Subclass of *RecordError*.

exception aerospike.exception.**IndexError**
The parent class for indexing exceptions. Subclass of *ServerError*.

index_name
The name of the index associated with the error.

exception aerospike.exception.**IndexNotFound**
Subclass of *IndexError*.

exception aerospike.exception.**IndexFoundError**
Subclass of *IndexError*.

exception aerospike.exception.**IndexOOM**
The index is out of memory. Subclass of *IndexError*.

exception aerospike.exception.**IndexNotReadable**
Subclass of *IndexError*.

exception aerospike.exception.**IndexNameMaxLen**
Subclass of *IndexError*.

exception aerospike.exception.**IndexNameMaxCount**
Reached the maximum allowed number of indexes. Subclass of *IndexError*.

exception aerospike.exception.**QueryError**
Exception class for query errors. Subclass of *AerospikeError*.

exception aerospike.exception.**QueryQueueFull**
Subclass of *QueryError*.

exception aerospike.exception.**QueryTimeout**
Subclass of *QueryError*.

exception aerospike.exception.**ClusterError**
Cluster discovery and connection errors. Subclass of *AerospikeError*.

exception aerospike.exception.**ClusterChangeError**
A cluster state change occurred during the request. This may also be returned by scan operations with the fail-on-cluster-change flag set. Subclass of *ClusterError*.

exception aerospike.exception.**AdminError**
The parent class for exceptions of the security API.

exception aerospike.exception.**ExpiredPassword**
Subclass of *AdminError*.

exception aerospike.exception.**ForbiddenPassword**
Subclass of *AdminError*.

exception aerospike.exception.**IllegalState**
Subclass of *AdminError*.

exception aerospike.exception.**InvalidCommand**
Subclass of *AdminError*.

exception aerospike.exception.**InvalidCredential**
Subclass of *AdminError*.

exception aerospike.exception.**InvalidField**
Subclass of *AdminError*.

exception aerospike.exception.**InvalidPassword**
Subclass of *AdminError*.

exception aerospike.exception.**InvalidPrivilege**
Subclass of *AdminError*.

exception aerospike.exception.**InvalidRole**
Subclass of *AdminError*.

exception aerospike.exception.**InvalidUser**
Subclass of *AdminError*.

exception aerospike.exception.**NotAuthenticated**
Subclass of *AdminError*.

exception aerospike.exception.**RoleExistsError**
Subclass of *AdminError*.

exception aerospike.exception.**RoleViolation**
Subclass of *AdminError*.

exception aerospike.exception.**SecurityNotEnabled**
Subclass of *AdminError*.

exception aerospike.exception.**SecurityNotSupported**
Subclass of *AdminError*.

exception aerospike.exception.**SecuritySchemeNotSupported**
Subclass of *AdminError*.

exception aerospike.exception.**UserExistsError**
Subclass of *AdminError*.

exception aerospike.exception.**UDFError**
The parent class for UDF exceptions exceptions. Subclass of *ServerError*.

module
The UDF module associated with the error.

func
Optionally the name of the UDF function.

exception aerospike.exception.**UDFNotFound**
Subclass of *UDFError*.

exception aerospike.exception.**LuaFileNotFound**
Subclass of *UDFError*.

1.7.3 Exception Hierarchy

```
AerospikeError (*)
 +-- TimeoutError (9)
 +-- ClientError (-1)
 |    +-- InvalidHost (-4)
 |    +-- ParamError (-2)
 +-- ServerError (1)
      +-- InvalidRequest (4)
      +-- ServerFull (8)
      +-- AlwaysForbidden (10)
```

(continues on next page)

(continued from previous page)

```
      +-- UnsupportedFeature (16)
      +-- DeviceOverload (18)
      +-- NamespaceNotFound (20)
      +-- ForbiddenError (22)
      +-- ElementNotFoundError (23)
      +-- ElementExistsError (24)
      +-- RecordError (*)
      |    +-- RecordKeyMismatch (19)
      |    +-- RecordNotFound (2)
      |    +-- RecordGenerationError (3)
      |    +-- RecordExistsError (5)
      |    +-- RecordTooBig (13)
      |    +-- RecordBusy (14)
      |    +-- BinNameError (21)
      |    +-- BinIncompatibleType (12)
      +-- IndexError (204)
      |    +-- IndexNotFound (201)
      |    +-- IndexFoundError (200)
      |    +-- IndexOOM (202)
      |    +-- IndexNotReadable (203)
      |    +-- IndexNameMaxLen (205)
      |    +-- IndexNameMaxCount (206)
      +-- QueryError (213)
      |    +-- QueryQueueFull (211)
      |    +-- QueryTimeout (212)
      +-- ClusterError (11)
      |    +-- ClusterChangeError (7)
      +-- AdminError (*)
      |    +-- SecurityNotSupported (51)
      |    +-- SecurityNotEnabled (52)
      |    +-- SecuritySchemeNotSupported (53)
      |    +-- InvalidCommand (54)
      |    +-- InvalidField (55)
      |    +-- IllegalState (56)
      |    +-- InvalidUser (60)
      |    +-- UserExistsError (61)
      |    +-- InvalidPassword (62)
      |    +-- ExpiredPassword (63)
      |    +-- ForbiddenPassword (64)
      |    +-- InvalidCredential (65)
      |    +-- InvalidRole (70)
      |    +-- RoleExistsError (71)
      |    +-- RoleViolation (81)
      |    +-- InvalidPrivilege (72)
      |    +-- NotAuthenticated (80)
      +-- UDFError (*)
           +-- UDFNotFound (1301)
           +-- LuaFileNotFound (1302)
```

1.8 aerospike_helpers — Aerospike Helper Package for bin operations (list, map, bit, etc.)

This package contains helpers to be used by the operate and operate_ordered methods for bin operations. (list, map, bitwise, etc.)

1.8.1 Subpackages

1.8.1.1 aerospike_helpers.operations package

aerospike_helpers.operations.operations module

Module with helper functions to create dictionaries consumed by the *aerospike.Client.operate* and *aerospike.Client.operate_ordered* methods for the aerospike.client class.

aerospike_helpers.operations.operations.**append**(*bin_name*, *append_item*)
Create an append operation dictionary.

The append operation appends *append_item* to the value in bin_name.

Parameters

- **bin** (*string*) – The name of the bin to be used.
- **append_item** – The value which will be appended to the item contained in the specified bin.

Returns A dictionary to be passed to operate or operate_ordered.

aerospike_helpers.operations.operations.**delete**()
Create a delete operation dictionary.

The delete operation deletes a record and all associated bins. Requires server version >= 4.7.0.8.

Returns A dictionary to be passed to operate or operate_ordered.

aerospike_helpers.operations.operations.**increment**(*bin_name*, *amount*)
Create an increment operation dictionary.

The increment operation increases a value in bin_name by the specified amount, or creates a bin with the value of amount.

Parameters

- **bin** (*string*) – The name of the bin to be incremented.
- **amount** – The amount by which to increment the item in the specified bin.

Returns A dictionary to be passed to operate or operate_ordered.

aerospike_helpers.operations.operations.**prepend**(*bin_name*, *prepend_item*)
Create a prepend operation dictionary.

The prepend operation prepends *prepend_item* to the value in bin_name.

Parameters

- **bin** (*string*) – The name of the bin to be used.
- **prepend_item** – The value which will be prepended to the item contained in the specified bin.

Returns A dictionary to be passed to operate or operate_ordered.

aerospike_helpers.operations.operations.**read**(*bin_name*)
Create a read operation dictionary.

The read operation reads and returns the value in *bin_name*.

Parameters **bin** – String the name of the bin from which to read.

Returns A dictionary to be passed to operate or operate_ordered.

aerospike_helpers.operations.operations.**touch**(*ttl=None*)
Create a touch operation dictionary.

Using ttl here is deprecated. It should be set in the record metadata for the operate method.

Parameters **ttl** (*int*) – Deprecated. The ttl that should be set for the record. This should be set in the metadata passed to the operate or operate_ordered methods.

Returns A dictionary to be passed to operate or operate_ordered.

aerospike_helpers.operations.operations.**write**(*bin_name*, *write_item*)
Create a write operation dictionary.

The write operation writes *write_item* into the bin specified by bin_name.

Parameters

- **bin** (*string*) – The name of the bin into which *write_item* will be stored.
- **write_item** – The value which will be written into the bin.

Returns A dictionary to be passed to operate or operate_ordered.

aerospike_helpers.operations.list_operations module

This module provides helper functions to produce dictionaries to be used with the *aerospike.Client.operate* and *aerospike.Client.operate_ordered* methods of the aerospike module.

List operations support nested CDTs through an optional ctx context argument. The ctx argument is a list of cdt_ctx objects. See *aerospike_helpers.cdt_ctx*.

Note: Nested CDT (ctx) requires server version >= 4.6.0

---

aerospike_helpers.operations.list_operations.**list_append**(*bin_name*, *value*, *policy=None*, *ctx=None*)
Creates a list append operation to be used with operate, or operate_ordered

The list append operation instructs the aerospike server to append an item to the end of a list bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **value** – The value to be appended to the end of the list.
- **policy** (*dict*) – An optional dictionary of *list write options*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_append_items**(*bin_name*, *values*, *policy=None*, *ctx=None*)

Creates a list append items operation to be used with operate, or operate_ordered

The list append items operation instructs the aerospike server to append multiple items to the end of a list bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **values** – (list): A sequence of items to be appended to the end of the list.
- **policy** (*dict*) – An optional dictionary of *list write options*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_clear**(*bin_name*, *ctx=None*)

Create list clear operation.

The list clear operation removes all items from the list specified by *bin_name*

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to be cleared
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get**(*bin_name*, *index*, *ctx=None*)

Create a list get operation.

The list get operation gets the value of the item at *index* and returns the value

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **index** (*int*) – The index of the item to be returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_index**(*bin_name*, *index*, *return_type*, *ctx=None*)

Create a list get index operation.

The list get operation gets the item at *index* and returns a value specified by *return_type*

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **index** (*int*) – The index of the item to be returned.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values

- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_index_range**(*bin_name*, *index*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a list get index range operation.

The list get by index range operation gets *count* items starting at *index* and returns a value specified by *return_type*

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **index** (*int*) – The index of the first item to be returned.
- **count** (*int*) – The number of list items to be selected.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **inverted** (*bool*) – Optional bool specifying whether to invert the return type. If set to *True*, all items outside of the specified range will be returned. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_rank**(*bin_name*, *rank*, *return_type*, *ctx=None*)

Create a list get by rank operation.

Server selects list item identified by *rank* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch a value from.
- **rank** (*int*) – The rank of the item to be fetched.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_rank_range**(*bin_name*, *rank*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a list get by rank range operation.

Server selects *count* items starting at the specified *rank* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.

- **rank** (*int*) – The rank of the first items to be returned.
- **count** (*int*) – A positive number indicating number of items to be returned.
- **return_type** (*int*) – Value specifying what should be returned from the operation.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the return type. If set to *True*, all items outside of the specified rank range will be returned. Default: *False*

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_value**(*bin_name*, *value*, *return_type*, *inverted=False*, *ctx=None*)

Create a list get by value operation.

Server selects list items with a value equal to *value* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **value** – The server returns all items matching this value
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the return type. If set to *True*, all items not equal to *value* will be selected. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_value_list**(*bin_name*, *value_list*, *return_type*, *inverted=False*, *ctx=None*)

Create a list get by value list operation.

Server selects list items with a value contained in *value_list* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **value_list** (*list*) – Return items from the list matching an item in this list.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the return type. If set to *True*, all items not matching an entry in *value_list* will be selected. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_value_range**(*bin_name*, *return_type*, *value_begin*, *value_end*, *inverted=False*, *ctx=None*)

Create a list get by value list operation.

Server selects list items with a value greater than or equal to *value_begin* and less than *value_end*. If *value_begin* is *None*, range is greater than or equal to the first element of the list. If *value_end* is *None* range extends to the end of the list. Server returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **value_begin** – The start of the value range.
- **value_end** – The end of the value range.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the return type. If set to *True*, all items not included in the specified range will be returned. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_by_value_rank_range_relative**(*bin_name*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a list get by value rank range relative operation

Create list get by value relative to rank range operation. Server selects list items nearest to value and greater by relative rank. Server returns selected data specified by return_type.

Note: This operation requires server version 4.3.0 or greater.

Examples

These examples show what would be returned for specific arguments when dealing with an ordered list: `[0,4,5,9,11,15]`

```
(value, offset, count) = [selected items]
(5, 0, None) = [5,9,11,15]
(5, 0, 2) = [5, 9]
(5, -1, None) = [4, 5, 9, 11, 15]
(5, -1, 3) = [4, 5, 9]
(3,3, None) = [11, 15]
```

(continues on next page)

(continued from previous page)

```
(3,-3, None) = [0, 4,5,9,11,15]
(3, 0, None) = [4,5,9,11,15]
```

Parameters

- **bin_name** (*str*) – The name of the bin containing the list.
- **value** (*str*) – The value of the item in the list for which to search
- **offset** (*int*) – Begin returning items with rank == rank(found_item) + offset
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **count** (*int*) – If specified, the number of items to return. If None, all items until end of list are returned.
- **inverted** (*bool*) – If True, the operation is inverted, and items outside of the specified range are returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_get_range**(*bin_name*, *index*, *count*, *ctx=None*)

Create a list get range operation.

The list get range operation gets *count* items starting *index* and returns the values.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **index** (*int*) – The index of the item to be returned.
- **count** (*int*) – A positive number of items to be returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_increment**(*bin_name*, *index*, *value*, *policy=None*, *ctx=None*)

Creates a list increment operation to be used with operate, or operate_ordered

The list insert operation inserts an item at index: *index* into the list contained in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index** (*int*) – The index of the list item to increment.
- **value** (*int, float*) – The value to be added to the list item.
- **policy** (*dict*) – An optional dictionary of *list write options*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_insert**(*bin_name*, *index*, *value*, *policy=None*, *ctx=None*)

Creates a list insert operation to be used with operate, or operate_ordered

The list insert operation inserts an item at index: *index* into the list contained in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index** (*int*) – The index at which to insert an item. The value may be positive to use zero based indexing or negative to index from the end of the list.
- **value** – The value to be inserted into the list.
- **policy** (*dict*) – An optional dictionary of *list write options*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_insert_items**(*bin_name*, *index*, *values*, *policy=None*, *ctx=None*)

Creates a list insert items operation to be used with operate, or operate_ordered

The list insert items operation inserts items at index: *index* into the list contained in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index** (*int*) – The index at which to insert the items. The value may be positive to use zero based indexing or negative to index from the end of the list.
- **values** (*list*) – The values to be inserted into the list.
- **policy** (*dict*) – An optional dictionary of *list write options*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_pop**(*bin_name*, *index*, *ctx=None*)

Creates a list pop operation to be used with operate, or operate_ordered

The list pop operation removes and returns an item index: *index* from list contained in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index** (*int*) – The index of the item to be removed.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_pop_range**(*bin_name*, *index*, *count*, *ctx=None*)
Creates a list pop range operation to be used with operate, or operate_ordered

The list insert range operation removes and returns *count* items starting from index: *index* from the list contained in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index** (*int*) – The index of the first item to be removed.
- **count** (*int*) – A positive number indicating how many items, including the first,
- **be removed and returned** (*to*) –
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove**(*bin_name*, *index*, *ctx=None*)
Create list remove operation.

The list remove operation removes an item located at *index* in the list specified by *bin_name*

Parameters

- **bin_name** (*str*) – The name of the bin containing the item to be removed.
- **index** (*int*) – The index at which to remove the item.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_index**(*bin_name*, *index*, *return_type*, *ctx=None*)
Create a list remove by index operation.

The list_remove_by_index operation removes the value of the item at *index* and returns a value specified by *return_type*

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to remove an item from.
- **index** (*int*) – The index of the item to be removed.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_index_range**(*bin_name*, *index*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a list remove by index range operation.

The list remove by index range operation removes *count* starting at *index* and returns a value specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to remove items from.
- **index** (*int*) – The index of the first item to be removed.
- **count** (*int*) – The number of items to be removed
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **inverted** (*bool*) – Optional bool specifying whether to invert the operation. If set to *True*, all items outside of the specified range will be removed. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_rank**(*bin_name*, *rank*, *return_type*, *ctx=None*)

Create a list remove by rank operation.

Server removes a list item identified by *rank* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch a value from.
- **rank** (*int*) – The rank of the item to be removed.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_rank_range**(*bin_name*, *rank*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a list remove by rank range operation.

Server removes *count* items starting at the specified *rank* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **rank** (*int*) – The rank of the first item to removed.

- **count** (*int*) – A positive number indicating number of items to be removed.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the operation. If set to *True*, all items outside of the specified rank range will be removed. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_value**(*bin_name*, *value*, *return_type*, *inverted=False*, *ctx=None*)

Create a list remove by value operation.

Server removes list items with a value equal to *value* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to remove items from.
- **value** – The server removes all list items matching this value.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the operation. If set to *True*, all items not equal to *value* will be removed. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_value_list**(*bin_name*, *value_list*, *return_type*, *inverted=False*, *ctx=None*)

Create a list remove by value list operation.

Server removes list items with a value matching one contained in *value_list* and returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to remove items from.
- **value_list** (*list*) – The server removes all list items matching one of these values.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the operation. If set to *True*, all items not equal to a value contained in *value_list* will be removed. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_value_range**(*bin_name*, *return_type*, *value_begin=None*, *value_end=None*, *inverted=False*, *ctx=None*)

Create a list remove by value range operation.

Server removes list items with a value greater than or equal to *value_begin* and less than *value_end*. If *value_begin* is *None*, range is greater than or equal to the first element of the list. If *value_end* is *None* range extends to the end of the list. Server returns selected data specified by *return_type*.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to fetch items from.
- **value_begin** – The start of the value range.
- **value_end** – The end of the value range.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **inverted** (*bool*) – Optional bool specifying whether to invert the operation. If set to *True*, all items not included in the specified range will be removed. Default: *False*
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_by_value_rank_range_relative**(*bin_name*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a list get by value rank range relative operation

Create list remove by value relative to rank range operation. Server removes and returns list items nearest to value and greater by relative rank. Server returns selected data specified by return_type.

Note: This operation requires server version 4.3.0 or greater.

These examples show what would be removed and returned for specific arguments when dealing with an ordered list: `[0,4,5,9,11,15]`

```
(value, offset, count) = [selected items]
(5,0,None) = [5,9,11,15]
(5,0,2) = [5, 9]
(5,-1, None) = [4,5,9,11,15]
(5, -1, 3) = [4,5,9]
(3,3, None) = [11,15]
```

(continues on next page)

(continued from previous page)

```
(3,-3, None) = [0,4,5,9,11,15]
(3, 0, None) = [4,5,9,11,15]
```

Parameters

- **bin_name** (*str*) – The name of the bin containing the list.
- **value** (*str*) – The value of the item in the list for which to search
- **offset** (*int*) – Begin removing and returning items with rank == rank(found_item) + offset
- **count** (*int*) – If specified, the number of items to remove and return. If None, all items until end of list are returned.
- **inverted** (*bool*) – If True, the operation is inverted, and items outside of the specified range are removed and returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_remove_range**(*bin_name*, *index*, *count*, *ctx=None*)

Create list remove range operation.

The list remove range operation removes *count* items starting at *index* in the list specified by *bin_name*

Parameters

- **bin_name** (*str*) – The name of the bin containing the items to be removed.
- **index** (*int*) – The index of the first item to remove.
- **count** (*int*) – A positive number representing the number of items to be removed.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_set**(*bin_name*, *index*, *value*, *policy=None*, *ctx=None*)

Create a list set operation.

The list set operations sets the value of the item at *index* to *value*

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to be operated on.
- **index** (*int*) – The index of the item to be set.
- **value** – The value to be assigned to the list item.
- **policy** (*dict*) – An optional dictionary of *list write options*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_set_order**(*bin_name*, *list_order*, *ctx=None*)
Create a list set order operation.

The list_set_order operation sets an order on a specified list bin.

Parameters

- **bin_name** (*str*) – The name of the list bin.
- **list_order** – The ordering to apply to the list. Should be aerospike.LIST_ORDERED or aerospike.LIST_UNORDERED .
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_size**(*bin_name*, *ctx=None*)
Create a list size operation.

Server returns the size of the list in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_sort**(*bin_name*, *sort_flags=<MagicMock id='140444825472464'>*, *ctx=None*)
Create a list sort operation

The list sort operation will sort the specified list bin.

Parameters

- **bin_name** (*str*) – The name of the bin to sort.
- **sort_flags** – Optional flags modifiying the behavior of list_sort. This should be constructed by bitwise or'ing together values from *List Sort Flags*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.list_operations.**list_trim**(*bin_name*, *index*, *count*, *ctx=None*)
Create a list trim operation.

Server removes items in list bin that do not fall into range specified by index and count range.

Parameters

- **bin_name** (*str*) – The name of the bin containing the list to be trimmed.
- **index** (*int*) – The index of the items to be kept.
- **count** (*int*) – A positive number of items to be kept.

- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations module

Helper functions to create map operation dictionaries arguments for. the *aerospike.Client.operate* and *aerospike.Client.operate_ordered* methods of the aerospike client.

Map operations support nested CDTs through an optional ctx context argument. The ctx argument is a list of cdt_ctx objects. See *aerospike_helpers.cdt_ctx*.

Note: Nested CDT (ctx) requires server version >= 4.6.0

---

aerospike_helpers.operations.map_operations.**map_clear**(*bin_name*, *ctx=None*)

Creates a map_clear operation to be used with operate or operate_ordered

The operation removes all items from the map stored in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_decrement**(*bin_name*, *key*, *amount*, *map_policy=None*, *ctx=None*)

Creates a map_decrement operation to be used with operate or operate_ordered

The operation allows a user to decrement the value of a value stored in the map on the server.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key** – The key for the value to be decremented.
- **amount** – The amount by which to decrement the value stored in map[key]
- **map_policy** (*list, optional*) – Optional *map_policy dictionary* dictates the type of map to create when it does not exist. The map policy also specifies the mode used when writing items to the map. Defaults to *None*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_index**(*bin_name*, *index*, *return_type*, *ctx=None*)

Creates a map_get_by_index operation to be used with operate or operate_ordered

The operation returns the entry at index from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **index** (*int*) – The index of the entry to return.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_index_range**(*bin_name*, *index_start*, *get_amt*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_get_by_index_range operation to be used with operate or operate_ordered

The operation returns get_amt entries starting at index_start from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **index_start** (*int*) – The index of the first entry to return.
- **get_amt** (*int*) – The number of entries to return from the map.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If true, entries in the specified index range should be ignored, and all other entries returned. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_key**(*bin_name*, *key*, *return_type*, *ctx=None*)

Creates a map_get_by_key operation to be used with operate or operate_ordered

The operation returns an item, specified by the key from the map stored in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key** – The key of the item to be returned from the map
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_key_index_range_relative**(*bin_name*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a map get by value rank range relative operation

Create map get by key relative to index range operation. Server removes and returns map items with key nearest to value and greater by relative index. Server returns selected data specified by return_type.

Note: This operation requires server version 4.3.0 or greater.

Examples

Examples for a key ordered map `{0: 6, 6: 12, 10: 18, 15: 24}` and return type of `aerospike.MAP_RETURN_KEY`

```
(value, offset, count) = [returned keys]
(5, 0, None) = [6, 10, 15]
(5, 0, 2) = [6, 10]
(5,-1, None) = [0, 6, 10, 15]
(5, -1, 3) = [0, 6, 10]
(3, 2, None) = [15]
(3, 5, None) = []
```

Parameters

- **bin_name** (*str*) – The name of the bin containing the list.
- **value** (*str*) – The value of the item in the list for which to search
- **offset** (*int*) – Begin removing and returning items with rank == rank(fount_item) + offset
- **count** (*int*) – If specified, the number of items to remove and return. If None, all items until end of list are returned.
- **inverted** (*bool*) – If True, the operation is inverted and items outside of the specified range are returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_key_list**(*bin_name*, *key_list*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_get_by_key_list operation to be used with operate or operate_ordered

The operation returns items, specified by the keys in key_list from the map stored in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key_list** (*list*) – A list of keys to be returned from the map.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If true, keys with values not specified in the key_list will be returned, and those keys specified in the key_list will be ignored. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_key_range**(*bin_name*, *key_range_start*, *key_range_end*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_get_by_key_range operation to be used with operate or operate_ordered

The operation returns items with keys between key_range_start(inclusive) and key_range_end(exclusive) from the map

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key_range_start** – The start of the range of keys to be returned. (Inclusive)
- **key_range_end** – The end of the range of keys to be returned. (Exclusive)
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, values outside of the specified range will be returned, and values inside of the range will be ignored. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_rank**(*bin_name*, *rank*, *return_type*, *ctx=None*)

Creates a map_get_by_rank operation to be used with operate or operate_ordered

The operation returns the item with the specified rank from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **rank** (*int*) – The rank of the entry to return.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_rank_range**(*bin_name*, *rank_start*, *get_amt*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_get_by_rank_range operation to be used with operate or operate_ordered

The operation returns item within the specified rank range from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **rank_start** (*int*) – The start of the rank of the entries to return.
- **get_amt** (*int*) – The number of entries to return.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, items with ranks inside the specified range should be ignored, and all other entries returned. Default: False.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_value**(*bin_name*, *value*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_get_by_value operation to be used with operate or operate_ordered

The operation returns entries whose value matches the specified value.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value** – Entries with a value matching this argument will be returned from the map.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, entries with a value different than the specified value will be returned. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_value_list**(*bin_name*, *key_list*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_get_by_value_list operation to be used with operate or operate_ordered

The operation returns entries whose values are specified in the value_list.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value_list** (*list*) – Entries with a value contained in this list will be returned from the map.

- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, entries with a value contained in value_list will be ignored, and all others will be returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_value_range**(*bin_name*, *value_start*, *value_end*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_get_by_value_range operation to be used with operate or operate_ordered

The operation returns items, with values between value_start(inclusive) and value_end(exclusive) from the map

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value_start** – The start of the range of values to be returned. (Inclusive)
- **value_end** – The end of the range of values to be returned. (Exclusive)
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, values outside of the specified range will be returned, and values inside of the range will be ignored. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_get_by_value_rank_range_relative**(*bin_name*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a map remove by value rank range relative operation

Create list map get by value relative to rank range operation. Server returns map items with value nearest to value and greater by relative rank. Server returns selected data specified by return_type.

Note: This operation requires server version 4.3.0 or greater.

Examples

Examples for map `{0: 6, 10: 18, 6: 12, 15: 24}` and return type of `aerospike.MAP_RETURN_KEY`

```
(value, offset, count) = [returned keys]
(6, 0, None) = [0, 6, 10, 15]
(5, 0, 2) = [0, 6]
(7, -1, 1) = [0]
(7, -1, 3) = [0, 6, 10]
```

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value** (*str*) – The value of the item in the list for which to search
- **offset** (*int*) – Begin removing and returning items with rank == rank(fount_item) + offset
- **count** (*int*) – If specified, the number of items to remove and return. If None, all items until end of list are returned.
- **inverted** (*bool*) – If True, the operation is inverted and items outside of the specified range are returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_increment**(*bin_name*, *key*, *amount*, *map_policy=None*, *ctx=None*)

Creates a map_increment operation to be used with operate or operate_ordered

The operation allows a user to increment the value of a value stored in the map on the server.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key** – The key for the value to be incremented.
- **amount** – The amount by which to increment the value stored in map[key]
- **map_policy** (*list, optional*) – Optional *map_policy dictionary* dictates the type of map to create when it does not exist. The map policy also specifies the mode used when writing items to the map. Defaults to *None*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_put**(*bin_name*, *key*, *value*, *map_policy=None*, *ctx=None*)

Creates a map_put operation to be used with operate or operate_ordered

The operation allows a user to set the value of an item in the map stored on the server.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key** – The key for the map.
- **value** – The item to store in the map with the corresponding key.
- **map_policy** (*list, optional*) – Optional *map_policy dictionary* dictates the type of map to create when it does not exist. The map policy also specifies the mode used when writing items to the map. Defaults to *None*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_put_items**(*bin_name*, *item_dict*, *map_policy=None*, *ctx=None*)

Creates a map_put_items operation to be used with operate or operate_ordered

The operation allows a user to add or update items in the map stored on the server.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **item_dict** (*dict*) – A dictionary of key value pairs to be added to the map on the server.
- **map_policy** (*list, optional*) – Optional *map_policy dictionary* dictates the type of map to create when it does not exist. The map policy also specifies the mode used when writing items to the map. Defaults to *None*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_index**(*bin_name*, *index*, *return_type*, *ctx=None*)

Creates a map_remove_by_index operation to be used with operate or operate_ordered

The operation removes the entry at index from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **index** (*int*) – The index of the entry to remove.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_index_range**(*bin_name*, *index_start*, *remove_amt*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_remove_by_index_range operation to be used with operate or operate_ordered

The operation removes remove_amt entries starting at index_start from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **index_start** (*int*) – The index of the first entry to remove.
- **remove_amt** (*int*) – The number of entries to remove from the map.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If true, entries in the specified index range should be kept, and all other entries removed. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_key**(*bin_name*, *key*, *return_type*, *ctx=None*)

Creates a map_remove_by_key operation to be used with operate or operate_ordered

The operation removes an item, specified by the key from the map stored in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key** – The key to be removed from the map
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_key_index_range_relative**(*bin_name*, *key*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a map get by value rank range relative operation

Create map remove by key relative to index range operation. Server removes and returns map items with key nearest to value and greater by relative index. Server returns selected data specified by return_type.

Note: This operation requires server version 4.3.0 or greater.

Examples

Examples for a key ordered map `{0: 6, 6: 12, 10: 18, 15: 24}` and return type of `aerospike.MAP_RETURN_KEY`

```
(value, offset, count) = [removed keys]
(5, 0, None) = [6, 10, 15]
(5, 0, 2) = [6, 10]
(5,-1, None) = [0, 6, 10, 15]
(5, -1, 3) = [0, 6, 10]
(3, 2, None) = [15]
(3, 5, None) = []
```

Parameters

- **bin_name** (*str*) – The name of the bin containing the list.
- **key** (*str*) – The key of the item in the list for which to search
- **offset** (*int*) – Begin removing and returning items with rank == rank(fount_item) + offset
- **count** (*int*) – If specified, the number of items to remove and return. If None, all items until end of list are returned.
- **inverted** (*bool*) – If True, the operation is inverted and items outside of the specified range are returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_key_list**(*bin_name*, *key_list*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_remove_by_key operation to be used with operate or operate_ordered

The operation removes items, specified by the keys in key_list from the map stored in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key_list** (*list*) – A list of keys to be removed from the map.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If true, keys with values not specified in the key_list will be removed, and those keys specified in the key_list will be kept. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_key_range**(*bin_name*, *key_range_start*, *key_range_end*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_remove_by_key_range operation to be used with operate or operate_ordered

The operation removes items, with keys between key_range_start(inclusive) and key_range_end(exclusive) from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **key_range_start** – The start of the range of keys to be removed. (Inclusive)
- **key_range_end** – The end of the range of keys to be removed. (Exclusive)
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, values outside of the specified range will be removed, and values inside of the range will be kept. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_rank**(*bin_name*, *rank*, *return_type*, *ctx=None*)

Creates a map_remove_by_rank operation to be used with operate or operate_ordered

The operation removes the item with the specified rank from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **rank** (*int*) – The rank of the entry to remove.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_rank_range**(*bin_name*, *rank_start*, *remove_amt*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_remove_by_rank_range operation to be used with operate or operate_ordered

The operation removes *remove_amt* items beginning with the item with the specified rank from the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **rank_start** (*int*) – The rank of the entry to remove.
- **remove_amt** (*int*) – The number of entries to remove.

- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, items with ranks inside the specified range should be kept, and all other entries removed. Default: False.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_value**(*bin_name*, *value*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_remove_by_value operation to be used with operate or operate_ordered

The operation removes key value pairs whose value matches the specified value.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value** – Entries with a value matching this argument will be removed from the map.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, entries with a value different than the specified value will be removed. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_value_list**(*bin_name*, *value_list*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_remove_by_value_list operation to be used with operate or operate_ordered

The operation removes key value pairs whose values are specified in the value_list.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value_list** (*list*) – Entries with a value contained in this list will be removed from the map.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, entries with a value contained in value_list will be kept, and all others will be removed and returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_value_range**(*bin_name*, *value_start*, *value_end*, *return_type*, *inverted=False*, *ctx=None*)

Creates a map_remove_by_value_range operation to be used with operate or operate_ordered

The operation removes items, with values between value_start(inclusive) and value_end(exclusive) from the map

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value_start** – The start of the range of values to be removed. (Inclusive)
- **value_end** – The end of the range of values to be removed. (Exclusive)
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **inverted** (*bool*) – If True, values outside of the specified range will be removed, and values inside of the range will be kept. Default: False
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_ctx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_remove_by_value_rank_range_relative**(*bin_name*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx=None*)

Create a map remove by value rank range relative operation

Create map remove by value relative to rank range operation. Server removes and returns map items nearest to value and greater by relative rank. Server returns selected data specified by return_type.

Note: This operation requires server version 4.3.0 or greater.

Examples

Examples for map `{0: 6, 10: 18, 6: 12, 15: 24}` and return type of `aerospike.MAP_RETURN_KEY`

```
(value, offset, count) = [removed keys]
(6, 0, None) = [0, 6, 10, 15]
(5, 0, 2) = [0, 6]
(7, -1, 1) = [0]
(7, -1, 3) = [0, 6, 10]
```

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **value** – The value of the entry in the map for which to search
- **offset** (*int*) – Begin removing and returning items with rank == rank(found_item) + offset
- **count** (*int*) – If specified, the number of items to remove and return. If None, all items with rank greater than found_item are returned.
- **return_type** – Specifies what to return from the operation.
- **inverted** (*bool*) – If True, the operation is inverted and items outside of the specified range are returned.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_set_policy**(*bin_name*, *policy*, *ctx=None*)

Creates a map_set_policy_operation to be used with operate or operate_ordered

The operation allows a user to set the policy for the map.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **policy** (*dict*) – The *map_policy dictionary*.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.map_operations.**map_size**(*bin_name*, *ctx=None*)

Creates a map_size operation to be used with operate or operate_ordered

The operation returns the size of the map stored in the specified bin.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **ctx** (*list*) – An optional list of nested CDT context operations (*cdt_cdx* object) for use on nested CDTs.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bit_operations module

Helper functions to create bit operation dictionary arguments for the *aerospike.Client.operate* and *aerospike.Client.operate_ordered* methods of the aerospike client.

> **Note:** Bitwise operations require server version >= 4.6.0

Bit offsets are oriented left to right. Negative offsets are supported and start backwards from the end of the target bitmap.

Offset examples:

- 0: leftmost bit in the map
- 4: fifth bit in the map
- -1: rightmost bit in the map
- -4: 3 bits from rightmost

Example:

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
from aerospike_helpers.operations import bitwise_operations
import sys

# Configure the client.
config = {"hosts": [("127.0.0.1", 3000)]}

# Create a client and connect it to the cluster.
try:
    client = aerospike.client(config).connect()
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

key = ("test", "demo", "foo")
five_ones_bin_name = "bitwise1"
five_one_blob = bytearray([1] * 5)

# Write the record.
try:
    client.put(key, {five_ones_bin_name: five_one_blob})
except ex.RecordError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))

# EXAMPLE 1: resize the five_ones bin to a bytesize of 10.
try:
    ops = [bitwise_operations.bit_resize(five_ones_bin_name, 10)]

    _, _, bins = client.get(key)
    _, _, _ = client.operate(key, ops)
    _, _, newbins = client.get(key)
```

(continues on next page)

(continued from previous page)

```
    print("EXAMPLE 1: before resize: ", bins)
    print("EXAMPLE 1: is now: ", newbins)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# EXAMPLE 2: shrink the five_ones bin to a bytesize of 5 from the front.
try:
    ops = [
        bitwise_operations.bit_resize(
            five_ones_bin_name, 5, resize_flags=aerospike.BIT_RESIZE_FROM_FRONT
        )
    ]

    _, _, bins = client.get(key)
    _, _, _ = client.operate(key, ops)
    _, _, newbins = client.get(key)
    print("EXAMPLE 2: before resize: ", bins)
    print("EXAMPLE 2: is now: ", newbins)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# Cleanup and close the connection to the Aerospike cluster.
client.remove(key)
client.close()

"""
EXPECTED OUTPUT:
EXAMPLE 1: before resize:  {'bitwise1': bytearray(b'\x01\x01\x01\x01\x01')}
EXAMPLE 1: is now:  {'bitwise1': bytearray(b'\x01\x01\x01\x01\x01\x00\x00\x00\x00\x00')}
EXAMPLE 2: before resize:  {'bitwise1': bytearray(b'\x01\x01\x01\x01\x01\x00\x00\x00\x00\
↪x00')}
EXAMPLE 2: is now:  {'bitwise1': bytearray(b'\x00\x00\x00\x00\x00')}
"""
```

Example:

```
from __future__ import print_function
import aerospike
from aerospike import exception as e
from aerospike_helpers.operations import bitwise_operations

config = {'hosts': [('127.0.0.1', 3000)]}
try:
    client = aerospike.client(config).connect()
except e.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(2)

key = ('test', 'demo', 'bit_example')
five_one_blob = bytearray([1] * 5)
```

(continues on next page)

(continued from previous page)

```
five_one_bin = 'bitwise1'

try:
    if client.exists(key):
        client.remove(key)
    bit_policy = {
        'map_write_mode': aerospike.BIT_WRITE_DEFAULT,
    }
    client.put(
        key,
        {
            five_one_bin: five_one_blob
        }
    )

    # Example 1: read bits
    ops = [
        bitwise_operations.bit_get(five_one_bin, 0, 40)
    ]
    print('=====EXAMPLE1=====')
    _, _, results = client.operate(key, ops)
    print(results)

    # Example 2: modify bits using the 'or' op, then read bits
    ops = [
        bitwise_operations.bit_or(five_one_bin, 0, 8, 1, bytearray([255]), bit_policy),
        bitwise_operations.bit_get(five_one_bin, 0, 40)
    ]
    print('=====EXAMPLE2=====')
    _, _, results = client.operate(key, ops)
    print(results)

    # Example 3: modify bits using the 'remove' op, then read bits'
    ops = [
        bitwise_operations.bit_remove(five_one_bin, 0, 2, bit_policy),
        bitwise_operations.bit_get(five_one_bin, 0, 24)
    ]
    print('=====EXAMPLE3=====')
    _, _, results = client.operate(key, ops)
    print(results)

except e.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))

client.close()

"""
EXPECTED OUTPUT:
=====EXAMPLE1=====
{'bitwise1': bytearray(b'\x01\x01\x01\x01\x01')}
=====EXAMPLE2=====
{'bitwise1': bytearray(b'\xff\x01\x01\x01\x01')}
```

(continues on next page)

(continued from previous page)

```
=====EXAMPLE3=====
{'bitwise1': bytearray(b'\x01\x01\x01')}
"""
```

See also:

Bits (Data Types).

aerospike_helpers.operations.bitwise_operations.**bit_add**(*bin_name*, *bit_offset*, *bit_size*, *value*, *sign*, *action*, *policy=None*)

Creates a bit_add_operation to be used with operate or operate_ordered.

Creates a bit add operation. Server adds value to the bin at bit_offset for bit_size. bit_size must <= 64. If Sign is true value will be treated as a signed number. If an underflow or overflow occurs, as_bit_overflow_action is used. Server returns nothing.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will be added.
- **bit_size** (*int*) – How many bits of value to add.
- **value** (*int*) – The value to be added.
- **sign** (*bool*) – True: treat value as signed, False: treat value as unsigned.
- **action** (*aerospike.constant*) – Action taken if an overflow/underflow occurs.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_and**(*bin_name*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_and_operation to be used with operate or operate_ordered.

Creates a bit and operation. Server performs an and op with value and bitmap in bin at bit_offset for bit_size. Server returns nothing.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will be modified.
- **bit_size** (*int*) – How many bits of value to and.
- **value_byte_size** (*int*) – Length of value in bytes.
- **value** (*bytes/byte array*) – Bytes to be used in and operation.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_count**(*bin_name*, *bit_offset*, *bit_size*)

Creates a bit_count_operation to be used with operate or operate_ordered.

Server returns an integer count of all set bits starting at bit_offset for bit_size bits.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the set bits will begin being counted.
- **bit_size** (*int*) – How many bits will be considered for counting.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_get**(*bin_name*, *bit_offset*, *bit_size*)
Creates a bit_get_operation to be used with operate or operate_ordered.

Server returns bits from bitmap starting at bit_offset for bit_size.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being read.
- **bit_size** (*int*) – How many bits to get.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_get_int**(*bin_name*, *bit_offset*, *bit_size*, *sign*)
Creates a bit_get_int_operation to be used with operate or operate_ordered.

Server returns an integer formed from the bits read from bitmap starting at bit_offset for bit_size.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being read.
- **bit_size** (*int*) – How many bits to get.
- **sign** (*bool*) – True: Treat read value as signed. False: treat read value as unsigned.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_insert**(*bin_name*, *byte_offset*, *value_byte_size*, *value*, *policy=None*)
Creates a bit_insert_operation to be used with operate or operate_ordered.

Server inserts the bytes from value into the bitmap at byte_offset. No value is returned.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **byte_offset** (*int*) – The offset where the bytes will be inserted.
- **value_byte_size** (*int*) – Size of value in bytes.
- **value** (*bytes/byte array*) –
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_lscan**(*bin_name*, *bit_offset*, *bit_size*, *value*)
Creates a bit_lscan_operation to be used with operate or operate_ordered.

Server returns an integer representing the bit offset of the first occurence of the specified value bit. Starts scanning at bit_offset for bit_size. Returns -1 if value not found.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being scanned.
- **bit_size** (*int*) – How many bits to scan.
- **value** (*bool*) – True: look for 1, False: look for 0.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_lshift**(*bin_name*, *bit_offset*, *bit_size*, *shift*, *policy=None*)

Creates a bit_lshift_operation to be used with operate or operate_ordered.

Server left shifts bitmap starting at bit_offset for bit_size by shift bits. No value is returned.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being shifted.
- **bit_size** (*int*) – The number of bits that will be shifted by shift places.
- **shift** (*int*) – How many bits to shift by.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_not**(*bin_name*, *bit_offset*, *bit_size*, *policy=None*)

Creates a bit_not_operation to be used with operate or operate_ordered.

Server negates bitmap starting at bit_offset for bit_size. No value is returned.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being scanned.
- **bit_size** (*int*) – How many bits to scan.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_or**(*bin_name*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_or_operation to be used with operate or operate_ordered.

Creates a bit or operation. Server performs bitwise or with value and bitmap in bin at bit_offset for bit_size. Server returns nothing.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being compared.

- **bit_size** (*int*) – How many bits of value to or.
- **value_byte_size** (*int*) – Length of value in bytes.
- **value** (*bytes/byte array*) – Value to be used in or operation.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_remove**(*bin_name*, *byte_offset*, *byte_size*, *policy=None*)

Creates a bit_remove_operation to be used with operate or operate_ordered.

Remove bytes from bitmap at byte_offset for byte_size.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **byte_offset** (*int*) – Position of bytes to be removed.
- **byte_size** (*int*) – How many bytes to remove.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_resize**(*bin_name*, *byte_size*, *policy=None*, *resize_flags=<MagicMock id='140444815844624'>*)

Creates a bit_resize_operation to be used with operate or operate_ordered.

Change the size of a bytes bin stored in a record on the Aerospike Server.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **byte_size** (*int*) – The new size of the bytes.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.
- **resize_flags** (*int, optional*) – Flags modifying the behavior of the resize. This should be constructed by bitwise or'ing together any of the values: *Bitwise Resize Flags* . e.g. *aerospike.BIT_RESIZE_GROW_ONLY | aerospike.BIT_RESIZE_FROM_FRONT* default: *aerospike.BIT_RESIZE_DEFAULT*

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_rscan**(*bin_name*, *bit_offset*, *bit_size*, *value*)

Creates a bit_rscan_operation to be used with operate or operate_ordered.

Server returns an integer representing the bit offset of the last occurence of the specified value bit. Starts scanning at bit_offset for bit_size. Returns -1 if value not found.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being scanned.
- **bit_size** (*int*) – How many bits to scan.

- **value** (*bool*) – True: Look for 1, False: look for 0.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_rshift**(*bin_name*, *bit_offset*, *bit_size*, *shift*, *policy=None*)

Creates a bit_rshift_operation to be used with operate or operate_ordered.

Server right shifts bitmap starting at bit_offset for bit_size by shift bits. No value is returned.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being shifted.
- **bit_size** (*int*) – The number of bits that will be shifted by shift places.
- **shift** (*int*) – How many bits to shift by.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_set**(*bin_name*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_set_operation to be used with operate or operate_ordered.

Set the value on a bitmap at bit_offset for bit_size in a record on the Aerospike Server.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will be set.
- **bit_size** (*int*) – How many bits of value to write.
- **value_byte_size** (*int*) – Size of value in bytes.
- **value** (*bytes/byte array*) – The value to be set.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_subtract**(*bin_name*, *bit_offset*, *bit_size*, *value*, *sign*, *action*, *policy=None*)

Creates a bit_subtract_operation to be used with operate or operate_ordered.

Creates a bit add operation. Server subtracts value from the bits at bit_offset for bit_size. bit_size must <= 64. If sign is true value will be treated as a signed number. If an underflow or overflow occurs, as_bit_overflow_action is used. Server returns nothing.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will be subtracted.
- **bit_size** (*int*) – How many bits of value to subtract.
- **value** (*int*) – The value to be subtracted.
- **sign** (*bool*) – True: treat value as signed, False: treat value as unsigned.

- **action** (*aerospike.constant*) – Action taken if an overflow/underflow occurs.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.bitwise_operations.**bit_xor**(*bin_name*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_xor_operation to be used with operate or operate_ordered.

Creates a bit and operation. Server performs bitwise xor with value and bitmap in bin at bit_offset for bit_size. Server returns nothing.

Parameters

- **bin_name** (*str*) – The name of the bin containing the map.
- **bit_offset** (*int*) – The offset where the bits will start being compared.
- **bit_size** (*int*) – How many bits of value to xor.
- **value_byte_size** (*int*) – Length of value in bytes.
- **value** (*bytes/byte array*) – Value to be used in xor operation.
- **policy** (*dict, optional*) – The *bit_policy policy* dictionary. default: None.

Returns A dictionary usable in operate or operate_ordered. The format of the dictionary should be considered an internal detail, and subject to change.

aerospike_helpers.operations.hll_operations module

Helper functions to create HyperLogLog operation dictionary arguments for the *aerospike.Client.operate* and *aerospike.Client.operate_ordered* methods of the aerospike client. HyperLogLog bins and operations allow for your application to form fast, reasonable approximations of members in the union or intersection between multiple HyperLogLog bins. HyperLogLog's estimates are a balance between complete accuracy and efficient savings in space and speed in dealing with extremely large datasets.

Note: HyperLogLog operations require server version >= 4.9.0

See also:

HyperLogLog (Data Type) more info..

Example:

```
from __future__ import print_function
import sys

import aerospike
from aerospike import exception as ex
from aerospike_helpers.operations import hll_operations as hll_ops
from aerospike_helpers.operations import operations


TEST_NS = "test"
TEST_SET = "demo"
```

(continues on next page)

(continued from previous page)

```
NUM_INDEX_BITS = 12
NUM_MH_BITS = 24

# Configure the client.
config = {"hosts": [("127.0.0.1", 3000)]}

# Create a client and connect it to the cluster.
try:
    client = aerospike.client(config).connect()
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# Add HLL bins.
customers = ["Amy", "Farnsworth", "Scruffy"]
customer_record_keys = [
    (TEST_NS, TEST_SET, "Amy"),
    (TEST_NS, TEST_SET, "Farnsworth"),
    (TEST_NS, TEST_SET, "Scruffy"),
]
items_viewed = [
    ("item%s" % str(i) for i in range(0, 500)),
    ("item%s" % str(i) for i in range(0, 750)),
    ("item%s" % str(i) for i in range(250, 1000)),
]

for customer, key, items in zip(customers, customer_record_keys, items_viewed):
    ops = [
        operations.write("name", customer),
        hll_ops.hll_add("viewed", list(items), NUM_INDEX_BITS, NUM_MH_BITS),
    ]

    try:
        client.operate(key, ops)
    except ex.ClientError as e:
        print("Error: {0} [{1}]".format(e.msg, e.code))
        sys.exit(1)

# Find out how many items viewed Amy, Farnsworth, and Scruffy have in common.
Farnsworth_viewed = client.get(customer_record_keys[1])[2]["viewed"]
Scruffy_viewed = client.get(customer_record_keys[2])[2]["viewed"]
viewed = [Farnsworth_viewed, Scruffy_viewed]
ops = [hll_ops.hll_get_intersect_count("viewed", viewed)]

try:
    _, _, res = client.operate(customer_record_keys[0], ops)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

print(
    "Estimated items viewed intersection: %d."
```

(continues on next page)

(continued from previous page)

```
    % res["viewed"]
)
print("Actual intersection: 250.\n")

# Find out how many unique products Amy, Farnsworth, and Scruffy have viewed.
Farnsworth_viewed = client.get(customer_record_keys[1])[2]["viewed"]
Scruffy_viewed = client.get(customer_record_keys[2])[2]["viewed"]
viewed = [Farnsworth_viewed, Scruffy_viewed]
ops = [hll_ops.hll_get_union_count("viewed", viewed)]

try:
    _, _, res = client.operate(customer_record_keys[0], ops)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

print(
    "Estimated items viewed union: %d."
    % res["viewed"]
)
print("Actual union: 1000.\n")

# Find the similarity of Amy, Farnsworth, and Scruffy's product views.
Farnsworth_viewed = client.get(customer_record_keys[1])[2]["viewed"]
Scruffy_viewed = client.get(customer_record_keys[2])[2]["viewed"]
viewed = [Farnsworth_viewed, Scruffy_viewed]
ops = [hll_ops.hll_get_similarity("viewed", viewed)]

try:
    _, _, res = client.operate(customer_record_keys[0], ops)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

print(
    "Estimated items viewed similarity: %f%%."
    % (res["viewed"] * 100)
)
print("Actual similarity: 25%.")

"""
Expected output:
Estimated items viewed intersection: 235.
Actual intersection: 250.

Estimated items viewed union: 922.
Actual union: 1000.

Estimated items viewed similarity: 25.488069%.
Actual similarity: 25%.
"""
```

aerospike_helpers.operations.hll_operations.**hll_add**(*bin_name*, *values*, *index_bit_count=None*, *mh_bit_count=None*, *policy=None*)

Creates a hll_add operation to be used with operate, or operate_ordered.

Server will add the values to the hll bin. If the HLL bin does not exist, it will be created with index_bit_count and/or mh_bit_count if they have been supplied.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **values** – The values to be added to the HLL set.
- **index_bit_count** – An optional number of index bits. Must be bewtween 4 and 16 inclusive.
- **mh_bit_count** – An optional number of min hash bits. Must be bewtween 4 and 58 inclusive.
- **policy** (*dict*) – An optional dictionary of *hll policy options*.

aerospike_helpers.operations.hll_operations.**hll_describe**(*bin_name*)

Creates a hll_describe operation to be used with operate, or operate_ordered.

Server returns index and minhash bit counts used to create HLL bin in a list of integers. The list size is 2.

Parameters **bin_name** (*str*) – The name of the bin to be operated on.

aerospike_helpers.operations.hll_operations.**hll_fold**(*bin_name*, *index_bit_count*)

Creates a hll_fold operation to be used with operate, or operate_ordered.

Servers folds index_bit_count to the specified value. This can only be applied when minhash bit count on the HLL bin is 0. Server does not return a value.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index_bit_count** – number of index bits. Must be bewtween 4 and 16 inclusive.

aerospike_helpers.operations.hll_operations.**hll_get_count**(*bin_name*)

Creates a hll_get_count operation to be used with operate, or operate_ordered.

Server returns estimated count of elements in the HLL bin.

Parameters **bin_name** (*str*) – The name of the bin to be operated on.

aerospike_helpers.operations.hll_operations.**hll_get_intersect_count**(*bin_name*, *hll_list*)

Creates a hll_get_intersect_count operation to be used with operate, or operate_ordered.

Server returns estimate of elements that would be contained by the intersection of these HLL objects.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **hll_list** (*list*) – The HLLs to be intersected.

aerospike_helpers.operations.hll_operations.**hll_get_similarity**(*bin_name*, *hll_list*)

Creates a hll_get_similarity operation to be used with operate, or operate_ordered.

Server returns estimated similarity of the HLL objects. Server returns a float.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **hll_list** (*list*) – The HLLs used for similarity estimation.

aerospike_helpers.operations.hll_operations.**hll_get_union**(*bin_name*, *hll_list*)
Creates a hll_get_union operation to be used with operate, or operate_ordered.

Server returns an HLL object that is the union of all specified HLL objects in hll_list with the HLL bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **hll_list** (*list*) – The HLLs to be unioned.

aerospike_helpers.operations.hll_operations.**hll_get_union_count**(*bin_name*, *hll_list*)
Creates a hll_get_union_count operation to be used with operate, or operate_ordered.

Server returns the estimated count of elements that would be contained by the union of all specified HLL objects in the list with the HLL bin.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **hll_list** (*list*) – The HLLs to be unioned.

aerospike_helpers.operations.hll_operations.**hll_init**(*bin_name*, *index_bit_count=None*, *mh_bit_count=None*, *policy=None*)
Creates a hll_init operation to be used with operate, or operate_ordered.

Server creates a new HLL or resets an existing HLL. If index_bit_count and mh_bit_count are None, an existing HLL bin will be reset but retain its configuration. If 1 of index_bit_count or mh_bit_count are set, an existing HLL bin will set that config and retain its current value for the unset config. If the HLL bin does not exist, index_bit_count is required to create it, mh_bit_count is optional. Server does not return a value.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index_bit_count** – An optional number of index bits. Must be bewtween 4 and 16 inclusive.
- **mh_bit_count** – An optional number of min hash bits. Must be bewtween 4 and 58 inclusive.
- **policy** (*dict*) – An optional dictionary of *hll policy options*.

aerospike_helpers.operations.hll_operations.**hll_refresh_count**(*bin_name*)
Creates a hll_refresh_count operation to be used with operate, or operate_ordered.

Server updates the cached count if it is stale. Server returns the count.

Parameters **bin_name** (*str*) – The name of the bin to be operated on.

aerospike_helpers.operations.hll_operations.**hll_set_union**(*bin_name*, *hll_list*, *policy=None*)
Creates a hll_set_union operation to be used with operate, or operate_ordered.

Server sets the union of all specified HLL objects with the HLL bin. Server returns nothing.

Parameters

- **bin_name** (*str*) – The name of the bin to be operated on.
- **hll_list** (*list*) – The HLLs who's union will be set.
- **policy** (*dict*) – An optional dictionary of *hll policy options*.

aerospike_helpers.operations.expression_operations module

This module provides helper functions to produce dictionaries to be used with the *aerospike.Client.operate* and *aerospike.Client.operate_ordered* methods of the aerospike module.

Expression operations support reading and writing the result of Aerospike expressions.

Note: Requires server version >= 5.6.0

aerospike_helpers.operations.expression_operations.**expression_read**(*bin_name*, *expression*, *expression_read_flags=<MagicMock id='140444813774736'>*)

Create an expression read operation dictionary.

Reads and returns the value produced by the evaluated expression.

Parameters

- **bin_name** – The name of the bin to read from. Even if no bin is being read from, the value will be returned with this bin name.
- **expression** – A compiled Aerospike expression, see expressions at *aerospike_helpers*.
- **expression_read_flags** – Optional, one or more Aerospike expression read flags, *Expression Read Flags*.

Returns A dictionary to be passed to operate or operate_ordered.

Example:

```
# Read the value of int bin "balance".
# Let 'client' be a connected aerospike client.
# Let int bin 'balance' == 50.

from aerospike_helpers.operations import expression_operations as expressions
from aerospike_helpers.expressions import *

expr = IntBin("balance").compile()
ops = [
    expressions.expression_read("balance", expr)
]
_, _, res = client.operate(self.key, ops)
print(res)

# EXPECTED OUTPUT: {"balance": 50}
```

aerospike_helpers.operations.expression_operations.**expression_write**(*bin_name*, *expression*, *expression_write_flags=<MagicMock id='140444822031568'>*)

Create an expression write operation dictionary.

Writes the value produced by the evaluated expression to the supplied bin.

Parameters

- **bin_name** – The name of the bin to write to.
- **expression** – A compiled Aerospike expression, see expressions at *aerospike_helpers*.
- **expression_write_flags** – Optional, one or more Aerospike expression write flags, *List Write Flags*.

Returns A dictionary to be passed to operate or operate_ordered.

Example:

```
# Write the value of int bin "balance" + 50 back to "balance".
# Let 'client' be a connected aerospike client.
# Let int bin 'balance' == 50.

from aerospike_helpers.operations import expression_operations as expressions
from aerospike_helpers.expressions import *

expr = Add(IntBin("balance"), 50).compile()
ops = [
    expressions.expression_write("balance", expr)
]
client.operate(self.key, ops)
_, _, res = client.get(self.key)
print(res)

# EXPECTED OUTPUT: {"balance": 100}
```

1.8.1.2 aerospike_helpers.expressions package

Classes for the creation and use of aerospike expressions. See:: Aerospike Expressions.

Aerospike Expressions are a small domain specific language that allow for filtering records in transactions by manipulating and comparing bins and record metadata. Expressions can be used everywhere that predicate expressions have been used and allow for expanded functionality and customizability.

In the Python client, Aerospike expressions are built using a series of classes that represent comparrison and logical operators, bins, metadata operations, and bin operations. Expressions are constructed using a Lisp like syntax by instantiating an expression that yields a boolean, such as Eq() or And(), while passing them other expressions and constants as arguments, and finally calling the compile() method. See the example below.

Example:

```
# See if integer bin "bin_name" contains a value equal to 10.
from aerospike_helpers import expressions as exp
expr = exp.Eq(exp.IntBin("bin_name"), 10).compile()
```

By passing these compiled expressions to transactions via the "expressions" policy field, the expressions will filter the results. See the example below.

Example:

```
from __future__ import print_function
import aerospike
from aerospike_helpers import expressions as exp
from aerospike import exception as ex
```

(continues on next page)

(continued from previous page)

```
import sys

TEST_NS = "test"
TEST_SET = "demo"
FIRST_RECORD_INDEX = 0
SECOND_RECORD_INDEX = 1
BIN_INDEX = 2

# Configure the client.
config = {"hosts": [("127.0.0.1", 3000)]}

# Create a client and connect it to the cluster.
try:
    client = aerospike.client(config).connect()
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# Write records
keys = [(TEST_NS, TEST_SET, i) for i in range(1, 5)]
records = [
            {'user': "Chief"     , 'team': "blue", 'scores': [6, 12, 4, 21], 'kd': 1.0,
↪'status': "MasterPlatinum" },
            {'user': "Arbiter"   , 'team': "blue", 'scores': [5, 10, 5, 8] , 'kd': 1.0,
↪'status': "MasterGold"     },
            {'user': "Johnson"   , 'team': "blue", 'scores': [8, 17, 20, 5], 'kd': 0.99,
↪'status': "SergeantGold"   },
            {'user': "Regret"    , 'team': "red" , 'scores': [4, 2, 3, 5]  , 'kd': 0.33,
↪'status': "ProphetSilver"  }
        ]

try:
    for key, record in zip(keys, records):
        client.put(key, record)
except ex.RecordError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))


# EXAMPLE 1: Get records for users who's top scores are above 20 using a scan.
try:
    expr = exp.GT(exp.ListGetByRank(None, aerospike.LIST_RETURN_VALUE, exp.ResultType.
↪INTEGER, -1, exp.ListBin("scores")), # rank -1 == largest value
                  20).compile()

    scan = client.scan(TEST_NS, TEST_SET)
    policy = {
        'expressions': expr
    }

    records = scan.results(policy)
    # This scan will only return the record for "Chief" since it is the only account␣
↪with a score over 20 using batch get.
```

(continues on next page)

(continued from previous page)

```
    print(records[FIRST_RECORD_INDEX][BIN_INDEX])
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# EXPECTED OUTPUT:
# {'user': 'Chief', 'team': 'blue', 'scores': [6, 12, 4, 21], 'kd': 1.0, 'status': 'MasterPlatinum
↪'}


# EXAMPLE 2: Get player's records with a kd >= 1.0 with a status including "Gold".
try:
    expr = exp.And(
        exp.CmpRegex(aerospike.REGEX_ICASE, '.*Gold', exp.StrBin('status')),
        exp.GE(exp.FloatBin("kd"), 1.0)).compile()

    policy = {
        'expressions': expr
    }

    records = client.get_many(keys, policy)
    # This get_many will only return the record for "Arbiter" since it is the only␣
↪account with a kd >= 1.0 and Gold status.
    print(records[SECOND_RECORD_INDEX][BIN_INDEX])
except ex.AerospikeError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)
finally:
    client.close()

# EXPECTED OUTPUT:
# {'user': 'Arbiter', 'team': 'blue', 'scores': [5, 10, 5, 8], 'kd': 1.0, 'status': 'MasterGold'}
```

By nesting expressions, complicated filters can be created. See the example below.

Example:

```
from aerospike_helpers import expressions as exp
expr = Eq(
    exp.ListGetByIndexRangeToEnd(ctx, aerospike.LIST_RETURN_VALUE, 0,
        exp.ListSort(ctx, aerospike.LIST_SORT_DEFAULT,
            exp.ListAppend(ctx, policy, value_x,
                exp.ListAppendItems(ctx, policy, value_y,
                    exp.ListInsert(ctx, policy, 1, value_z, bin_name))))),
    expected_answer
),
```

Note:

```
Aerospike expressions are evaluated server side, expressions used for filtering are␣
↪called filter-expressions
and do not return any values to the client or write any values to the server.
When the following documentation says an expression returns a "list expression", it␣
↪means that the expression returns a
```

(continues on next page)

(continued from previous page)

```
list during evalution on the server side. Expressions used with expression_read() or␣
↪expression_write() do send their
return values to the client or write them to the server. These expressions are called␣
↪operation-expressions.
When these docs say that an expression parameter requires an "integer or integer␣
↪expression".
It means it will accept a literal integer, or an expression that will return an integer␣
↪during evaluation. When the docs say
an expression returns an "expression" this means that the data type returned may vary,␣
↪usually depending on the `return_type` parameter.
```

Current Limitations:

```
Currently, Aerospike expressions for the python client do not support comparing as_
↪python_bytes blobs.
Comparrisions between constant map values and map expressions are  also unsupported.
```

The expressions module uses typehints, here are a table of custom typehints mapped to standard types.

Table 1: Title

Type Name	Type Description
TypeResultType	Optional[int]
TypeFixedEle	Union[int, float, str, bytes, dict]
TypeFixed	Optional[Dict[str, TypeFixedEle]]
TypeCompiledOp	Tuple[int, TypeResultType, TypeFixed, int]
TypeExpression	List[TypeCompiledOp]
TypeChild	Union[int, float, str, bytes, _AtomExpr]
TypeChildren	Tuple[TypeChild, ...]
TypeBinName	Union[_BaseExpr, str]
TypeListValue	Union[_BaseExpr, List[Any]]
TypeIndex	Union[_BaseExpr, int, aerospike.CDTInfinite]
TypeCDT	Union[None, List[cdt_ctx._cdt_ctx]]
TypeRank	Union[_BaseExpr, int, aerospike.CDTInfinite]
TypeCount	Union[_BaseExpr, int, aerospike.CDTInfinite]
TypeValue	Union[_BaseExpr, Any]
TypePolicy	Union[Dict[str, Any], None]
TypeComparisonArg	Union[_BaseExpr, int, str, list, dict, aerospike.CDTInfinite]
TypeGeo	Union[_BaseExpr, aerospike.GeoJSON]
TypeKey	Union[_BaseExpr, Any]
TypeKeyList	Union[_BaseExpr, List[Any]]
TypeBitValue	Union[bytes, bytearray]
TypeNumber	Union[_BaseExpr, int, float]
TypeFloat	Union[_BaseExpr, float]
TypeInteger	Union[_BaseExpr, int]
TypeBool	Union[_BaseExpr, bool]

Note: Requires server version >= 5.2.0

aerospike_helpers.expressions.basemodule

Base expressions include operators, bin, and meta data related expressions.

Example:

```
import aerospike_helpers.expressions.base as exp
# See if integer bin "bin_name" contains a value equal to 10.
expr = exp.Eq(exp.IntBin("bin_name"), 10).compile()
```

class aerospike_helpers.expressions.base.**And**(**exprs*)

Create an "and" operator that applies to a variable amount of expressions.

__init__(**exprs*)

Create an "and" operator that applies to a variable amount of expressions.

Parameters ***exprs** (*_BaseExpr*) – Variable amount of expressions to be ANDed together.

Returns (boolean value)

Example:

```
# (a > 5 || a == 0) && b < 3
expr = And(
    Or(
        GT(IntBin("a"), 5),
        Eq(IntBin("a"), 0)),
    LT(IntBin("b"), 3)).compile()
```

class aerospike_helpers.expressions.base.**BinExists**(*bin*)

Create an expression that returns True if bin exists.

__init__(*bin*)

Create an expression that returns True if bin exists.

Parameters **bin** (*str*) – bin name.

Return (boolean value) True if bin exists, False otherwise.

Example:

```
#Bin "a" exists in record.
expr = BinExists("a").compile()
```

class aerospike_helpers.expressions.base.**BinType**(*bin*)

Create an expression that returns the type of a bin as one of the aerospike *bin types*

__init__(*bin*)

Create an expression that returns the type of a bin as one of the aerospike *bin types*.

Parameters **bin** (*str*) – bin name.

Return (integer value) returns the bin type.

Example:

```
# bin "a" == type string.
expr = Eq(BinType("a"), aerospike.AS_BYTES_STRING).compile()
```

class aerospike_helpers.expressions.base.**BlobBin**(*bin*)

Create an expression that returns a bin as a blob. Returns the unknown-value if the bin is not a blob.

__init__(*bin*)

Create an expression that returns a bin as a blob. Returns the unknown-value if the bin is not a blob.

Parameters **bin** (*str*) – Bin name.

:return (blob bin)

Example:

```
#. Blob bin "a" == bytearray([0x65, 0x65])
expr = Eq(BlobBin("a"), bytearray([0x65, 0x65])).compile()
```

class aerospike_helpers.expressions.base.**BoolBin**(*bin*)

Create an expression that returns a bin as a boolean. Returns the unknown-value if the bin is not a boolean.

__init__(*bin*)

Create an expression that returns a bin as a boolean. Returns the unknown-value if the bin is not a boolean.

Parameters **bin** (*str*) – Bin name.

Returns (boolean bin)

Example:

```
# Boolean bin "a" is True.
expr = BoolBin("a").compile()
```

class aerospike_helpers.expressions.base.**CmpGeo**(*expr0*, *expr1*)

Create a point within region or region contains point expression.

__init__(*expr0*, *expr1*)

Create a point within region or region contains point expression.

Parameters

- **expr0** (*TypeGeo*) – Left expression in comparrison.
- **expr1** (*TypeGeo*) – Right expression in comparrison.

Returns (boolean value)

Example:

```
# Geo bin "point" is within geo bin "region".
expr = CmpGeo(GeoBin("point"), GeoBin("region")).compile()
```

class aerospike_helpers.expressions.base.**CmpRegex**(*options*, *regex_str*, *cmp_str*)

Create an expression that performs a regex match on a string bin or value expression.

__init__(*options*, *regex_str*, *cmp_str*)

Create an expression that performs a regex match on a string bin or value expression.

Parameters

- **options** (*int*) – One of the aerospike regex constants, *Regex Flag Values*.
- **regex_str** (*str*) – POSIX regex string.
- **cmp_str** (*Union[_BaseExpr, str]*) – String expression to compare against.

Returns (boolean value)

Example:

```
# Select string bin "a" that starts with "prefix" and ends with "suffix".
# Ignore case and do not match newline.
expr = CmpRegex(aerospike.REGEX_ICASE | aerospike.REGEX_NEWLINE, "prefix.
↪*suffix", BinStr("a")).compile()
```

class aerospike_helpers.expressions.base.**Cond**(**exprs*)

Conditionally select an expression from a variable number of expression pairs followed by default expression action. Takes a set of test-expression/action-expression pairs. It evaluates each test one at a time. If a test returns True, 'cond' evaluates and returns the value of the corresponding expression and doesn't evaluate any of the other tests or expressions. The final expression is the default expression and is evaluated if all tests evaluate to False All actions-expressions must evaluate to the same type or be the Unknown expression.

__init__(**exprs*)

Conditionally select an expression from a variable number of expression pairs followed by default expression action. Takes a set of test-expression/action-expression pairs. It evaluates each test one at a time. If a test returns True, 'cond' evaluates and returns the value of the corresponding expression and doesn't evaluate any of the other tests or expressions. The final expression is the default expression and is evaluated if all tests evaluate to False All actions-expressions must evaluate to the same type or be the Unknown expression. Requires server version 5.6.0+.

Parameters ***exprs** (*_BaseExpr*) – bool exp1, action exp1, bool exp2, action exp2, ..., action-default

Returns (boolean value)

Example:

```
# Apply operator based on type and test if greater than 100.
expr = GT(
        Cond(
            Eq(IntBin("type"), 0),
                Add(IntBin("val1"), IntBin("val2")),
            Eq(IntBin("type"), 1),
                Sub(IntBin("val1"), IntBin("val2")),
            Eq(IntBin("type"), 2),
                Mul(IntBin("val1"), IntBin("val2")))
        100).compile()
```

class aerospike_helpers.expressions.base.**Def**(*var_name*, *expr*)

Assign variable to an expression that can be accessed later.

__init__(*var_name*, *expr*)

Assign variable to an expression that can be accessed later. Requires server version 5.6.0+.

Parameters

- **var_name** (*str*) – Variable name.
- **expr** (*_BaseExpr*) – Variable is set to result of this expression.

Returns (a variabe name expression pair)

Example:

```
# for int bin "a", 5 < a < 10
expr = Let(Def("x", IntBin("a")),
        And(
            LT(5, Var("x")),
            LT(Var("x"), 10))).compile()
```

class aerospike_helpers.expressions.base.**DeviceSize**

Create an expression that returns record size on disk. If server storage-engine is memory, then zero is returned. This expression usually evaluates quickly because record meta data is cached in memory.

__init__()

Create an expression that returns record size on disk. If server storage-engine is memory, then zero is returned. This expression usually evaluates quickly because record meta data is cached in memory.

Return (integer value) Uncompressed storage size of the record.

Example:

```
# Record device size >= 100 KB.
expr = GE(DeviceSize(), 100 * 1024).compile()
```

class aerospike_helpers.expressions.base.**DigestMod**(*mod*)

Create an expression that returns record digest modulo as integer.

__init__(*mod*)

Create an expression that returns record digest modulo as integer.

Parameters **mod** (*int*) – Divisor used to divide the digest to get a remainder.

Return (integer value) Value in range 0 and mod (exclusive).

Example:

```
# Records that have digest(key) % 3 == 1.
expr = Eq(DigestMod(3), 1).compile()
```

class aerospike_helpers.expressions.base.**Eq**(*expr0*, *expr1*)

Create an equals, (==) expression.

__init__(*expr0*, *expr1*)

Create an equals, (==) expression.

Parameters

- **expr0** (*TypeComparisonArg*) – Left argument to ==.
- **expr1** (*TypeComparisonArg*) – Right argument to ==.

Returns (boolean value)

Example:

```
# Integer bin "a" == 11
expr = Eq(IntBin("a"), 11).compile()
```

class aerospike_helpers.expressions.base.**Exclusive**(**exprs*)

Create an expression that returns True if only one of the expressions are True.

__init__(**exprs*)

Create an expression that returns True if only one of the expressions are True.

Parameters ***exprs** (*_BaseExpr*) – Variable amount of expressions to be checked.

Returns (boolean value)

Example:

```
# exclusive(a == 0, b == 0)
expr = exclusive(
                Eq(IntBin("a"), 0),
                Eq(IntBin("b"), 0)).compile()
```

class aerospike_helpers.expressions.base.**FloatBin**(*bin*)

Create an expression that returns a bin as a float. Returns the unknown-value if the bin is not a float.

__init__(*bin*)

Create an expression that returns a bin as a float. Returns the unknown-value if the bin is not a float.

Parameters **bin** (*str*) – Bin name.

Returns (float bin)

Example:

```
# Float bin "a" > 2.71.
expr = GT(FloatBin("a"), 2.71).compile()
```

class aerospike_helpers.expressions.base.**GE**(*expr0*, *expr1*)

Create a greater than or equal to (>=) expression.

__init__(*expr0*, *expr1*)

Create a greater than or equal to (>=) expression.

Parameters

- **expr0** (*TypeComparisonArg*) – Left argument to >=.
- **expr1** (*TypeComparisonArg*) – Right argument to >=.

Returns (boolean value)

Example:

```
# Integer bin "a" >= 88.
expr = GE(IntBin("a"), 88).compile()
```

class aerospike_helpers.expressions.base.**GT**(*expr0*, *expr1*)

Create a greater than (>) expression.

__init__(*expr0*, *expr1*)

Create a greater than (>) expression.

Parameters

- **expr0** (*TypeComparisonArg*) – Left argument to >.
- **expr1** (*TypeComparisonArg*) – Right argument to >.

Returns (boolean value)

Example:

```
# Integer bin "a" > 8.
expr = GT(IntBin("a"), 8).compile()
```

class aerospike_helpers.expressions.base.**GeoBin**(*bin*)
Create an expression that returns a bin as a geojson. Returns the unknown-value if the bin is not a geojson.

__init__(*bin*)
Create an expression that returns a bin as a geojson. Returns the unknown-value if the bin is not a geojson.

Parameters **bin** (*str*) – Bin name.

:return (geojson bin)

Example:

```
#GeoJSON bin "a" contained by GeoJSON bin "b".
expr = CmpGeo(GeoBin("a"), GeoBin("b")).compile()
```

class aerospike_helpers.expressions.base.**HLLBin**(*bin*)
Create an expression that returns a bin as a HyperLogLog. Returns the unknown-value if the bin is not a HyperLogLog.

__init__(*bin*)
Create an expression that returns a bin as a HyperLogLog. Returns the unknown-value if the bin is not a HyperLogLog.

Parameters **bin** (*str*) – Bin name.

:return (HyperLogLog bin)

Example:

```
# Does HLL bin "a" have a hll_count > 1000000.
expr = GT(HllGetCount(HllBin("a"), 1000000)).compile()
```

class aerospike_helpers.expressions.base.**IntBin**(*bin*)
Create an expression that returns a bin as an integer. Returns the unknown-value if the bin is not an integer.

__init__(*bin*)
Create an expression that returns a bin as an integer. Returns the unknown-value if the bin is not an integer.

Parameters **bin** (*str*) – Bin name.

Returns (integer bin)

Example:

```
# Integer bin "a" == 200.
expr = Eq(IntBin("a"), 200).compile()
```

class aerospike_helpers.expressions.base.**IsTombstone**
Create an expression that returns if record has been deleted and is still in tombstone state. This expression usually evaluates quickly because record meta data is cached in memory. NOTE: this is only applicable for XDR filter expressions.

__init__()
Create an expression that returns if record has been deleted and is still in tombstone state. This expression usually evaluates quickly because record meta data is cached in memory. NOTE: this is only applicable for XDR filter expressions.

Return (boolean value) True if the record is a tombstone, false otherwise.

Example:

```
# Detect deleted records that are in tombstone state.
expr = IsTombstone().compile()
```

class aerospike_helpers.expressions.base.**KeyBlob**

Create an expression that returns the key as a blob. Returns the unknown-value if the key is not a blob.

__init__()

Create an expression that returns the key as a blob. Returns the unknown-value if the key is not a blob.

Return (blob value) Blob value of the key if the key is a blob.

Example:

```
# blob record key <= bytearray([0x65, 0x65]).
expr = GE(KeyBlob(), bytearray([0x65, 0x65])).compile()
```

class aerospike_helpers.expressions.base.**KeyExists**

Create an expression that returns if the primary key is stored in the record storage data as a boolean expression. This would occur on record write, when write policies set the *key* field to aerospike.POLICY_KEY_SEND.

__init__()

Create an expression that returns if the primary key is stored in the record storage data as a boolean expression. This would occur on record write, when write policies set the *key* field to aerospike.POLICY_KEY_SEND.

Return (boolean value) True if the record has a stored key, false otherwise.

Example:

```
# Key exists in record meta data.
expr = KeyExists().compile()
```

class aerospike_helpers.expressions.base.**KeyInt**

Create an expression that returns the key as an integer. Returns the unknown-value if the key is not an integer.

__init__()

Create an expression that returns the key as an integer. Returns the unknown-value if the key is not an integer.

Return (integer value) Integer value of the key if the key is an integer.

Example:

```
# Integer record key >= 10000.
expr = GE(KeyInt(), 10000).compile()
```

class aerospike_helpers.expressions.base.**KeyStr**

Create an expression that returns the key as a string. Returns the unknown-value if the key is not a string.

__init__()

Create an expression that returns the key as a string. Returns the unknown-value if the key is not a string.

Return (string value) string value of the key if the key is an string.

Example:

```
# string record key == "aaa".
expr = Eq(KeyStr(), "aaa").compile()
```

class aerospike_helpers.expressions.base.**LE**(*expr0*, *expr1*)
Create a less than or equal to (<=) expression.

__init__(*expr0*, *expr1*)
Create a less than or equal to (<=) expression.

Parameters

- **expr0** (*TypeComparisonArg*) – Left argument to <=.
- **expr1** (*TypeComparisonArg*) – Right argument to <=.

Returns (boolean value)

Example:

```
# Integer bin "a" <= 1.
expr = LE(IntBin("a"), 1).compile()
```

class aerospike_helpers.expressions.base.**LT**(*expr0*, *expr1*)
Create a less than (<) expression.

__init__(*expr0*, *expr1*)
Create a less than (<) expression.

Parameters

- **expr0** (*TypeComparisonArg*) – Left argument to <.
- **expr1** (*TypeComparisonArg*) – Right argument to <.

Returns (boolean value)

Example:

```
# Integer bin "a" < 1000.
expr = LT(IntBin("a"), 1000).compile()
```

class aerospike_helpers.expressions.base.**LastUpdateTime**
Create an expression that the returns record last update time expressed as 64 bit integer nanoseconds since 1970-01-01 epoch.

__init__()
Create an expression that the returns record last update time expressed as 64 bit integer nanoseconds since 1970-01-01 epoch.

Return (integer value) When the record was last updated.

Example:

```
# Record last update time >= 2020-01-15.
expr = GE(LastUpdateTime(), 1577836800).compile()
```

class aerospike_helpers.expressions.base.**Let**(**exprs*)
Defines variables to be used within the <code>let</code> expression's scope. The last argument can be any expression and should make use of the defined variables. The <code>let</code> expression returns the evaluated result of the last argument. This expression is useful if you need to reuse the result of a complicated or expensive expression.

__init__(**exprs*)

Defines variables to be used within the <code>let</code> expression's scope. The last argument can be any expression and should make use of the defined variables. The <code>let</code> expression returns the evaluated result of the last argument. This expression is useful if Requires server version 5.6.0+.

Parameters ***exprs** (*_BaseExpr*) – Variable number of Def expressions followed by a scoped expression.

Returns (result of scoped expression)

Example:

```
# for int bin "a", 5 < a < 10
expr = Let(Def("x", IntBin("a")),
        And(
            LT(5, Var("x")),
            LT(Var("x"), 10))).compile()
```

class aerospike_helpers.expressions.base.**ListBin**(*bin*)

Create an expression that returns a bin as a list. Returns the unknown-value if the bin is not a list.

__init__(*bin*)

Create an expression that returns a bin as a list. Returns the unknown-value if the bin is not a list.

Parameters **bin** (*str*) – Bin name.

:return (list bin)

Example:

```
# List bin "a" contains at least one item with value "abc".
expr = GT(ListGetByValue(None, aerospike.LIST_RETURN_COUNT,
            ResultType.INTEGER, "abc", ListBin("a")),
        0).compile()
```

class aerospike_helpers.expressions.base.**MapBin**(*bin*)

Create an expression that returns a bin as a map. Returns the unknown-value if the bin is not a map.

__init__(*bin*)

Create an expression that returns a bin as a map. Returns the unknown-value if the bin is not a map.

Parameters **bin** (*str*) – Bin name.

:return (map bin)

Example:

```
# Map bin "a" size > 7.
expr = GT(MapSize(None, MapBin("a")), 7).compile()
```

class aerospike_helpers.expressions.base.**NE**(*expr0*, *expr1*)

Create a not equals (not ==) expressions.

__init__(*expr0*, *expr1*)

Create a not equals (not ==) expressions.

Parameters

- **expr0** (*TypeComparisonArg*) – Left argument to *not* ==.
- **expr1** (*TypeComparisonArg*) – Right argument to *not* ==.

Returns (boolean value)

Example:

```
# Integer bin "a" not == 13.
expr = NE(IntBin("a"), 13).compile()
```

class aerospike_helpers.expressions.base.**Not**(**exprs*)

Create a "not" (not) operator expression.

__init__(**exprs*)

Create a "not" (not) operator expression.

Parameters ***exprs** (*_BaseExpr*) – Variable amount of expressions to be negated.

Returns (boolean value)

Example:

```
# not (a == 0 or a == 10)
expr = Not(Or(
            Eq(IntBin("a"), 0),
            Eq(IntBin("a"), 10))).compile()
```

class aerospike_helpers.expressions.base.**Or**(**exprs*)

Create an "or" operator that applies to a variable amount of expressions.

__init__(**exprs*)

Create an "or" operator that applies to a variable amount of expressions.

Parameters ***exprs** (*_BaseExpr*) – Variable amount of expressions to be ORed together.

Returns (boolean value)

Example:

```
# (a == 0 || b == 0)
expr = Or(
        Eq(IntBin("a"), 0),
        Eq(IntBin("b"), 0)).compile()
```

class aerospike_helpers.expressions.base.**SetName**

Create an expression that returns record set name string. This expression usually evaluates quickly because record meta data is cached in memory.

__init__()

Create an expression that returns record set name string. This expression usually evaluates quickly because record meta data is cached in memory.

Return (string value) Name of the set this record belongs to.

Example:

```
# Record set name == "myset".
expr = Eq(SetName(), "myset").compile()
```

class aerospike_helpers.expressions.base.**SinceUpdateTime**

Create an expression that returns milliseconds since the record was last updated. This expression usually evaluates quickly because record meta data is cached in memory.

__init__()
Create an expression that returns milliseconds since the record was last updated. This expression usually evaluates quickly because record meta data is cached in memory.

Return (integer value) Number of milliseconds since last updated.

Example:

```
# Record last updated more than 2 hours ago.
expr = GT(SinceUpdateTime(), 2 * 60 * 1000).compile()
```

class aerospike_helpers.expressions.base.**StrBin**(*bin*)
Create an expression that returns a bin as a string. Returns the unknown-value if the bin is not a string.

__init__(*bin*)
Create an expression that returns a bin as a string. Returns the unknown-value if the bin is not a string.

Parameters **bin** (*str*) – Bin name.

Returns (string bin)

Example:

```
# String bin "a" == "xyz".
expr = Eq(StrBin("a"), "xyz").compile()
```

class aerospike_helpers.expressions.base.**TTL**
Create an expression that returns record expiration time (time to live) in integer seconds.

__init__()
Create an expression that returns record expiration time (time to live) in integer seconds.

Return (integer value) Number of seconds till the record will expire, returns -1 if the record never expires.

Example:

```
# Record expires in less than 1 hour.
expr = LT(TTL(), 60 * 60).compile()
```

class aerospike_helpers.expressions.base.**Unknown**
Create an 'Unknown' expression. Used to intentionally fail an expression operation such as expression_read or expression_write with error code 26, OpNotApplicable. When evaluated, the Unkown expression return the unknown-value returned by other failed expressions. These failures can be ignored with the expression flag aerospike.EXP_READ_EVAL_NO_FAIL. This expression is most useful as a case in a conditional expression used in an expression operation.

__init__()
Create an 'Unknown' expression. Used to intentionally fail an expression. The failure can be ignored with EXP_WRITE_EVAL_NO_FAIL or EXP_READ_NO_FAIL see *Expression Write Flags*.

:return (unknown value)

Example:

```
# If IntBin("balance") >= 50, get "balance" + 50.
# Otherwise, fail the expression via Unknown().
# This sort of expression is useful with expression operations
# expression_read() and expression_write().
Let(Def("bal", IntBin("balance")),
```

(continues on next page)

(continued from previous page)

```
    Cond(
        GE(Var("bal"), 50),
            Add(Var("bal"), 50),
        Unknown())
)
```

class aerospike_helpers.expressions.base.**Var**(*var_name*)
Retrieve expression value from a variable.

__init__(*var_name*)

Retrieve expression value from a variable. Requires server version 5.6.0+.

Parameters **var_name** (*str*) – Variable name.

Returns (value stored in variable)

Example:

```
# for int bin "a", 5 < a < 10
expr = Let(Def("x", IntBin("a")),
        And(
            LT(5, Var("x")),
            LT(Var("x"), 10))).compile()
```

class aerospike_helpers.expressions.base.**VoidTime**
Create an expression that returns record expiration time expressed as 64 bit integer nanoseconds since 1970-01-01 epoch.

__init__()
Create an expression that returns record expiration time expressed as 64 bit integer nanoseconds since 1970-01-01 epoch.

Return (integer value) Expiration time in nanoseconds since 1970-01-01.

Example:

```
# Record expires on 2021-01-01.
expr = And(
        GE(VoidTime(), 1609459200),
        LT(VoidTime(), 1609545600)).compile()
```

aerospike_helpers.expressions.list module

List expressions contain list read and modify expressions.

Example:

```
import aerospike_helpers.expressions as exp
#Take the size of list bin "a".
expr = exp.ListSize(None, exp.ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListAppend**(*ctx*, *policy*, *value*, *bin*)
Create an expression that appends value to end of list.

__init__(*ctx*, *policy*, *value*, *bin*)
Create an expression that appends value to end of list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional dictionary of list write options *list write options*.
- **value** (*TypeValue*) – Value or value expression to append to list.
- **bin** (*TypeBinName*) – List bin or list expression.

Returns List expression.

Example:

```
# Check if length of list bin "a" is > 5 after appending 1 item.
expr = GT(
        ListSize(None, ListAppend(None, None, 3, ListBin("a"))),
        5).compile()
```

class aerospike_helpers.expressions.list.**ListAppendItems**(*ctx*, *policy*, *value*, *bin*)
Create an expression that appends a list of items to the end of a list.

__init__(*ctx*, *policy*, *value*, *bin*)
Create an expression that appends a list of items to the end of a list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional dictionary of list write options *list write options*.
- **value** (*TypeValue*) – List or list expression of items to be appended.
- **bin** (*TypeBinName*) – Bin name or list expression.

Returns List expression.

Example:

```
# Check if length of list bin "a" is > 5 after appending multiple items.
expr = GT(
        ListSize(None, ListAppendItems(None, None, [3, 2], ListBin("a"))),
        5).compile()
```

class aerospike_helpers.expressions.list.**ListClear**(*ctx*, *bin*)
Create an expression that removes all items in a list.

__init__(*ctx*, *bin*)
Create an expression that removes all items in a list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **bin** (*TypeBinName*) – List bin or list expression to clear.

Returns List expression.

Example:

```
# Clear list value of list nested in list bin "a" index 1.
from aerospike_helpers import cdt_ctx
expr = ListClear([cdt_ctx.cdt_ctx_list_index(1)], "a").compile()
```

class aerospike_helpers.expressions.list.**ListGetByIndex**(*ctx*, *return_type*, *value_type*, *index*, *bin*)

Create an expression that selects list item identified by index and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value_type*, *index*, *bin*)

Create an expression that selects list item identified by index and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values
- **value_type** (*int*) – The value type that will be returned by this expression (ResultType).
- **index** (*TypeIndex*) – Integer or integer expression of index to get element at.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the value at index 0 in list bin "a". (assume this value is an integer)
expr = ListGetByIndex(None, aerospike.LIST_RETURN_VALUE, ResultType.INTEGER, 0,
↪ ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByIndexRange**(*ctx*, *return_type*, *index*, *count*, *bin*)

Create an expression that selects "count" list items starting at specified index and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *index*, *count*, *bin*)

Create an expression that selects "count" list items starting at specified index and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **index** (*TypeIndex*) – Integer or integer expression of index to start getting elements at.
- **count** (*TypeCount*) – Integer or integer expression for count of elements to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get elements at indexes 3, 4, 5, 6 in list bin "a".
expr = ListGetByIndexRange(None, aerospike.LIST_RETURN_VALUE, 3, 4, ListBin("a
↪")).compile()
```

(continues on next page)

(continued from previous page)

class aerospike_helpers.expressions.list.**ListGetByIndexRangeToEnd**(*ctx*, *return_type*, *index*, *bin*)

Create an expression that selects list items starting at specified index to the end of list and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *index*, *bin*)

Create an expression that selects list items starting at specified index to the end of list and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **index** (*TypeIndex*) – Integer or integer expression of index to start getting elements at.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get element 5 to end from list bin "a".
expr = ListGetByIndexRangeToEnd(None, aerospike.LIST_RETURN_VALUE, 5, ListBin(
↪"a")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByRank**(*ctx*, *return_type*, *value_type*, *rank*, *bin*)

Create an expression that selects list item identified by rank and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value_type*, *rank*, *bin*)

Create an expression that selects list item identified by rank and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **value_type** (*int*) – The value type that will be returned by this expression (ResultType).
- **rank** (*TypeRank*) – Rank integer or integer expression of element to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the smallest element in list bin "a".
expr = ListGetByRank(None, aerospike.LIST_RETURN_VALUE, aerospike.ResultType.
↪INTEGER, 0, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByRankRange**(*ctx*, *return_type*, *rank*, *count*, *bin*)
Create an expression that selects "count" list items starting at specified rank and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *rank*, *count*, *bin*)
Create an expression that selects "count" list items starting at specified rank and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **rank** (*TypeRank*) – Rank integer or integer expression of first element to get.
- **count** (*TypeCount*) – Count integer or integer expression for how many elements to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the 3 smallest elements in list bin "a".
expr = ListGetByRankRange(None, aerospike.LIST_RETURN_VALUE, 0, 3, ListBin("a
↪")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByRankRangeToEnd**(*ctx*, *return_type*, *rank*, *bin*)
Create an expression that selects list items starting at specified rank to the last ranked item and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *rank*, *bin*)
Create an expression that selects list items starting at specified rank to the last ranked item and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **rank** (*TypeRank*) – Rank integer or integer expression of first element to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the three largest elements in list bin "a".
expr = ListGetByRankRangeToEnd(None, aerospike.LIST_RETURN_VALUE, -3, ListBin(
↪"a")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByValue**(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects list items identified by value and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects list items identified by value and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **value** (*TypeValue*) – Value or value expression of element to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the index of the element with value, 3, in list bin "a".
expr = ListGetByValue(None, aerospike.LIST_RETURN_INDEX, 3, ListBin("a")).
↪compile()
```

class aerospike_helpers.expressions.list.**ListGetByValueList**(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects list items identified by values and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects list items identified by values and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **value** (*TypeListValue*) – List or list expression of values of elements to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
#Get the indexes of the the elements in list bin "a" with values [3, 6, 12].
expr = ListGetByValueList(None, aerospike.LIST_RETURN_INDEX, [3, 6, 12],␣
↪ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByValueRange**(*ctx*, *return_type*, *value_begin*, *value_end*, *bin*)
Create an expression that selects list items identified by value range and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value_begin*, *value_end*, *bin*)
Create an expression that selects list items identified by value range and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **value_begin** (*TypeValue*) – Value or value expression of first element to get.
- **value_end** (*TypeValue*) – Value or value expression of ending element.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get rank of values between 3 (inclusive) and 7 (exclusive) in list bin "a".
expr = ListGetByValueRange(None, aerospike.LIST_RETURN_RANK, 3, 7, ListBin("a
↪")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByValueRelRankRange**(*ctx*, *return_type*, *value*, *rank*, *count*, *bin*)

Create an expression that selects list items nearest to value and greater by relative rank with a count limit and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value*, *rank*, *count*, *bin*)

Create an expression that selects list items nearest to value and greater by relative rank with a count limit and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **value** (*TypeValue*) – Value or vaule expression to get items relative to.
- **rank** (*TypeRank*) – Rank intger expression. rank relative to "value" to start getting elements.
- **count** (*TypeCount*) – Integer value or integer value expression, how many elements to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the next 2 values in list bin "a" larger than 3.
expr = ListGetByValueRelRankRange(None, aerospike.LIST_RETURN_VALUE, 3, 1, 2,␣
↪ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListGetByValueRelRankRangeToEnd**(*ctx*, *return_type*, *value*, *rank*, *bin*)

Create an expression that selects list items nearest to value and greater by relative rank

__init__(*ctx*, *return_type*, *value*, *rank*, *bin*)

Create an expression that selects list items nearest to value and greater by relative rank and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values One of the aerospike list return types.
- **value** (*TypeValue*) – Value or vaule expression to get items relative to.
- **rank** (*TypeRank*) – Rank intger expression. rank relative to "value" to start getting elements.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the values of all elements in list bin "a" larger than 3.
expr = ListGetByValueRelRankRangeToEnd(None, aerospike.LIST_RETURN_VALUE, 3, 1,
↪ ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListIncrement**(*ctx*, *policy*, *index*, *value*, *bin*)

Create an expression that increments list[index] by value.

__init__(*ctx*, *policy*, *index*, *value*, *bin*)

Create an expression that increments list[index] by value.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional dictionary of list write options *list write options*.Optional list write policy.
- **index** (*TypeIndex*) – Index of value to increment.
- **value** (*TypeValue*) – Value or value expression.
- **bin** (*TypeBinName*) – Bin name or list expression.

Returns List expression.

Example:

```
# Check if incremented value in list bin "a" is the largest in the list.
expr = Eq(
        ListGetByRank(None, aerospike.LIST_RETURN_VALUE, ResultType.INTEGER, -
↪1, #rank of -1 == largest element.
             ListIncrement(None, None, 1, 5, ListBin("a"))),
        ListGetByIndex(None, aerospike.LIST_RETURN_VALUE, ResultType.INTEGER,␣
↪1,
             ListIncrement(None, None, 1, 5, ListBin("a")))
).compile()
```

class aerospike_helpers.expressions.list.**ListInsert**(*ctx*, *policy*, *index*, *value*, *bin*)

Create an expression that inserts value to specified index of list.

__init__(*ctx*, *policy*, *index*, *value*, *bin*)

Create an expression that inserts value to specified index of list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional dictionary of list write options *list write options*.
- **index** (*TypeIndex*) – Target index for insertion, integer or integer expression.
- **value** (*TypeValue*) – Value or value expression to be inserted.
- **bin** (*TypeBinName*) – Bin name or list expression.

Returns List expression.

Example:

```
# Check if list bin "a" has length > 5 after insert.
expr = GT(
        ListSize(None, ListInsert(None, None, 0, 3, ListBin("a"))),
        5).compile()
```

class aerospike_helpers.expressions.list.**ListInsertItems**(*ctx*, *policy*, *index*, *values*, *bin*)

Create an expression that inserts each input list item starting at specified index of list.

__init__(*ctx*, *policy*, *index*, *values*, *bin*)

Create an expression that inserts each input list item starting at specified index of list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional dictionary of list write options *list write options*.Optional list write policy.
- **index** (*TypeIndex*) – Target index where item insertion will begin, integer or integer expression.
- **values** (*TypeListValue*) – List or list expression of items to be inserted.
- **bin** (*TypeBinName*) – Bin name or list expression.

Returns List expression.

Example:

```
# Check if list bin "a" has length > 5 after inserting items.
expr = GT(
        ListSize(None, ListInsertItems(None, None, 0, [4, 7], ListBin("a"))),
        5).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByIndex**(*ctx*, *index*, *bin*)

Create an expression that removes "count" list items starting at specified index.

__init__(*ctx*, *index*, *bin*)

Create an expression that removes "count" list items starting at specified index.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **index** (*TypeIndex*) – Index integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Get size of list bin "a" after index 3 has been removed.
expr = ListSize(None, ListRemoveByIndex(None, 3, ListBin("a"))).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByIndexRange**(*ctx*, *index*, *count*, *bin*)

Create an expression that removes "count" list items starting at specified index.

__init__(*ctx*, *index*, *count*, *bin*)

Create an expression that removes "count" list items starting at specified index.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **index** (*TypeIndex*) – Starting index integer or integer expression of elements to remove.
- **count** (*TypeCount*) – Integer or integer expression, how many elements to remove.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Get size of list bin "a" after index 3, 4, and 5 have been removed.
expr = ListSize(None, ListRemoveByIndexRange(None, 3, 3, ListBin("a"))).
↪compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByIndexRangeToEnd**(*ctx*, *index*, *bin*)

Create an expression that removes list items starting at specified index to the end of list.

__init__(*ctx*, *index*, *bin*)

Create an expression that removes list items starting at specified index to the end of list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **index** (*TypeIndex*) – Starting index integer or integer expression of elements to remove.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Remove all elements starting from index 3 in list bin "a".
expr = ListRemoveByIndexRangeToEnd(None, 3, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByRank**(*ctx*, *rank*, *bin*)

Create an expression that removes list item identified by rank.

__init__(*ctx*, *rank*, *bin*)

Create an expression that removes list item identified by rank.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Remove smallest value in list bin "a".
expr = ListRemoveByRank(None, 0, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByRankRange**(*ctx*, *rank*, *count*, *bin*)

Create an expression that removes "count" list items starting at specified rank.

__init__(*ctx*, *rank*, *count*, *bin*)

Create an expression that removes "count" list items starting at specified rank.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to start removing at.
- **count** (*TypeCount*) – Count integer or integer expression of elements to remove.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Remove the 3 smallest items from list bin "a".
expr = ListRemoveByRankRange(None, 0, 3, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByRankRangeToEnd**(*ctx*, *rank*, *bin*)

Create an expression that removes list items starting at specified rank to the last ranked item.

__init__(*ctx*, *rank*, *bin*)

Create an expression that removes list items starting at specified rank to the last ranked item.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to start removing at.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Remove the 2 largest elements from List bin "a".
expr = ListRemoveByRankRangeToEnd(None, -2, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByValue**(*ctx*, *value*, *bin*)

Create an expression that removes list items identified by value.

__init__(*ctx*, *value*, *bin*)

Create an expression that removes list items identified by value.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **value** (*TypeValue*) – Value or value expression to remove.

- **bin** (*TypeBinName*) – bin name or bin expression.

Returns list expression.

Example:

```
# See if list bin "a", with `3` removed, is equal to list bin "b".
expr = Eq(ListRemoveByValue(None, 3, ListBin("a")), ListBin("b")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByValueList**(*ctx*, *values*, *bin*)

Create an expression that removes list items identified by values.

__init__(*ctx*, *values*, *bin*)

Create an expression that removes list items identified by values.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **values** (*TypeListValue*) – List of values or list expression.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Remove elements with values [1, 2, 3] from list bin "a".
expr = ListRemoveByValueList(None, [1, 2, 3], ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByValueRange**(*ctx*, *begin*, *end*, *bin*)

Create an expression that removes list items identified by value range (begin inclusive, end exclusive). If begin is None, the range is less than end. If end is None, the range is greater than or equal to begin.

__init__(*ctx*, *begin*, *end*, *bin*)

Create an expression that removes list items identified by value range (begin inclusive, end exclusive). If begin is None, the range is less than end. If end is None, the range is greater than or equal to begin.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **begin** (*TypeValue*) – Begin value or value expression for range.
- **end** (*TypeValue*) – End value or value expression for range.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Remove list of items with values >= 3 and < 7 from list bin "a".
expr = ListRemoveByValueRange(None, 3, 7, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByValueRelRankRange**(*ctx*, *value*, *rank*, *count*, *bin*)

Create an expression that removes list items nearest to value and greater by relative rank with a count limit.

__init__(*ctx*, *value*, *rank*, *count*, *bin*)

Create an expression that removes list items nearest to value and greater by relative rank with a count limit.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.

- **value** (*TypeValue*) – Start value or value expression.
- **rank** (*TypeRank*) – Rank integer or integer expression.
- **count** (*TypeCount*) – How many elements to remove.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# After removing the 3 elements larger than 4 by relative rank, does list bin
↪"a" include 9?.
expr = GT(
        ListGetByValue(None, aerospike.LIST_RETURN_COUNT, 9,
            ListRemoveByValueRelRankRange(None, 4, 1, 0, ListBin("a"))),
        0).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByValueRelRankToEnd**(*ctx*, *value*, *rank*, *bin*)

Create an expression that removes list items nearest to value and greater by relative rank.

__init__(*ctx*, *value*, *rank*, *bin*)

Create an expression that removes list items nearest to value and greater by relative rank.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **value** (*TypeValue*) – Start value or value expression.
- **rank** (*TypeRank*) – Rank integer or integer expression.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Remove elements larger than 4 by relative rank in list bin "a".
expr = ListRemoveByValueRelRankToEnd(None, 4, 1, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListSet**(*ctx*, *policy*, *index*, *value*, *bin*)

Create an expression that sets item value at specified index in list.

__init__(*ctx*, *policy*, *index*, *value*, *bin*)

Create an expression that sets item value at specified index in list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional dictionary of list write options *list write options*.Optional list write policy.
- **index** (*TypeIndex*) – index of value to set.
- **value** (*TypeValue*) – value or value expression to set index in list to.
- **bin** (*TypeBinName*) – bin name or list expression.

Returns List expression.

Example:

```
# Get smallest element in list bin "a" after setting index 1 to 10.
expr = ListGetByRank(None, aerospike.LIST_RETURN_VALUE, ResultType.INTEGER, 0,
                ListSet(None, None, 1, 10, ListBin("a"))).compile()
```

class aerospike_helpers.expressions.list.**ListSize**(*ctx*, *bin*)

Create an expression that returns list size.

__init__(*ctx*, *bin*)

Create an expression that returns list size.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Integer expression.

Example:

```
#Take the size of list bin "a".
expr = ListSize(None, ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListSort**(*ctx*, *order*, *bin*)

Create an expression that sorts a list.

__init__(*ctx*, *order*, *bin*)

Create an expression that sorts a list.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **order** (*int*) – Optional flags modifiying the behavior of list_sort. This should be constructed by bitwise or'ing together values from *List Sort Flags*.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns list expression.

Example:

```
# Get value of sorted list bin "a".
expr = ListSort(None, aerospike.LIST_SORT_DEFAULT, "a").compile()
```

aerospike_helpers.expressions.map module

Map expressions contain map read and modify expressions.

Example:

```
import aerospike_helpers.expressions as exp
#Take the size of map bin "b".
expr = exp.MapSize(None, exp.MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapClear**(*ctx*, *bin*)

Create an expression that removes all items in map.

__init__(*ctx*, *bin*)

Create an expression that removes all items in map.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Clear map bin "b".
expr = MapClear(None, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByIndex**(*ctx*, *return_type*, *value_type*, *index*, *bin*)

Create an expression that selects map item identified by index and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value_type*, *index*, *bin*)

Create an expression that selects map item identified by index and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **value_type** (*int*) – The value type that will be returned by this expression (ResultType).
- **index** (*TypeIndex*) – Integer or integer expression of index to get element at.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the value at index 0 in map bin "b". (assume this value is an integer)
expr = MapGetByIndex(None, aerospike.MAP_RETURN_VALUE, ResultType.INTEGER, 0,␣
↪MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByIndexRange**(*ctx*, *return_type*, *index*, *count*, *bin*)

Create an expression that selects "count" map items starting at specified index.

__init__(*ctx*, *return_type*, *index*, *count*, *bin*)

Create an expression that selects "count" map items starting at specified index and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **index** (*TypeIndex*) – Integer or integer expression of index to start getting elements at.
- **count** (*TypeCount*) – Integer or integer expression for count of elements to get.

- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get elements at indexes 3, 4, 5, 6 in map bin "b".
expr = MapGetByIndexRange(None, aerospike.MAP_RETURN_VALUE, 3, 4, MapBin("b")).
↪compile()
```

class aerospike_helpers.expressions.map.**MapGetByIndexRangeToEnd**(*ctx*, *return_type*, *index*, *bin*)

Create an expression that selects map items starting at specified index to the end of map.

__init__(*ctx*, *return_type*, *index*, *bin*)

Create an expression that selects map items starting at specified index to the end of map and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **index** (*TypeIndex*) – Integer or integer expression of index to start getting elements at.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get element at index 5 to end from map bin "b".
expr = MapGetByIndexRangeToEnd(None, aerospike.MAP_RETURN_VALUE, 5, MapBin("b
↪")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByKey**(*ctx*, *return_type*, *value_type*, *key*, *bin*)

Create an expression that selects map item identified by key and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value_type*, *key*, *bin*)

Create an expression that selects map item identified by key and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **value_type** (*int*) – The value type that will be returned by this expression (ResultType).
- **key** (*TypeKey*) – Key value or value expression of element to get.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the value at key "key0" in map bin "b". (assume the value at key0 is an␣
↪integer)
expr = MapGetByKey(None, aerospike.MAP_RETURN_VALUE, ResultType.INTEGER, "key0
↪", MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByKeyList**(*ctx*, *return_type*, *keys*, *bin*)

Create an expression that selects map items identified by keys and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *keys*, *bin*)

Create an expression that selects map items identified by keys and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **keys** (*TypeKeyList*) – List of key values or list expression.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get elements at keys "key3", "key4", "key5" in map bin "b".
expr = MapGetByKeyList(None, aerospike.MAP_RETURN_VALUE, ["key3", "key4", "key5
↪"], MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByKeyRange**(*ctx*, *return_type*, *begin*, *end*, *bin*)

Create an expression that selects map items identified by key range.

__init__(*ctx*, *return_type*, *begin*, *end*, *bin*)

Create an expression that selects map items identified by key range (begin inclusive, end exclusive). If begin is nil, the range is less than end. If end is aerospike.CDTInfinite(), the range is greater than equal to begin. Expression returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **begin** (*TypeKey*) – Key value or expression.
- **end** (*TypeKey*) – Key value or expression.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get elements at keys "key3", "key4", "key5", "key6" in map bin "b".
expr = MapGetByKeyRange(None, aerospike.MAP_RETURN_VALUE, "key3", "key7",␣
↪MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByKeyRelIndexRange**(*ctx*, *return_type*, *key*, *index*, *count*, *bin*)

Create an expression that selects map items nearest to key and greater by index with a count limit.

__init__(*ctx*, *return_type*, *key*, *index*, *count*, *bin*)

Create an expression that selects map items nearest to key and greater by index with a count limit. Expression returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **key** (*TypeKey*) – Key value or value expression.
- **index** (*TypeIndex*) – Index integer or integer value expression.
- **count** (*TypeCount*) – Integer count or integer value expression.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the next 2 elements with keys larger than "key3" from map bin "b".
expr = MapGetByKeyRelIndexRange(None, aerospike.MAP_RETURN_VALUE, "key3", 1, 2,
↪ MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByKeyRelIndexRangeToEnd**(*ctx*, *return_type*, *key*, *index*, *bin*)

Create an expression that selects map items nearest to key and greater by index with a count limit.

__init__(*ctx*, *return_type*, *key*, *index*, *bin*)

Create an expression that selects map items nearest to key and greater by index with a count limit. Expression returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **key** (*TypeKey*) – Key value or value expression.
- **index** (*TypeIndex*) – Index integer or integer value expression.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get elements with keys larger than "key2" from map bin "b".
expr = MapGetByKeyRelIndexRangeToEnd(None, aerospike.MAP_RETURN_VALUE, "key2",␣
↪1, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByRank**(*ctx*, *return_type*, *value_type*, *rank*, *bin*)
Create an expression that selects map items identified by rank and returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value_type*, *rank*, *bin*)
Create an expression that selects map items identified by rank and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **value_type** (*int*) – The value type that will be returned by this expression (ResultType).
- **rank** (*TypeRank*) – Rank integer or integer expression of element to get.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the smallest element in map bin "b".
expr = MapGetByRank(None, aerospike.MAP_RETURN_VALUE, aerospike.ResultType.
↪INTEGER, 0, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByRankRange**(*ctx*, *return_type*, *rank*, *count*, *bin*)
Create an expression that selects "count" map items starting at specified rank.

__init__(*ctx*, *return_type*, *rank*, *count*, *bin*)
Create an expression that selects "count" map items starting at specified rank and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **rank** (*TypeRank*) – Rank integer or integer expression of first element to get.
- **count** (*TypeCount*) – Count integer or integer expression for how many elements to get.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the 3 smallest elements in map bin "b".
expr = MapGetByRankRange(None, aerospike.MAP_RETURN_VALUE, 0, 3, MapBin("b")).
↪compile()
```

class aerospike_helpers.expressions.map.**MapGetByRankRangeToEnd**(*ctx*, *return_type*, *rank*, *bin*)
Create an expression that selects map items starting at specified rank to the last ranked item.

__init__(*ctx*, *return_type*, *rank*, *bin*)
Create an expression that selects map items starting at specified rank to the last ranked item and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **rank** (*TypeRank*) – Rank integer or integer expression of first element to get.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the three largest elements in map bin "b".
expr = MapGetByRankRangeToEnd(None, aerospike.MAP_RETURN_VALUE, -3, MapBin("b
↪")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByValue**(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects map items identified by value.

__init__(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects map items identified by value and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **value** (*TypeValue*) – Value or value expression of element to get.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the rank of the element with value, 3, in map bin "b".
expr = MapGetByValue(None, aerospike.MAP_RETURN_RANK, 3, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByValueList**(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects map items identified by values.

__init__(*ctx*, *return_type*, *value*, *bin*)
Create an expression that selects map items identified by values and returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.

- **value** (*TypeListValue*) – List or list expression of values of elements to get.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the indexes of the the elements in map bin "b" with values [3, 6, 12].
expr = MapGetByValueList(None, aerospike.MAP_RETURN_INDEX, [3, 6, 12], MapBin(
↪"b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByValueRange**(*ctx*, *return_type*, *value_begin*, *value_end*, *bin*)

Create an expression that selects map items identified by value range.

__init__(*ctx*, *return_type*, *value_begin*, *value_end*, *bin*)

Create an expression that selects map items identified by value range (begin inclusive, end exclusive). If begin is None, the range is less than end. If end is None, the range is greater than equal to begin. Expression returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **value_begin** (*TypeValue*) – Value or value expression of first element to get.
- **value_end** (*TypeValue*) – Value or value expression of ending element.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get elements with values between 3 and 7 from map bin "b".
expr = MapGetByValueRange(None, aerospike.MAP_RETURN_VALUE, 3, 7, MapBin("b")).
↪compile()
```

class aerospike_helpers.expressions.map.**MapGetByValueRelRankRange**(*ctx*, *return_type*, *value*, *rank*, *count*, *bin*)

Create an expression that selects map items nearest to value and greater by relative rank with a count limit. Expression returns selected data specified by return_type.

__init__(*ctx*, *return_type*, *value*, *rank*, *count*, *bin*)

Create an expression that selects map items nearest to value and greater by relative rank with a count limit. Expression returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike list return types.
- **value** (*TypeValue*) – Value or vaule expression to get items relative to.

- **rank** (*TypeRank*) – Rank intger expression. rank relative to "value" to start getting elements.
- **count** (*TypeCount*) – Integer value or integer value expression, how many elements to get.
- **bin** (*TypeBinName*) – List bin name or list expression.

Returns Expression.

Example:

```
# Get the next 2 values in map bin "b" larger than 3.
expr = MapGetByValueRelRankRange(None, aerospike.MAP_RETURN_VALUE, 3, 1, 2,␣
↪MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByValueRelRankRangeToEnd**(*ctx*, *return_type*, *value*, *rank*, *bin*)

Create an expression that selects map items nearest to value and greater by relative rank.

__init__(*ctx*, *return_type*, *value*, *rank*, *bin*)

Create an expression that selects map items nearest to value and greater by relative rank. Expression returns selected data specified by return_type.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values One of the aerospike map return types.
- **value** (*TypeValue*) – Value or vaule expression to get items relative to.
- **rank** (*TypeRank*) – Rank intger expression. rank relative to "value" to start getting elements.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Expression.

Example:

```
# Get the values of all elements in map bin "b" larger than 3.
expr = MapGetByValueRelRankRangeToEnd(None, aerospike.MAP_RETURN_VALUE, 3, 1,␣
↪MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapIncrement**(*ctx*, *policy*, *key*, *value*, *bin*)

Create an expression that increments a map value, by value, for all items identified by key. Valid only for numbers.

__init__(*ctx*, *policy*, *key*, *value*, *bin*)

Create an expression that increments a map value, by value, for all items identified by key. Valid only for numbers.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional map write policy.
- **key** (*TypeKey*) – Key value or value expression element to increment.
- **value** (*TypeValue*) – Increment element by value expression.

- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Increment element at 'vageta' in map bin "b" by 9000.
expr = MapIncrement(None, None, 'vageta', 9000, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapPut**(*ctx*, *policy*, *key*, *value*, *bin*)

Create an expression that writes key/val to map bin.

__init__(*ctx*, *policy*, *key*, *value*, *bin*)

Create an expression that writes key/val to map bin.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional map write policy.
- **key** (*TypeKey*) – Key value or value expression to put into map.
- **value** (*TypeValue*) – Value or value expression to put into map.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Put {27: 'key27'} into map bin "b".
expr = MapPut(None, None, 27, 'key27', MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapPutItems**(*ctx*, *policy*, *map*, *bin*)

Create an expression that writes each map item to map bin.

__init__(*ctx*, *policy*, *map*, *bin*)

Create an expression that writes each map item to map bin.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **policy** (*TypePolicy*) – Optional map write policy.
- **map** (*map*) – Map or map expression of items to put into target map.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Put {27: 'key27', 28: 'key28'} into map bin "b".
expr = MapPut(None, None, {27: 'key27', 28: 'key28'}, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByIndex**(*ctx*, *index*, *bin*)

Create an expression that removes map item identified by index.

__init__(*ctx*, *index*, *bin*)

Create an expression that removes map item identified by index.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.

- **index** (*TypeIndex*) – Index integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – Bin name or map expression.

Returns Map expression.

Example:

```
# Remove element with smallest key from map bin "b".
expr = MapRemoveByIndex(None, 0, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByIndexRange**(*ctx*, *index*, *count*, *bin*)

Create an expression that removes count map items starting at specified index.

__init__(*ctx*, *index*, *count*, *bin*)

Create an expression that removes count map items starting at specified index.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **index** (*TypeIndex*) – Starting index integer or integer expression of elements to remove.
- **count** (*TypeCount*) – Integer or integer expression, how many elements to remove.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Map expression.

Example:

```
# Get size of map bin "b" after index 3, 4, and 5 have been removed.
expr = MapSize(None, MapRemoveByIndexRange(None, 3, 3, MapBin("b"))).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByIndexRangeToEnd**(*ctx*, *index*, *bin*)

Create an expression that removes map items starting at specified index to the end of map.

__init__(*ctx*, *index*, *bin*)

Create an expression that removes map items starting at specified index to the end of map.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **index** (*TypeIndex*) – Starting index integer or integer expression of elements to remove.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Map expression.

Example:

```
# Remove all elements starting from index 3 in map bin "b".
expr = MapRemoveByIndexRangeToEnd(None, 3, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKey**(*ctx*, *key*, *bin*)

Create an expression that removes a map item identified by key.

__init__(*ctx*, *key*, *bin*)

Create an expression that removes a map item identified by key.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **key** (*TypeKey*) – Key value or value expression of key to element to remove.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Remove element at key 1 in map bin "b".
expr = MapRemoveByKey(None, 1, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKeyList**(*ctx*, *keys*, *bin*)

Create an expression that removes map items identified by keys.

__init__(*ctx*, *keys*, *bin*)

Create an expression that removes map items identified by keys.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **key** (*List[TypeKey]*) – List of key values or a list expression of keys to elements to remove.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Remove elements at keys [1, 2] in map bin "b".
expr = MapRemoveByKeyList(None, [1, 2], MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKeyRange**(*ctx*, *begin*, *end*, *bin*)

Create an expression that removes map items identified by key range (begin inclusive, end exclusive). If begin is None, the range is less than end. If end is None, the range is greater than equal to begin.

__init__(*ctx*, *begin*, *end*, *bin*)

Create an expression that removes map items identified by key range (begin inclusive, end exclusive). If begin is None, the range is less than end. If end is None, the range is greater than equal to begin.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **begin** (*TypeValue*) – Begin value expression.
- **end** (*TypeValue*) – End value expression.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Remove elements at keys between 1 and 10 in map bin "b".
expr = MapRemoveByKeyRange(None, 1, 10 MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKeyRelIndexRange**(*ctx*, *key*, *index*, *count*, *bin*)

Create an expression that removes map items nearest to key and greater by index with a count limit.

__init__(*ctx*, *key*, *index*, *count*, *bin*)

Create an expression that removes map items nearest to key and greater by index with a count limit.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **key** (*TypeKey*) – Key value or expression for key to start removing from.
- **index** (*TypeIndex*) – Index integer or integer expression.
- **count** (*TypeCount*) – Integer expression for how many elements to remove.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns [Map expression.

Example:

```
# Remove 3 elements with keys greater than "key1" from map bin "b".
expr = MapRemoveByKeyRelIndexRange(None, "key1", 1, 3, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKeyRelIndexRangeToEnd**(*ctx*, *key*, *index*, *bin*)

Create an expression that removes map items nearest to key and greater by index.

__init__(*ctx*, *key*, *index*, *bin*)

Create an expression that removes map items nearest to key and greater by index.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **key** (*TypeKey*) – Key value or expression for key to start removing from.
- **index** (*TypeIndex*) – Index integer or integer expression.
- **bin** (*TypeBinName*) – Map bin or map expression.

Returns Map expression.

Example:

```
# Map bin "b" has {"key1": 1, "key2": 2, "key3": 3, "key4": 4}.
# Remove each element where the key has greater index than "key1".
expr = MapRemoveByKeyRelIndexRangeToEnd(None, "key1", 1, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByRank**(*ctx*, *rank*, *bin*)

Create an expression that removes map item identified by rank.

__init__(*ctx*, *rank*, *bin*)

Create an expression that removes map item identified by rank.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Map expression.

Example:

```
# Remove smallest value in map bin "b".
expr = MapRemoveByRank(None, 0, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByRankRange**(*ctx*, *rank*, *count*, *bin*)

Create an expression that removes "count" map items starting at specified rank.

__init__(*ctx*, *rank*, *count*, *bin*)
Create an expression that removes "count" map items starting at specified rank.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to start removing at.
- **count** (*TypeCount*) – Count integer or integer expression of elements to remove.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Map expression.

Example:

```
# Remove the 3 smallest items from map bin "b".
expr = MapRemoveByRankRange(None, 0, 3, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByRankRangeToEnd**(*ctx*, *rank*, *bin*)
Create an expression that removes map items starting at specified rank to the last ranked item.

__init__(*ctx*, *rank*, *bin*)
Create an expression that removes map items starting at specified rank to the last ranked item.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to start removing at.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Map expression.

Example:

```
# Remove the 2 largest elements from map bin "b".
expr = MapRemoveByRankRangeToEnd(None, -2, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValue**(*ctx*, *value*, *bin*)
Create an expression that removes map items identified by value.

__init__(*ctx*, *value*, *bin*)
Create an expression that removes map items identified by value.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **value** (*TypeValue*) – Value or value expression to remove.
- **bin** (*TypeBinName*) – Bin name or map expression.

Returns Map expression.

Example:

```
# Remove {"key1": 1} from map bin "b".
expr = MapRemoveByValue(None, 1, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValueList**(*ctx*, *values*, *bin*)
Create an expression that removes map items identified by values.

__init__(*ctx*, *values*, *bin*)
Create an expression that removes map items identified by values.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **values** (*TypeListValue*) – List of values or list expression.
- **bin** (*TypeBinName*) – Bin name or map expression.

Returns Map expression.

Example:

```
# Remove elements with values 1, 2, 3 from map bin "b".
expr = MapRemoveByValueList(None, [1, 2, 3], MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValueRange**(*ctx*, *begin*, *end*, *bin*)
Create an expression that removes map items identified by value range (begin inclusive, end exclusive). If begin is nil, the range is less than end. If end is aerospike.CDTInfinite(), the range is greater than equal to begin.

__init__(*ctx*, *begin*, *end*, *bin*)
Create an expression that removes map items identified by value range (begin inclusive, end exclusive). If begin is nil, the range is less than end. If end is aerospike.CDTInfinite(), the range is greater than equal to begin.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **begin** (*TypeValue*) – Begin value or value expression for range.
- **end** (*TypeValue*) – End value or value expression for range.
- **bin** (*TypeBinName*) – Bin name or map expression.

Returns Map expression.

Example:

```
# Remove list of items with values >= 3 and < 7 from map bin "b".
expr = MapRemoveByValueRange(None, 3, 7, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValueRelRankRange**(*ctx*, *value*, *rank*, *count*, *bin*)
Create an expression that removes map items nearest to value and greater by relative rank with a count limit.

__init__(*ctx*, *value*, *rank*, *count*, *bin*)
Create an expression that removes map items nearest to value and greater by relative rank with a count limit.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **value** (*TypeValue*) – Value or value expression to start removing from.
- **rank** (*TypeRank*) – Integer or integer expression of rank.
- **count** (*TypeCount*) – Integer count or integer expression for how many elements to remove.

- **bin** (*TypeBinName*) – Bin name or map expression.

Returns Map expression.

Example:

```
# Remove the next 4 elements larger than 3 from map bin "b".
expr = MapRemoveByValueRelRankRangeToEnd(None, 3, 1, 4, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValueRelRankRangeToEnd**(*ctx*, *value*, *rank*, *bin*)

Create an expression that removes map items nearest to value and greater by relative rank.

__init__(*ctx*, *value*, *rank*, *bin*)

Create an expression that removes map items nearest to value and greater by relative rank.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **value** (*TypeValue*) – Value or value expression to start removing from.
- **rank** (*TypeRank*) – Integer or integer expression of rank.
- **bin** (*TypeBinName*) – Bin name or map expression.

Returns Map expression.

Example:

```
# Remove all elements with values larger than 3 from map bin "b".
expr = MapRemoveByValueRelRankRangeToEnd(None, 3, 1, MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapSize**(*ctx*, *bin*)

Create an expression that returns map size.

__init__(*ctx*, *bin*)

Create an expression that returns map size.

Parameters

- **ctx** (*TypeCDT*) – Optional context path for nested CDT.
- **bin** (*TypeBinName*) – Map bin name or map expression.

Returns Integer expression.

Example:

```
#Take the size of map bin "b".
expr = MapSize(None, MapBin("b")).compile()
```

aerospike_helpers.expressions.bit module

Bitwise expressions contain bit read and modify expressions.

Example:

```
import aerospike_helpers.expressions as exp
# Let blob bin "c" == bytearray([3] * 5).
# Count set bits starting at 3rd byte in bin "c" to get count of 6.
expr = exp.BitCount(16, 8 * 3, exp.BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitAdd**(*policy*, *bit_offset*, *bit_size*, *value*, *action*, *bin*)

Create an expression that performs a bit_add operation.

__init__(*policy*, *bit_offset*, *bit_size*, *value*, *action*, *bin*)

Create an expression that performs a bit_add operation. Note: integers are stored big-endian.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **value** (*int*) – Integer value or expression for value to add.
- **action** (*int*) – An aerospike bit overflow action.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# Bit add the second byte of bin "c" to get bytearray([1, 2, 1, 1, 1])
expr = BitAdd(None, 8, 8, 1, aerospike.BIT_OVERFLOW_FAIL).compile()
```

class aerospike_helpers.expressions.bitwise.**BitAnd**(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_and operation.

__init__(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_and operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **value** (*TypeBitValue*) – Bytes value or blob expression containing bytes to use in operation.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# bitwise and `0` with the first byte of blob bin c so that the returned value␣
↪is bytearray([0, 5, 5, 5, 5]).
expr = BitAnd(None, 0, 8, bytearray([0]), BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitCount**(*bit_offset*, *bit_size*, *bin*)

Create an expression that performs a bit_count operation.

__init__(*bit_offset*, *bit_size*, *bin*)

Create an expression that performs a bit_count operation.

Parameters

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to count.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Blob, bit_size bits rounded up to the nearest byte size.

Example:

```
# Let blob bin "c" == bytearray([3] * 5).
# Count set bits starting at 3rd byte in bin "c" to get count of 6.
expr = BitCount(16, 8 * 3, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitGet**(*bit_offset*, *bit_size*, *bin*)

Create an expression that performs a bit_get operation.

__init__(*bit_offset*, *bit_size*, *bin*)

Create an expression that performs a bit_get operation.

Parameters

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to get.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Blob, bit_size bits rounded up to the nearest byte size.

Example:

```
# Let blob bin "c" == bytearray([1, 2, 3, 4, 5).
# Get 2 from bin "c".
expr = BitGet(8, 8, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitGetInt**(*bit_offset*, *bit_size*, *sign*, *bin*)

Create an expression that performs a bit_get_int operation.

__init__(*bit_offset*, *bit_size*, *sign*, *bin*)

Create an expression that performs a bit_get_int operation.

Parameters

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to get.
- **bool** (*sign*) – True for signed, False for unsigned.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Integer expression.

Example:

```
# Let blob bin "c" == bytearray([1, 2, 3, 4, 5).
# Get 2 as an integer from bin "c".
expr = BitGetInt(8, 8, True, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitInsert**(*policy*, *byte_offset*, *value*, *bin*)

Create an expression that performs a bit_insert operation.

__init__(*policy*, *byte_offset*, *value*, *bin*)

Create an expression that performs a bit_insert operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **byte_offset** (*int*) – Integer byte index of where to insert the value.
- **value** (*TypeBitValue*) – A bytes value or blob value expression to insert.
- **bin** (*TypeBinName*) – Blob bin name or blob value expression.

Returns Resulting blob containing the inserted bytes.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# Insert 3 so that returned value is bytearray([1, 3, 1, 1, 1, 1]).
expr = BitInsert(None, 1, bytearray([3]), BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitLeftScan**(*bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_lscan operation.

__init__(*bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_lscan operation.

Parameters

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to read.
- **bool** (*value*) – Bit value to check for.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Index of the left most bit starting from bit_offset set to value. Returns -1 if not found.

Example:

```
# Let blob bin "c" == bytearray([3] * 5).
# Scan the first byte of bin "c" for the first bit set to 1. (should get 6)
expr = BitLeftScan(0, 8, True, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitLeftShift**(*policy*, *bit_offset*, *bit_size*, *shift*, *bin*)

Create an expression that performs a bit_lshift operation.

__init__(*policy*, *bit_offset*, *bit_size*, *shift*, *bin*)

Create an expression that performs a bit_lshift operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.

- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **shift** (*int*) – Number of bits to shift by.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# Bit left shift the first byte of bin "c" to get bytearray([8, 1, 1, 1, 1]).
expr = BitLeftShift(None, 0, 8, 3, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitNot**(*policy*, *bit_offset*, *bit_size*, *bin*)

Create an expression that performs a bit_not operation.

__init__(*policy*, *bit_offset*, *bit_size*, *bin*)

Create an expression that performs a bit_not operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([255] * 5).
# bitwise, not, all of "c" to get bytearray([254] * 5).
expr = BitNot(None, 0, 40, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitOr**(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_or operation.

__init__(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_or operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **value** (*TypeBitValue*) – Bytes value or blob expression containing bytes to use in operation.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# bitwise Or `8` with the first byte of blob bin c so that the returned value␣
↪is bytearray([9, 1, 1, 1, 1]).
expr = BitOr(None, 0, 8, bytearray([8]), BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitRemove**(*policy*, *byte_offset*, *byte_size*, *bin*)

Create an expression that performs a bit_remove operation.

__init__(*policy*, *byte_offset*, *byte_size*, *bin*)

Create an expression that performs a bit_remove operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **byte_offset** (*int*) – Byte index of where to start removing from.
- **byte_size** (*int*) – Number of bytes to remove.
- **bin** (*TypeBinName*) – Blob bin name or blob value expression.

Returns Resulting blob containing the remaining bytes.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# Remove 1 element so that the returned value is bytearray([1] * 4).
expr = BitRemove(None, 1, 1, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitResize**(*policy*, *byte_size*, *flags*, *bin*)

Create an expression that performs a bit_resize operation.

__init__(*policy*, *byte_size*, *flags*, *bin*)

Create an expression that performs a bit_resize operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **byte_size** (*int*) – Number of bytes the resulting blob should occupy.
- **flags** (*int*) – One or a combination of bit resize flags.
- **bin** (*TypeBinName*) – Blob bin name or blob value expression.

Returns Blob value expression of resized blob bin.

Example:

```
# Blob bin "c" == bytearray([1] * 5).
# Resize blob bin "c" from the front so that the returned value is␣
↪bytearray([0] * 5 + [1] * 5).
expr = BitResize(None, 10, aerospike.BIT_RESIZE_FROM_FRONT, BlobBin("c")).
↪compile()
```

class aerospike_helpers.expressions.bitwise.**BitRightScan**(*bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_rscan operation.

__init__(*bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_rscan operation.

Parameters

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to read.
- **bool** (*value*) – Bit value to check for.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Index of the right most bit starting from bit_offset set to value. Returns -1 if not found.

Example:

```
# Let blob bin "c" == bytearray([3] * 5).
# Scan the first byte of bin "c" for the right most bit set to 1. (should get␣
↪7)
expr = BitRightScan(0, 8, True, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitRightShift**(*policy*, *bit_offset*, *bit_size*, *shift*, *bin*)

Create an expression that performs a bit_rshift operation.

__init__(*policy*, *bit_offset*, *bit_size*, *shift*, *bin*)

Create an expression that performs a bit_rshift operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **shift** (*int*) – Number of bits to shift by.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([8] * 5).
# Bit left shift the first byte of bin "c" to get bytearray([4, 8, 8, 8, 8]).
expr = BitRightShift(None, 0, 8, 1, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitSet**(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_set operation.

__init__(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_set operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start overwriting.
- **bit_size** (*int*) – Number of bits to overwrite.
- **value** (*TypeBitValue*) – Bytes value or blob expression containing bytes to write.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob expression with the bits overwritten.

Example:

```
# Let blob bin "c" == bytearray([0] * 5).
# Set bit at offset 7 with size 1 bits to 1 to make the returned value␣
↪bytearray([1, 0, 0, 0, 0]).
expr = BitSet(None, 7, 1, bytearray([255]), BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitSetInt**(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_set_int operation. Note: integers are stored big-endian.

__init__(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_set_int operation. Note: integers are stored big-endian.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start writing.
- **bit_size** (*int*) – Number of bits to overwrite.
- **value** (*int*) – Integer value or integer expression containing value to write.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob expression with the bits overwritten.

Example:

```
# Let blob bin "c" == bytearray([0] * 5).
# Set bit at offset 7 with size 1 bytes to 1 to make the returned value␣
↪bytearray([1, 0, 0, 0, 0]).
expr = BitSetInt(None, 7, 1, 1, BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitSubtract**(*policy*, *bit_offset*, *bit_size*, *value*, *action*, *bin*)

Create an expression that performs a bit_subtract operation. Note: integers are stored big-endian.

__init__(*policy*, *bit_offset*, *bit_size*, *value*, *action*, *bin*)

Create an expression that performs a bit_subtract operation. Note: integers are stored big-endian.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **value** (*int*) – Integer value or expression for value to add.
- **action** (*int*) – An aerospike bit overflow action.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# Bit subtract the second byte of bin "c" to get bytearray([1, 0, 1, 1, 1])
expr = BitSubtract(None, 8, 8, 1, aerospike.BIT_OVERFLOW_FAIL).compile()
```

class aerospike_helpers.expressions.bitwise.**BitXor**(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)

Create an expression that performs a bit_xor operation.

__init__(*policy*, *bit_offset*, *bit_size*, *value*, *bin*)
Create an expression that performs a bit_xor operation.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike bit policy.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **value** (*TypeBitValue*) – Bytes value or blob expression containing bytes to use in operation.
- **bin** (*TypeBinName*) – Blob bin name or blob expression.

Returns Resulting blob with the bits operated on.

Example:

```
# Let blob bin "c" == bytearray([1] * 5).
# bitwise Xor `1` with the first byte of blob bin c so that the returned value␣
↪is bytearray([0, 1, 1, 1, 1]).
expr = BitXor(None, 0, 8, bytearray([1]), BlobBin("c")).compile()
```

aerospike_helpers.expressions.hllmodule

Hyper Log Log expressions contain hll read and modify expressions.

Example:

```
import aerospike_helpers.expressions as exp
# Get count from HLL bin "d".
expr = exp.HLLGetCount(exp.HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLAdd**(*policy*, *list*, *index_bit_count*, *mh_bit_count*, *bin*)
Create an expression that performs an hll_add.

__init__(*policy*, *list*, *index_bit_count*, *mh_bit_count*, *bin*)
Create an expression that performs an hll_add.

Parameters

- **policy** (*TypePolicy*) – An optional aerospike HLL policy.
- **list** (*TypeListValue*) – A list or list expression of elements to add to the HLL.
- **index_bit_count** (*int*) – Number of index bits. Must be between 4 and 16 inclusive.
- **mh_bit_count** (*int*) – Number of min hash bits. Must be between 4 and 51 inclusive.
- **bin** (*TypeBinName*) – A hll bin name or bin expression to apply this function to.

Returns Returns the resulting hll bin after adding elements from list.

Example:

```
# Let HLL bin "d" have the following elements, ['key1', 'key2', 'key3'], index_
↪bits 8, mh_bits 8.
# Add ['key4', 'key5', 'key6'] so that the returned value is ['key1', 'key2', 'key3',
↪'key4', 'key5', 'key6']
expr = HLLAdd(None, ['key4', 'key5', 'key6'], 8, 8, HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLDescribe**(*bin*)
Create an expression that performs an as_operations_hll_describe.

__init__(*bin*)
Create an expression that performs an as_operations_hll_describe.

Parameters **bin** (*TypeBinName*) – A hll bin name or bin expression to read from.

Returns List bin, a list containing the index_bit_count and minhash_bit_count.

Example:

```
# Get description of HLL bin "d".
expr = HLLDescribe(HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLGetCount**(*bin*)
Create an expression that performs an as_operations_hll_get_count.

__init__(*bin*)
Create an expression that performs an as_operations_hll_get_count.

Parameters **bin** (*TypeBinName*) – A hll bin name or bin expression to read from.

Returns Integer bin, the estimated number of unique elements in an HLL.

Example:

```
# Get count from HLL bin "d".
expr = HLLGetCount(HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLGetIntersectCount**(*values*, *bin*)
Create an expression that performs an as_operations_hll_get_inersect_count.

__init__(*values*, *bin*)
Create an expression that performs an as_operations_hll_get_inersect_count.

Parameters

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to intersect with bin.
- **bin** (*TypeBinName*) – A hll bin name or bin expression to read from.

Returns Integer bin, estimated number of elements in the set intersection.

Example:

```
# Let HLLBin "d" contain keys ['key%s' % str(i) for i in range(10000)].
# Let values be a list containing one HLL object with keys ['key%s' % str(i) for␣
↪i in range(5000, 15000)].
# Find the count of keys in the intersection of HLL bin "d" and all HLLs in␣
↪values. (Should be around 5000)
expr = HLLGetIntersectCount(values, HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLGetSimilarity**(*values*, *bin*)
Create an expression that performs an as_operations_hll_get_similarity.

__init__(*values*, *bin*)
Create an expression that performs an as_operations_hll_get_similarity.

Parameters

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to calculate similarity with.
- **bin** (*TypeBinName*) – A hll bin name or bin expression to read from.

Returns Float bin, stimated similarity between 0.0 and 1.0.

Example:

```
# Let HLLBin "d" contain keys ['key%s' % str(i) for i in range(10000)].
# Let values be a list containing one HLL object with keys ['key%s' % str(i) for
↪i in range(5000, 15000)].
# Find the similarity the HLL in values to HLL bin "d". (Should be around 0.33)
# Note that similarity is defined as intersect(A, B, ...) / union(A, B, ...).
expr = HLLGetSimilarity(values, HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLGetUnion**(*values*, *bin*)

Create an expression that performs an hll_get_union.

__init__(*values*, *bin*)

Create an expression that performs an hll_get_union.

Parameters

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to union with bin.
- **bin** (*TypeBinName*) – A hll bin name or bin expression to read from.

Returns HLL bin representing the set union.

Example:

```
# Let HLLBin "d" contain keys ['key%s' % str(i) for i in range(10000)].
# Let values be a list containing HLL objects retrieved from the aerospike
↪database.
# Find the union of HLL bin "d" and all HLLs in values.
expr = HLLGetUnion(values, HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLGetUnionCount**(*values*, *bin*)

Create an expression that performs an as_operations_hll_get_union_count.

__init__(*values*, *bin*)

Create an expression that performs an as_operations_hll_get_union_count.

Parameters

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to union with bin.
- **bin** (*TypeBinName*) – A hll bin name or bin expression to read from.

Returns Integer bin, estimated number of elements in the set union.

Example:

```
# Let HLLBin "d" contain keys ['key%s' % str(i) for i in range(10000)].
# Let values be a list containing one HLL object with keys ['key%s' % str(i) for
↪i in range(5000, 15000)].
# Find the count of keys in the union of HLL bin "d" and all HLLs in values.
↪(Should be around 15000)
expr = HLLGetUnionCount(values, HLLBin("d")).compile()
```

class aerospike_helpers.expressions.hll.**HLLInit**(*policy*, *index_bit_count*, *mh_bit_count*, *bin*)
Create an expression that performs an hll_init.

__init__(*policy*, *index_bit_count*, *mh_bit_count*, *bin*)
Creates a new HLL or resets an existing HLL. If index_bit_count and mh_bit_count are None, an existing HLL bin will be reset but retain its configuration. If 1 of index_bit_count or mh_bit_count are set, an existing HLL bin will set that config and retain its current value for the unset config. If the HLL bin does not exist, index_bit_count is required to create it, mh_bit_count is optional.

Parameters

- **policy** (*TypePolicy*) – An optional dictionary of *hll policy options*.
- **index_bit_count** (*int*) – Number of index bits. Must be between 4 and 16 inclusive.
- **mh_bit_count** (*int*) – Number of min hash bits. Must be between 4 and 51 inclusive.
- **bin** (*TypeBinName*) – A hll bin name or bin expression to apply this function to.

Returns Returns the resulting hll.

Example:

```
# Create an HLL with 12 index bits and 24 min hash bits.
expr = HLLInit(None, 12, 24, HLLBin("my_hll"))
```

class aerospike_helpers.expressions.hll.**HLLMayContain**(*list*, *bin*)
Create an expression that checks if the HLL bin contains any keys in list.

__init__(*list*, *bin*)
Create an expression that checks if the HLL bin contains any keys in list.

Parameters

- **list** (*TypeListValue*) – A list expression of keys to check if the HLL may contain them.
- **bin** (*TypeBinName*) – Integer bin, a hll bin name or bin expression to read from.

Returns 1 if bin contains any key in list, 0 otherwise.

Example:

```
# Check if HLL bin "d" contains any of the keys in `list`.
expr = HLLMayContain(["key1", "key2", "key3"], HLLBin("d")).compile()
```

aerospike_helpers.expressions.arithmeticmodule

Arithmetic expressions provide arithmetic operator support for Aerospike expressions.

Example:

```
import aerospike_helpers.expressions as exp
# Add integer bin "a" to integer bin "b" and see if the result is > 20.
expr = exp.GT(exp.Add(exp.IntBin("a"), exp.IntBin("b")), 20).compile()
```

class aerospike_helpers.expressions.arithmetic.**Abs**(*value*)
Create an absoloute value operator expression.

__init__(*value*)

Create operator that returns absolute value of a number. All arguments must resolve to integer or float. Requires server version 5.6.0+.

Abs is also available via operator overloading using the bultin abs() function and any subclass of _BaseExpr. See the second example.

Parameters **value** (*TypeNumber*) – Float or integer expression or value to take absolute value of.

Returns (number value)

Example:

```
# For int bin "a", abs("a") == 1
expr = Eq(Abs(IntBin("a")), 1).compile()

# Using operator overloading
expr = Eq(abs(IntBin("a")), 1).compile()
```

class aerospike_helpers.expressions.arithmetic.**Add**(**args*)

Create an add, (+) expression.

__init__(**args*)

Create an add, (+) expression. All arguments must be the same type (integer or float). Requires server version 5.6.0+.

Add is also available via operator overloading using + and any subclass of _BaseExpr. See the second example.

Parameters ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values to be added together.

Returns (integer or float value).

Example:

```
# Integer bin "a" + "b" == 11
expr = Eq(Add(IntBin("a"), IntBin("b")), 11).compile()

# Using operator overloading.
expr = Eq(IntBin("a") + IntBin("b"), 11).compile()
```

class aerospike_helpers.expressions.arithmetic.**Ceil**(*value*)

Create a ceiling operator expression.

__init__(*value*)

Create ceil expression that rounds a floating point number up to the closest integer value. Requires server version 5.6.0+.

Ceil is also available via operator overloading using the math.ceil() function and any subclass of _BaseExpr. See the second example.

Parameters **value** (*TypeFloat*) – Float expression or value to take ceiling of.

Returns (float value)

Example:

```
# Ceil(2.25) == 3.0
expr = Eq(Ceil(2.25), 3.0).compile()

# Using operator overloading
expr = Eq(math.ceil(2.25), 3.0).compile()
```

class aerospike_helpers.expressions.arithmetic.**Div**(**args*)

Create a divide, (/) expression.

__init__(**args*)

Create "divide" (/) operator that applies to a variable number of expressions. If there is only one argument, returns the reciprocal for that argument. Otherwise, return the first argument divided by the product of the rest. All arguments must resolve to the same type (integer or float). Requires server version 5.6.0+.

Div is also available via operator overloading using / and any subclass of _BaseExpr. See the second example.

Floor div is also avaliable via // but must be used with floats.

Parameters ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values to be divided.

Returns (integer or float value)

Example:

```
# Integer bin "a" / "b" / "c" >= 11
expr = GE(Div(IntBin("a"), IntBin("b"), IntBin("c")), 11).compile()

# Using operator overloading.
expr = GE(IntBin("a") / IntBin("b") / IntBin("c"), 11).compile()

# Float bin "a" // "b" // "c" >= 11.0
expr = GE(FloatBin("a") // FloatBin("b") // FloatBin("c"), 11.0).compile()
```

class aerospike_helpers.expressions.arithmetic.**Floor**(*value*)

Create a floor operator expression.

__init__(*value*)

Create floor expression that rounds a floating point number down to the closest integer value. Requires server version 5.6.0+.

Floor is also available via operator overloading using the math.floor() function and any subclass of _BaseExpr. See the second example.

Parameters **value** (*TypeFloat*) – Float expression or value to take floor of.

Returns (float value)

Example:

```
# Floor(2.25) == 2.0
expr = Eq(Floor(2.25), 2.0).compile()

# Using operator overloading
expr = Eq(math.floor(2.25), 2.0).compile()
```

class aerospike_helpers.expressions.arithmetic.**Log**(*num*, *base*)
Create a log operator expression.

__init__(*num*, *base*)

Create "log" operator for logarithm of "num" with base "base". All arguments must resolve to floats. Requires server version 5.6.0+.

Parameters

- **num** (*TypeFloat*) – Float expression or value number.
- **base** (*TypeFloat*) – Float expression or value base.

Returns (float value)

Example:

```
# For float bin "a", log("a", 2.0) == 16.0
expr = Eq(Log(FloatBin("a"), 2.0), 16.0).compile()
```

class aerospike_helpers.expressions.arithmetic.**Max**(**args*)
Create expression that returns the maximum value in a variable number of expressions.

__init__(**args*)

Create expression that returns the maximum value in a variable number of expressions. All arguments must be the same type (integer or float). Requires server version 5.6.0+.

Parameters ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values from which to find the maximum value.

Returns (integer or float value).

Example:

```
# for integer bins a, b, c, max(a, b, c) > 100
expr = GT(Max(IntBin("a"), IntBin("b"), IntBin("c")), 100).compile()
```

class aerospike_helpers.expressions.arithmetic.**Min**(**args*)
Create expression that returns the minimum value in a variable number of expressions.

__init__(**args*)

Create expression that returns the minimum value in a variable number of expressions. All arguments must be the same type (integer or float). Requires server version 5.6.0+.

Parameters ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values from which to find the minimum value.

Returns (integer or float value).

Example:

```
# for integer bins a, b, c, min(a, b, c) > 0
expr = GT(Min(IntBin("a"), IntBin("b"), IntBin("c")), 0).compile()
```

class aerospike_helpers.expressions.arithmetic.**Mod**(*numerator*, *denominator*)

Create a modulo, (%) expression.

__init__(*numerator*, *denominator*)

Create "modulo" (%) operator that determines the remainder of "numerator" divided by "denominator". All arguments must resolve to integers. Requires server version 5.6.0+.

Mod is also available via operator overloading using % and any subclass of _BaseExpr. See the second example.

Parameters

- **numerator** (*TypeInteger*) – Integer expression or value numerator.
- **denominator** (*TypeInteger*) – Integer expression or value denominator.

Returns (integer value)

Example:

```
# For int bin "a" % 10 == 0
expr = Eq(Mod(IntBin("a"), 10), 0).compile()

# Using operator overloading.
expr = Eq(IntBin("a") % 10, 0).compile()
```

class aerospike_helpers.expressions.arithmetic.**Mul**(**args*)

Create a multiply, (*) expression.

__init__(**args*)

Create "multiply" (*) operator that applies to a variable number of expressions. Return the product of all arguments. If only one argument is supplied, return that argument. All arguments must resolve to the same type (integer or float). Requires server version 5.6.0+.

Mul is also available via operator overloading using * and any subclass of _BaseExpr. See the second example.

Parameters ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values to be multiplied.

Returns (integer or float value)

Example:

```
# Integer bin "a" * "b" >= 11
expr = GE(Mul(IntBin("a"), IntBin("b")), 11).compile()
```

(continues on next page)

(continued from previous page)

```
# Using operator overloading.
expr = GE(IntBin("a") * IntBin("b"), 11).compile()
```

class aerospike_helpers.expressions.arithmetic.**Pow**(*base*, *exponent*)
Create a pow, (**) expression.

__init__(*base*, *exponent*)

Create "pow" operator that raises a "base" to the "exponent" power. All arguments must resolve to floats. Requires server version 5.6.0+.

Pow is also available via operator overloading using ** and any subclass of _BaseExpr. See the second example.

Parameters

- **base** (*TypeFloat*) – Float expression or value base.
- **exponent** (*TypeFloat*) – Float expression or value exponent.

Returns (float value)

Example:

```
# Float bin "a" ** 2.0 == 16.0
expr = Eq(Pow(FloatBin("a"), 2.0), 16.0).compile()

# Using operator overloading.
expr = Eq(FloatBin("a") ** 2.0, 16.0).compile()
```

class aerospike_helpers.expressions.arithmetic.**Sub**(**args*)
Create a subtraction, (-) expression.

__init__(**args*)

Create "subtract" (-) operator that applies to a variable number of expressions. If only one argument is provided, return the negation of that argument. Otherwise, return the sum of the 2nd to Nth argument subtracted from the 1st argument. All arguments must resolve to the same type (integer or float). Requires server version 5.6.0+.

Sub is also available via operator overloading using - and any subclass of _BaseExpr. See the second example.

Parameters ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values to be subtracted.

Returns (integer or float value)

Example:

```
# Integer bin "a" - "b" == 11
expr = Eq(Sub(IntBin("a"), IntBin("b")), 11).compile()

# Using operator overloading.
expr = Eq(IntBin("a") - IntBin("b"), 11).compile()
```

class aerospike_helpers.expressions.arithmetic.**ToFloat**(*value*)
Create expression that converts an integer to a float.

__init__(*value*)

Create expression that converts an integer to a float. Requires server version 5.6.0+.

Parameters **value** (*TypeInteger*) – Integer expression or value to convert to float.

Returns (float value)

Example:

```
#For int bin "a", float(IntBin("a")) == 2
expr = Eq(ToFloat(IntBin("a")), 2).compile()
```

class aerospike_helpers.expressions.arithmetic.**ToInt**(*value*)
Create expression that converts a float to an integer.

__init__(*value*)

Create expression that converts a float to an integer. Requires server version 5.6.0+.

Parameters **value** (*TypeFloat*) – Float expression or value to convert to int.

Returns (integer value)

Example:

```
#For float bin "a", int(FloatBin("a")) == 2
expr = Eq(ToInt(FloatBin("a")), 2).compile()
```

aerospike_helpers.expressions.bitwise_operatorsmodule

Bitwise operator expressions provide support for bitwise operators like & and >> in Aerospike expressions.

Example:

```
import aerospike_helpers.expressions as exp
# Let int bin "a" == 0xAAAA.
# Use bitwise and to apply a mask 0xFF00 to 0xAAAA and check for 0xAA00.
expr = exp.Eq(exp.IntAnd(IntBin("a"), 0xFF00), 0xAA00).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntAnd**(**exprs*)
Create integer "and" (&) operator expression that is applied to two or more integers.

__init__(**exprs*)
Create integer "and" (&) operator expression that is applied to two or more integers. All arguments must resolve to integers. Requires server version 5.6.0+.

Parameters ***exprs** (*TypeInteger*) – A variable amount of integer expressions or values to be bitwise ANDed.

Returns (integer value)

Example:

```
# for int bin "a", a & 0xff == 0x11
expr = Eq(IntAnd(IntBin("a"), 0xff), 0x11).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntArithmeticRightShift**(*value*, *shift*)
Create integer "arithmetic right shift" (>>) operator.

__init__(*value*, *shift*)
Create integer "arithmetic right shift" (>>) operator. Requires server version 5.6.0+.

Parameters

- **value** (*TypeInteger*) – An integer value or expression to be right shifted.
- **shift** (*TypeInteger*) – An integer value or expression for number of bits to right shift *value* by.

Returns (integer value)

Example:

```
# for int bin "a", a >> 8 > 0xff
expr = GT(IntArithmeticRightShift(IntBin("a"), 8), 0xff).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntCount**(*value*)
Create expression that returns count of integer bits that are set to 1.

__init__(*value*)
Create expression that returns count of integer bits that are set to 1. Requires server version 5.6.0+.

Parameters **value** (*TypeInteger*) – An integer value or expression to have bits counted.

Returns (integer value)

Example:

```
# for int bin "a", count(a) == 4
expr = Eq(IntCount(IntBin("a")), 4).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntLeftScan**(*value*, *search*)
Create expression that scans integer bits from left (most significant bit) to right (least significant bit).

__init__(*value*, *search*)
Create expression that scans integer bits from left (most significant bit) to right (least significant bit), looking for a search bit value. When the search value is found, the index of that bit (where the most significant bit is index 0) is returned. If "search" is true, the scan will search for the bit value 1. If "search" is false it will search for bit value 0. Requires server version 5.6.0+.

Parameters

- **value** (*TypeInteger*) – An integer value or expression to be scanned.
- **search** (*TypeBool*) – A bool expression or value to scan for.

Returns (integer value)

Example:

```
# for int bin "a", lscan(a, True) == 4
expr = GT(lscan(IntBin("a"), True), 4).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntLeftShift**(*value*, *shift*)
Create integer "left shift" (<<) operator.

__init__(*value*, *shift*)
Create integer "left shift" (<<) operator. Requires server version 5.6.0+.

Parameters

- **value** (*TypeInteger*) – An integer value or expression to be left shifted.
- **shift** (*TypeInteger*) – An integer value or expression for number of bits to left shift *value* by.

Returns (integer value)

Example:

```
# for int bin "a", a << 8 > 0xff
expr = GT(IntLeftShift(IntBin("a"), 8), 0xff).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntNot**(*expr*)
Create integer "not" (~) operator.

__init__(*expr*)
Create integer "not" (~) operator. Requires server version 5.6.0+.

Parameters **expr** (*TypeInteger*) – An integer value or expression to be bitwise negated.

Returns (integer value)

Example:

```
# for int bin "a", ~ a == 7
expr = Eq(IntNot(IntBin("a")), 7).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntOr**(**exprs*)
Create integer "or" (|) operator expression that is applied to two or more integers.

__init__(**exprs*)
Create integer "or" (|) operator expression that is applied to two or more integers. All arguments must resolve to integers. Requires server version 5.6.0+.

Parameters ***exprs** (*TypeInteger*) – A variable amount of integer expressions or values to be bitwise ORed.

Returns (integer value)

Example:

```
# for int bin "a", a | 0x10 not == 0
expr = NE(IntOr(IntBin("a"), 0x10), 0).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntRightScan**(*value*, *search*)
Create expression that scans integer bits from right (least significant bit) to left (most significant bit).

__init__(*value*, *search*)
Create expression that scans integer bits from right (least significant bit) to left (most significant bit), looking for a search bit value. When the search value is found, the index of that bit (where the most significant bit is index 0) is returned. If "search" is true, the scan will search for the bit value 1. If "search" is false it will search for bit value 0. Requires server version 5.6.0+.

Parameters

- **value** (*TypeInteger*) – An integer value or expression to be scanned.
- **search** (*TypeBool*) – A bool expression or value to scan for.

Returns (integer value)

Example:

```
# for int bin "a", rscan(a, True) == 4
expr = GT(IntRightScan(IntBin("a"), True), 4).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntRightShift**(*value*, *shift*)

Create integer "logical right shift" (>>>) operator.

__init__(*value*, *shift*)

Create integer "logical right shift" (>>>) operator. Requires server version 5.6.0+.

Parameters

- **value** (*TypeInteger*) – An integer value or expression to be right shifted.
- **shift** (*TypeInteger*) – An integer value or expression for number of bits to right shift *value* by.

Returns (integer value)

Example:

```
# for int bin "a", a >>> 8 > 0xff
expr = GT(IntRightShift(IntBin("a"), 8), 0xff).compile()
```

class aerospike_helpers.expressions.bitwise_operators.**IntXOr**(**exprs*)

Create integer "xor" (^) operator that is applied to two or more integers.

__init__(**exprs*)

Create integer "xor" (^) operator that is applied to two or more integers. All arguments must resolve to integers. Requires server version 5.6.0+.

Parameters ***exprs** (*TypeInteger*) – A variable amount of integer expressions or values to be bitwise XORed.

Returns (integer value)

Example:

```
# for int bin "a", "b", a ^ b == 16
expr = Eq(IntXOr(IntBin("a"), IntBin("b")), 16).compile()
```

aerospike_helpers.expressions.resourcesmodule

Resources used by all expressions.

class aerospike_helpers.expressions.resources.**ResultType**

Flags used to indicate expression value_type.

BOOLEAN = 1

INTEGER = 2

STRING = 3

LIST = 4

MAP = 5

BLOB = 6

FLOAT = 7

GEOJSON = 8

HLL = 9

1.8.1.3 aerospike_helpers.cdt_ctx module

Note: Requires server version >= 4.6.0

Helper functions to generate complex data type context (cdt_ctx) objects for use with operations on nested CDTs (list, map, etc).

Example:

```
from __future__ import print_function
import aerospike
from aerospike import exception as ex
from aerospike_helpers import cdt_ctx
from aerospike_helpers.operations import map_operations
from aerospike_helpers.operations import list_operations
import sys

# Configure the client.
config = {"hosts": [("127.0.0.1", 3000)]}

# Create a client and connect it to the cluster.
try:
    client = aerospike.client(config).connect()
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

key = ("test", "demo", "foo")
nested_list = [{"name": "John", "id": 100}, {"name": "Bill", "id": 200}]
nested_list_bin_name = "nested_list"

# Write the record.
try:
    client.put(key, {nested_list_bin_name: nested_list})
except ex.RecordError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))

# EXAMPLE 1: read a value from the map nested at list index 1.
try:
    ctx = [cdt_ctx.cdt_ctx_list_index(1)]

    ops = [
        map_operations.map_get_by_key(
            nested_list_bin_name, "id", aerospike.MAP_RETURN_VALUE, ctx
        )
    ]
```

(continues on next page)

(continued from previous page)

```
    _, _, result = client.operate(key, ops)
    print("EXAMPLE 1, id is: ", result)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# EXAMPLE 2: write a new nested map at list index 2 and get the value at its 'name' key.
# NOTE: The map is appened to the list, then the value is read using the ctx.
try:
    new_map = {"name": "Cindy", "id": 300}

    ctx = [cdt_ctx.cdt_ctx_list_index(2)]

    ops = [
        list_operations.list_append(nested_list_bin_name, new_map),
        map_operations.map_get_by_key(
            nested_list_bin_name, "name", aerospike.MAP_RETURN_VALUE, ctx
        ),
    ]

    _, _, result = client.operate(key, ops)
    print("EXAMPLE 2, name is: ", result)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# Cleanup and close the connection to the Aerospike cluster.
client.remove(key)
client.close()

"""
EXPECTED OUTPUT:
EXAMPLE 1, id is:  {'nested_list': 200}
EXAMPLE 2, name is:  {'nested_list': 'Cindy'}
"""
```

aerospike_helpers.cdt_ctx.**cdt_ctx_list_index**(*index*)

Creates a nested cdt_ctx object for use with list or map operations.

The cdt_ctx object is initialized to lookup an object in a list by index. If the index is negative, the lookup starts backwards from the end of the list. If it is out of bounds, a parameter error will be returned.

Parameters **index** (*int*) – The index to look for in the list.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_list_index_create**(*index*, *order=<MagicMock id='140444911463440'>*, *pad=False*)

Creates a nested cdt_ctx object for use with list or map operations.

Create a list with the given sort order at the given index.

Parameters

- **key** (*object*) – The index to create the list at.

- **order** (*int*) – The sort order to create the list with. One of *list sort orders*. Default == aerospike.LIST_UNORDERED
- **pad** (*Bool*) – If index is out of bounds and pad is True, the list will be created at index and empty list elements inserted behind it. Pad is only compatible with unordered lists.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_list_rank**(*rank*)
Creates a nested cdt_ctx object for use with list or map operations.

The cdt_ctx object is initialized to lookup an object in a list by rank. If the rank is negative, the lookup starts backwards from the largest rank value.

Parameters **rank** (*int*) – The rank to look for in the list.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_list_value**(*value*)
Creates a nested cdt_ctx object for use with list or map operations.

The cdt_ctx object is initialized to lookup an object in a list by value.

Parameters **value** (*object*) – The value to look for in the list.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_map_index**(*index*)
Creates a nested cdt_ctx object for use with list or map operations.

The cdt_ctx object is initialized to lookup an object in a map by index. If the index is negative, the lookup starts backwards from the end of the map. If it is out of bounds, a parameter error will be returned.

Parameters **index** (*int*) – The index to look for in the map.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_map_key**(*key*)
Creates a nested cdt_ctx object for use with list or map operations.

The cdt_ctx object is initialized to lookup an object in a map by key.

Parameters **key** (*object*) – The key to look for in the map.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_map_key_create**(*key*, *order=<MagicMock id='140444911470992'>*)
Creates a nested cdt_ctx object for use with list or map operations.

Create a map with the given sort order at the given key.

Parameters

- **key** (*object*) – The key to create the map at.
- **order** (*int*) – The sort order to create the map with. One of the aerospike *map sort orders*. Default == aerospike.MAP_UNORDERED

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_map_rank**(*rank*)
Creates a nested cdt_ctx object for use with list or map operations.

The cdt_ctx object is initialized to lookup an object in a map by index. If the rank is negative, the lookup starts backwards from the largest rank value.

Parameters **rank** (*int*) – The rank to look for in the map.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

aerospike_helpers.cdt_ctx.**cdt_ctx_map_value**(*value*)

Creates a nested cdt_ctx object for use with list or map operations.

The cdt_ctx object is initialized to lookup an object in a map by value.

Parameters **value** (*object*) – The value to look for in the map.

Returns A cdt_ctx object, a list of these is usable with list and map operations.

1.9 GeoJSON Class — GeoJSON

1.9.1 GeoJSON

class aerospike.**GeoJSON**

Starting with version 3.7.0, the Aerospike server supports storing GeoJSON data. A Geo2DSphere index can be built on a bin which contains GeoJSON data, enabling queries for the points contained within given shapes using *geo_within_geojson_region()* and *geo_within_radius()*, and for the regions which contain a point using *geo_contains_geojson_point()* and *geo_contains_point()*.

On the client side, wrapping geospatial data in an instance of the *aerospike.GeoJSON* class enables serialization of the data into the correct type during write operation, such as *put()*. On reading a record from the server, bins with geospatial data it will be deserialized into a *GeoJSON* instance.

See also:

Geospatial Index and Query.

```
from __future__ import print_function
import aerospike
from aerospike import GeoJSON

config = { 'hosts': [ ('127.0.0.1', 3000)]}
client = aerospike.client(config).connect()
client.index_geo2dsphere_create('test', 'pads', 'loc', 'pads_loc_geo')
# Create GeoJSON point using WGS84 coordinates.
latitude = 28.608389
longitude = -80.604333
loc = GeoJSON({'type': "Point",
               'coordinates': [longitude, latitude]})
print(loc)
# Alternatively create the GeoJSON point from a string
loc = aerospike.geojson('{"type": "Point", "coordinates": [-80.604333, 28.608389]}')

# Create a user record.
bins = {'pad_id': 1,
        'loc': loc}

# Store the record.
client.put(('test', 'pads', 'launchpad1'), bins)

# Read the record.
(k, m, b) = client.get(('test', 'pads', 'launchpad1'))
```

(continues on next page)

(continued from previous page)

```
print(b)
client.close()
```

class GeoJSON([*geo_data*])
Optionally initializes an object with a GeoJSON `str` or a `dict` of geospatial data.

wrap(*geo_data*)
Sets the geospatial data of the *GeoJSON* wrapper class.

Parameters **geo_data** (*dict*) – a `dict` representing the geospatial data.

unwrap() → dict of geospatial data
Gets the geospatial data contained in the *GeoJSON* class.

Returns a `dict` representing the geospatial data.

loads(*raw_geo*)
Sets the geospatial data of the *GeoJSON* wrapper class from a GeoJSON string.

Parameters **raw_geo** (*str*) – a GeoJSON string representation.

dumps() → a GeoJSON string
Gets the geospatial data contained in the *GeoJSON* class as a GeoJSON string.

Returns a GeoJSON `str` representing the geospatial data.

New in version 1.0.53.

1.10 Data_Mapping — Python Data Mappings

How Python types map to server types

Note: By default, the *aerospike.Client* maps the supported types `int`, `bool`, `str`, `float`, `bytearray`, `list`, `dict` to matching aerospike server types (int, string, double, blob, list, map). When an unsupported type is encountered, the module uses cPickle to serialize and deserialize the data, storing it into a blob of type 'Python' (AS_BYTES_PYTHON).

The functions *set_serializer()* and *set_deserializer()* allow for user-defined functions to handle serialization, instead. The user provided function will be run instead of cPickle. The serialized data is stored as type (AS_BYTES_BLOB). This type allows the storage of binary data readable by Aerospike Clients in other languages. The *serialization* config param of *aerospike.client()* registers an instance-level pair of functions that handle serialization.

Unless a user specified serializer has been provided, all other types will be stored as Python specific bytes. Python specific bytes may not be readable by Aerospike Clients for other languages.

Warning: Aerospike is introducing a new boolean data type in server version 5.6. In order to support cross client compatibility and rolling upgrades, Python client version 6.x comes with a new client config, send_bool_as. Send_bool_as configures how the client writes Python booleans and allows for opting into using the new boolean type. It is important to consider how other clients connected to the Aerospike database write booleans in order to maintain cross client compatibility. If a client reads and writes booleans as integers then the Python client should too, if they work with the same data. Send_bool_as can be set so the client writes Python booleans as

AS_BYTES_PYTHON, integer, or the new server boolean type. All versions before 6.x wrote Python booleans as AS_BYTES_PYTHON.

The following table shows which Python types map directly to Aerospike server types.

Note: `aerospike.KeyOrderedDict` is a special case. Like dict, KeyOrderedDict maps to the aerospike map data type. However, the map will be sorted in key order before being sent to the server, see *Map Order*.

Python Type	Server type
int	integer
bool	depends on send_bool_as
str	string
unicode	string
float	double
dict	map
aerospike.KeyOrderedDict	key ordered map
list	list
bytearray	blob
aerospike.GeoJSON	GeoJSON

It is possible to nest these datatypes. For example a list may contain a dictionary, or a dictionary may contain a list as a value.

Note: Unless a user specified serializer has been provided, all other types will be stored as Python specific bytes. Python specific bytes may not be readable by Aerospike Clients for other languages.

1.11 KeyOrderedDict Class — `KeyOrderedDict`

1.11.1 `KeyOrderedDict`

The KeyOrderedDict class is a dictionary that directly maps to a key ordered map on the Aerospike server. This assists in matching key ordered maps through various read operations. See the example snippet below.

```
import aerospike
from aerospike_helpers.operations import map_operations as mop
from aerospike_helpers.operations import list_operations as lop
import aerospike_helpers.cdt_ctx as ctx
from aerospike import KeyOrderedDict

config = { 'hosts': [ ("localhost", 3000), ] }
client = aerospike.client(config).connect()
map_policy={'map_order': aerospike.MAP_KEY_VALUE_ORDERED}

key = ("test", "demo", 100)
client.put(key, {'map_list': []})
```

(continues on next page)

(continued from previous page)

```
map_ctx1 = ctx.cdt_ctx_list_index(0)
map_ctx2 = ctx.cdt_ctx_list_index(1)
map_ctx3 = ctx.cdt_ctx_list_index(2)

my_dict1 = {'a': 1, 'b': 2, 'c': 3}
my_dict2 = {'d': 4, 'e': 5, 'f': 6}
my_dict3 = {'g': 7, 'h': 8, 'i': 9}

ops = [
        lop.list_append_items('map_list', [my_dict1, my_dict2, my_dict3]),
        mop.map_set_policy('map_list', map_policy, [map_ctx1]),
        mop.map_set_policy('map_list', map_policy, [map_ctx2]),
        mop.map_set_policy('map_list', map_policy, [map_ctx3])
    ]
client.operate(key, ops)

_, _, res = client.get(key)
print(res)

element = KeyOrderedDict({'f': 6, 'e': 5, 'd': 4}) # this will match my_dict2␣
↪because it will be converted to key ordered.

ops = [
        lop.list_get_by_value('map_list', element, aerospike.LIST_RETURN_COUNT)
    ]
_, _, res = client.operate(key, ops)
print(res)

client.remove(key)
client.close()

# EXPECTED OUTPUT:
# {'map_list': [{'a': 1, 'b': 2, 'c': 3}, {'d': 4, 'e': 5, 'f': 6}, {'g': 7, 'h': 8, 'i': 9}
↪]}
# {'map_list': 1}
```

KeyOrderedDict inherits from `dict` and has no extra functionality. The only difference is its mapping to a key ordered map.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

aerospike (*64-bit Linux and OS X*), 3
aerospike.exception (*64-bit Linux and OS X*), 111
aerospike.predexp (*64-bit Linux and OS X*), 100
aerospike.predicates (*64-bit Linux and OS X*), 94
aerospike_helpers, 117
aerospike_helpers.cdt_ctx, 223
aerospike_helpers.expressions.arithmetic, 213
aerospike_helpers.expressions.base, 163
aerospike_helpers.expressions.bitwise, 203
aerospike_helpers.expressions.bitwise_operators, 219
aerospike_helpers.expressions.hll, 210
aerospike_helpers.expressions.list, 174
aerospike_helpers.expressions.map, 187
aerospike_helpers.expressions.resources, 222
aerospike_helpers.operations.bitwise_operations, 145
aerospike_helpers.operations.expression_operations, 158
aerospike_helpers.operations.hll_operations, 153
aerospike_helpers.operations.list_operations, 118
aerospike_helpers.operations.map_operations, 131
aerospike_helpers.operations.operations, 117

INDEX

Symbols

__init__() (*aerospike_helpers.expressions.arithmetic.Abs method*), 213
__init__() (*aerospike_helpers.expressions.arithmetic.Add method*), 214
__init__() (*aerospike_helpers.expressions.arithmetic.Ceil method*), 214
__init__() (*aerospike_helpers.expressions.arithmetic.Div method*), 215
__init__() (*aerospike_helpers.expressions.arithmetic.Floor method*), 215
__init__() (*aerospike_helpers.expressions.arithmetic.Log method*), 216
__init__() (*aerospike_helpers.expressions.arithmetic.Max method*), 216
__init__() (*aerospike_helpers.expressions.arithmetic.Min method*), 216
__init__() (*aerospike_helpers.expressions.arithmetic.Mod method*), 217
__init__() (*aerospike_helpers.expressions.arithmetic.Mul method*), 217
__init__() (*aerospike_helpers.expressions.arithmetic.Pow method*), 218
__init__() (*aerospike_helpers.expressions.arithmetic.Sub method*), 218
__init__() (*aerospike_helpers.expressions.arithmetic.ToFloat method*), 219
__init__() (*aerospike_helpers.expressions.arithmetic.ToInt method*), 219
__init__() (*aerospike_helpers.expressions.base.And method*), 163
__init__() (*aerospike_helpers.expressions.base.BinExists method*), 163
__init__() (*aerospike_helpers.expressions.base.BinType method*), 163
__init__() (*aerospike_helpers.expressions.base.BlobBin method*), 163
__init__() (*aerospike_helpers.expressions.base.BoolBin method*), 164
__init__() (*aerospike_helpers.expressions.base.CmpGeo method*), 164
__init__() (*aerospike_helpers.expressions.base.CmpRegex method*), 164
__init__() (*aerospike_helpers.expressions.base.Cond method*), 165
__init__() (*aerospike_helpers.expressions.base.Def method*), 165
__init__() (*aerospike_helpers.expressions.base.DeviceSize method*), 166
__init__() (*aerospike_helpers.expressions.base.DigestMod method*), 166
__init__() (*aerospike_helpers.expressions.base.Eq method*), 166
__init__() (*aerospike_helpers.expressions.base.Exclusive method*), 166
__init__() (*aerospike_helpers.expressions.base.FloatBin method*), 167
__init__() (*aerospike_helpers.expressions.base.GE method*), 167
__init__() (*aerospike_helpers.expressions.base.GT method*), 167
__init__() (*aerospike_helpers.expressions.base.GeoBin method*), 168
__init__() (*aerospike_helpers.expressions.base.HLLBin method*), 168
__init__() (*aerospike_helpers.expressions.base.IntBin method*), 168
__init__() (*aerospike_helpers.expressions.base.IsTombstone method*), 168
__init__() (*aerospike_helpers.expressions.base.KeyBlob method*), 169
__init__() (*aerospike_helpers.expressions.base.KeyExists method*), 169
__init__() (*aerospike_helpers.expressions.base.KeyInt method*), 169
__init__() (*aerospike_helpers.expressions.base.KeyStr method*), 169
__init__() (*aerospike_helpers.expressions.base.LE method*), 170
__init__() (*aerospike_helpers.expressions.base.LT method*), 170
__init__() (*aerospike_helpers.expressions.base.LastUpdateTime method*), 170
__init__() (*aerospike_helpers.expressions.base.Let*

method), 170
__init__() (*aerospike_helpers.expressions.base.ListBin method*), 171
__init__() (*aerospike_helpers.expressions.base.MapBin method*), 171
__init__() (*aerospike_helpers.expressions.base.NE method*), 171
__init__() (*aerospike_helpers.expressions.base.Not method*), 172
__init__() (*aerospike_helpers.expressions.base.Or method*), 172
__init__() (*aerospike_helpers.expressions.base.SetName method*), 172
__init__() (*aerospike_helpers.expressions.base.SinceUpdateTime method*), 172
__init__() (*aerospike_helpers.expressions.base.StrBin method*), 173
__init__() (*aerospike_helpers.expressions.base.TTL method*), 173
__init__() (*aerospike_helpers.expressions.base.Unknown method*), 173
__init__() (*aerospike_helpers.expressions.base.Var method*), 174
__init__() (*aerospike_helpers.expressions.base.VoidTime method*), 174
__init__() (*aerospike_helpers.expressions.bitwise.BitAdd method*), 203
__init__() (*aerospike_helpers.expressions.bitwise.BitAnd method*), 203
__init__() (*aerospike_helpers.expressions.bitwise.BitCount method*), 204
__init__() (*aerospike_helpers.expressions.bitwise.BitGet method*), 204
__init__() (*aerospike_helpers.expressions.bitwise.BitGetInt method*), 204
__init__() (*aerospike_helpers.expressions.bitwise.BitInsert method*), 205
__init__() (*aerospike_helpers.expressions.bitwise.BitLeftScan method*), 205
__init__() (*aerospike_helpers.expressions.bitwise.BitLeftShift method*), 205
__init__() (*aerospike_helpers.expressions.bitwise.BitNot method*), 206
__init__() (*aerospike_helpers.expressions.bitwise.BitOr method*), 206
__init__() (*aerospike_helpers.expressions.bitwise.BitRemove method*), 207
__init__() (*aerospike_helpers.expressions.bitwise.BitResize method*), 207
__init__() (*aerospike_helpers.expressions.bitwise.BitRightScan method*), 207
__init__() (*aerospike_helpers.expressions.bitwise.BitRightShift method*), 208
__init__() (*aerospike_helpers.expressions.bitwise.BitSet method*), 208
__init__() (*aerospike_helpers.expressions.bitwise.BitSetInt method*), 209
__init__() (*aerospike_helpers.expressions.bitwise.BitSubtract method*), 209
__init__() (*aerospike_helpers.expressions.bitwise.BitXor method*), 209
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntAnd method*), 219
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntArithmet method*), 220
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntCount method*), 220
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntLeftScan method*), 220
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntLeftShift method*), 220
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntNot method*), 221
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntOr method*), 221
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntRightSca method*), 221
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntRightShi method*), 222
__init__() (*aerospike_helpers.expressions.bitwise_operators.IntXOr method*), 222
__init__() (*aerospike_helpers.expressions.hll.HLLAdd method*), 210
__init__() (*aerospike_helpers.expressions.hll.HLLDescribe method*), 211
__init__() (*aerospike_helpers.expressions.hll.HLLGetCount method*), 211
__init__() (*aerospike_helpers.expressions.hll.HLLGetIntersectCount method*), 211
__init__() (*aerospike_helpers.expressions.hll.HLLGetSimilarity method*), 211
__init__() (*aerospike_helpers.expressions.hll.HLLGetUnion method*), 212
__init__() (*aerospike_helpers.expressions.hll.HLLGetUnionCount method*), 212
__init__() (*aerospike_helpers.expressions.hll.HLLInit method*), 213
__init__() (*aerospike_helpers.expressions.hll.HLLMayContain method*), 213
__init__() (*aerospike_helpers.expressions.list.ListAppend method*), 174
__init__() (*aerospike_helpers.expressions.list.ListAppendItems method*), 175
__init__() (*aerospike_helpers.expressions.list.ListClear method*), 175
__init__() (*aerospike_helpers.expressions.list.ListGetByIndex method*), 176
__init__() (*aerospike_helpers.expressions.list.ListGetByIndexRange*

method), 176
__init__() (aerospike_helpers.expressions.list.ListGetByIndexRange method), 177
__init__() (aerospike_helpers.expressions.list.ListGetByRank method), 177
__init__() (aerospike_helpers.expressions.list.ListGetByRankRange method), 178
__init__() (aerospike_helpers.expressions.list.ListGetByRankRangeToEnd method), 178
__init__() (aerospike_helpers.expressions.list.ListGetByValue method), 178
__init__() (aerospike_helpers.expressions.list.ListGetByValueList method), 179
__init__() (aerospike_helpers.expressions.list.ListGetByValueRange method), 179
__init__() (aerospike_helpers.expressions.list.ListGetByValueRelRankRange method), 180
__init__() (aerospike_helpers.expressions.list.ListGetByValueRelRankRangeToEnd method), 180
__init__() (aerospike_helpers.expressions.list.ListIncrement method), 181
__init__() (aerospike_helpers.expressions.list.ListInsert method), 181
__init__() (aerospike_helpers.expressions.list.ListInsertItems method), 182
__init__() (aerospike_helpers.expressions.list.ListRemoveByIndex method), 182
__init__() (aerospike_helpers.expressions.list.ListRemoveByIndexRange method), 183
__init__() (aerospike_helpers.expressions.list.ListRemoveByIndexRangeToEnd method), 183
__init__() (aerospike_helpers.expressions.list.ListRemoveByRank method), 183
__init__() (aerospike_helpers.expressions.list.ListRemoveByRankRange method), 184
__init__() (aerospike_helpers.expressions.list.ListRemoveByRankRangeToEnd method), 184
__init__() (aerospike_helpers.expressions.list.ListRemoveByValue method), 184
__init__() (aerospike_helpers.expressions.list.ListRemoveByValueList method), 185
__init__() (aerospike_helpers.expressions.list.ListRemoveByValueRange method), 185
__init__() (aerospike_helpers.expressions.list.ListRemoveByValueRelRankRange method), 185
__init__() (aerospike_helpers.expressions.list.ListRemoveByValueRelRankRangeToEnd method), 186
__init__() (aerospike_helpers.expressions.list.ListSet method), 186
__init__() (aerospike_helpers.expressions.list.ListSize method), 187
__init__() (aerospike_helpers.expressions.list.ListSort method), 187
__init__() (aerospike_helpers.expressions.map.MapClear method), 187
__init__() (aerospike_helpers.expressions.map.MapGetByIndex method), 188
__init__() (aerospike_helpers.expressions.map.MapGetByIndexRange method), 188
__init__() (aerospike_helpers.expressions.map.MapGetByIndexRangeToEnd method), 189
__init__() (aerospike_helpers.expressions.map.MapGetByKey method), 189
__init__() (aerospike_helpers.expressions.map.MapGetByKeyList method), 190
__init__() (aerospike_helpers.expressions.map.MapGetByKeyRange method), 190
__init__() (aerospike_helpers.expressions.map.MapGetByKeyRelIndexRange method), 191
__init__() (aerospike_helpers.expressions.map.MapGetByKeyRelIndexRangeToEnd method), 191
__init__() (aerospike_helpers.expressions.map.MapGetByRank method), 192
__init__() (aerospike_helpers.expressions.map.MapGetByRankRange method), 192
__init__() (aerospike_helpers.expressions.map.MapGetByRankRangeToEnd method), 192
__init__() (aerospike_helpers.expressions.map.MapGetByValue method), 193
__init__() (aerospike_helpers.expressions.map.MapGetByValueList method), 193
__init__() (aerospike_helpers.expressions.map.MapGetByValueRange method), 194
__init__() (aerospike_helpers.expressions.map.MapGetByValueRelRankRangeToEnd method), 194
__init__() (aerospike_helpers.expressions.map.MapGetByValueRelRankRange method), 195
__init__() (aerospike_helpers.expressions.map.MapIncrement method), 195
__init__() (aerospike_helpers.expressions.map.MapPut method), 196
__init__() (aerospike_helpers.expressions.map.MapPutItems method), 196
__init__() (aerospike_helpers.expressions.map.MapRemoveByIndex method), 196
__init__() (aerospike_helpers.expressions.map.MapRemoveByIndexRange method), 197
__init__() (aerospike_helpers.expressions.map.MapRemoveByIndexRangeToEnd method), 197
__init__() (aerospike_helpers.expressions.map.MapRemoveByKey method), 197
__init__() (aerospike_helpers.expressions.map.MapRemoveByKeyList method), 198
__init__() (aerospike_helpers.expressions.map.MapRemoveByKeyRange method), 198
__init__() (aerospike_helpers.expressions.map.MapRemoveByKeyRelIndexRange method), 198
__init__() (aerospike_helpers.expressions.map.MapRemoveByKeyRelIndexRangeToEnd

method), 199
__init__() (*aerospike_helpers.expressions.map.MapRemoveByKeyList method*), 199
__init__() (*aerospike_helpers.expressions.map.MapRemoveByRankRange method*), 199
__init__() (*aerospike_helpers.expressions.map.MapRemoveByRankRangeToEnd method*), 200
__init__() (*aerospike_helpers.expressions.map.MapRemoveByValue method*), 200
__init__() (*aerospike_helpers.expressions.map.MapRemoveByValueList method*), 201
__init__() (*aerospike_helpers.expressions.map.MapRemoveByValueRange method*), 201
__init__() (*aerospike_helpers.expressions.map.MapRemoveByValueRelativeRankRange method*), 201
__init__() (*aerospike_helpers.expressions.map.MapRemoveByValueRelativeRankRangeToEnd method*), 202
__init__() (*aerospike_helpers.expressions.map.MapSize method*), 202
__version__ (*in module aerospike*), 23

A

Abs (*class in aerospike_helpers.expressions.arithmetic*), 213
Add (*class in aerospike_helpers.expressions.arithmetic*), 214
add_ops() (*aerospike.Query method*), 89
add_ops() (*aerospike.Scan method*), 76
Admin Operations, 60
admin_change_password() (*aerospike.Client method*), 62
admin_create_role() (*aerospike.Client method*), 60
admin_create_user() (*aerospike.Client method*), 62
admin_drop_role() (*aerospike.Client method*), 61
admin_drop_user() (*aerospike.Client method*), 62
admin_get_role() (*aerospike.Client method*), 61
admin_get_roles() (*aerospike.Client method*), 62
admin_grant_privileges() (*aerospike.Client method*), 61
admin_grant_roles() (*aerospike.Client method*), 63
admin_query_role() (*aerospike.Client method*), 62
admin_query_roles() (*aerospike.Client method*), 62
admin_query_user() (*aerospike.Client method*), 63
admin_query_users() (*aerospike.Client method*), 63
admin_revoke_privileges() (*aerospike.Client method*), 61
admin_revoke_roles() (*aerospike.Client method*), 63
admin_set_password() (*aerospike.Client method*), 63
admin_set_quotas() (*aerospike.Client method*), 61
admin_set_whitelist() (*aerospike.Client method*), 61
AdminError, 114
aerospike
    module, 3
aerospike.exception
    module, 111
aerospike.LIST_SORT_DEFAULT (*in module aerospike*), 18
aerospike.LIST_SORT_DROP_DUPLICATES (*in module aerospike*), 18
aerospike.predexp
    module, 100
aerospike.predicates
    module, 94
aerospike_helpers
    module, 117
aerospike_helpers.cdt_ctx
    module, 223
aerospike_helpers.expressions.arithmetic
    module, 213
aerospike_helpers.expressions.base
    module, 163
aerospike_helpers.expressions.bitwise
    module, 203
aerospike_helpers.expressions.bitwise_operators
    module, 219
aerospike_helpers.expressions.hll
    module, 210
aerospike_helpers.expressions.list
    module, 174
aerospike_helpers.expressions.map
    module, 187
aerospike_helpers.expressions.resources
    module, 222
aerospike_helpers.operations.bitwise_operations
    module, 145
aerospike_helpers.operations.expression_operations
    module, 158
aerospike_helpers.operations.hll_operations
    module, 153
aerospike_helpers.operations.list_operations
    module, 118
aerospike_helpers.operations.map_operations
    module, 131
aerospike_helpers.operations.operations
    module, 117
AerospikeError, 112
AlwaysForbidden, 113
And (*class in aerospike_helpers.expressions.base*), 163
append() (*aerospike.Client method*), 38
append() (*in module aerospike_helpers.operations.operations*), 117
apply() (*aerospike.Client method*), 48
apply() (*aerospike.Query method*), 88
apply() (*aerospike.Scan method*), 76
AS_BOOL (*in module aerospike*), 17
AS_BYTES_BLOB (*in module aerospike*), 22
AS_BYTES_CSHARP (*in module aerospike*), 22

AS_BYTES_DOUBLE (*in module aerospike*), 22
AS_BYTES_ERLANG (*in module aerospike*), 22
AS_BYTES_GEOJSON (*in module aerospike*), 22
AS_BYTES_HLL (*in module aerospike*), 22
AS_BYTES_INTEGER (*in module aerospike*), 22
AS_BYTES_JAVA (*in module aerospike*), 22
AS_BYTES_LIST (*in module aerospike*), 22
AS_BYTES_MAP (*in module aerospike*), 22
AS_BYTES_PHP (*in module aerospike*), 22
AS_BYTES_PYTHON (*in module aerospike*), 22
AS_BYTES_RUBY (*in module aerospike*), 22
AS_BYTES_STRING (*in module aerospike*), 22
AS_BYTES_TYPE_MAX (*in module aerospike*), 22
AS_BYTES_UNDEF (*in module aerospike*), 22
AUTH_EXTERNAL (*in module aerospike*), 16
AUTH_EXTERNAL_INSECURE (*in module aerospike*), 16
AUTH_INTERNAL (*in module aerospike*), 16

B

Batch Operations, 34
between() (*in module aerospike.predicates*), 94
bin (*aerospike.exception.RecordError attribute*), 113
BinExists (*class in aerospike_helpers.expressions.base*), 163
BinIncompatibleType, 113
BinNameError, 113
BinType (*class in aerospike_helpers.expressions.base*), 163
bit_add() (*in module aerospike_helpers.operations.bitwise_operations*), 148
bit_and() (*in module aerospike_helpers.operations.bitwise_operations*), 148
bit_count() (*in module aerospike_helpers.operations.bitwise_operations*), 148
bit_get() (*in module aerospike_helpers.operations.bitwise_operations*), 149
bit_get_int() (*in module aerospike_helpers.operations.bitwise_operations*), 149
bit_insert() (*in module aerospike_helpers.operations.bitwise_operations*), 149
bit_lscan() (*in module aerospike_helpers.operations.bitwise_operations*), 149
bit_lshift() (*in module aerospike_helpers.operations.bitwise_operations*), 150
bit_not() (*in module aerospike_helpers.operations.bitwise_operations*), 150
bit_or() (*in module aerospike_helpers.operations.bitwise_operations*), 150
BIT_OVERFLOW_FAIL (*in module aerospike*), 21
BIT_OVERFLOW_SATURATE (*in module aerospike*), 21
BIT_OVERFLOW_WRAP (*in module aerospike*), 21
bit_remove() (*in module aerospike_helpers.operations.bitwise_operations*), 151
bit_resize() (*in module aerospike_helpers.operations.bitwise_operations*), 151
BIT_RESIZE_DEFAULT (*in module aerospike*), 20
BIT_RESIZE_FROM_FRONT (*in module aerospike*), 20
BIT_RESIZE_GROW_ONLY (*in module aerospike*), 20
BIT_RESIZE_SHRINK_ONLY (*in module aerospike*), 20
bit_rscan() (*in module aerospike_helpers.operations.bitwise_operations*), 151
bit_rshift() (*in module aerospike_helpers.operations.bitwise_operations*), 152
bit_set() (*in module aerospike_helpers.operations.bitwise_operations*), 152
bit_subtract() (*in module aerospike_helpers.operations.bitwise_operations*), 152
BIT_WRITE_CREATE_ONLY (*in module aerospike*), 20
BIT_WRITE_DEFAULT (*in module aerospike*), 20
BIT_WRITE_NO_FAIL (*in module aerospike*), 20
BIT_WRITE_PARTIAL (*in module aerospike*), 20
BIT_WRITE_UPDATE_ONLY (*in module aerospike*), 20
bit_xor() (*in module aerospike_helpers.operations.bitwise_operations*), 153
BitAdd (*class in aerospike_helpers.expressions.bitwise*), 203
BitAnd (*class in aerospike_helpers.expressions.bitwise*), 203
BitCount (*class in aerospike_helpers.expressions.bitwise*), 204
BitGet (*class in aerospike_helpers.expressions.bitwise*), 204
BitGetInt (*class in aerospike_helpers.expressions.bitwise*), 204
BitInsert (*class in aerospike_helpers.expressions.bitwise*), 205
BitLeftScan (*class in aerospike_helpers.expressions.bitwise*), 205
BitLeftShift (*class in aerospike_helpers.expressions.bitwise*), 205
BitNot (*class in aerospike_helpers.expressions.bitwise*), 206

BitOr (*class in aerospike_helpers.expressions.bitwise*), 206
BitRemove (*class in aerospike_helpers.expressions.bitwise*), 207
BitResize (*class in aerospike_helpers.expressions.bitwise*), 207
BitRightScan (*class in aerospike_helpers.expressions.bitwise*), 207
BitRightShift (*class in aerospike_helpers.expressions.bitwise*), 208
BitSet (*class in aerospike_helpers.expressions.bitwise*), 208
BitSetInt (*class in aerospike_helpers.expressions.bitwise*), 209
BitSubtract (*class in aerospike_helpers.expressions.bitwise*), 209
BitXor (*class in aerospike_helpers.expressions.bitwise*), 209
BLOB (*aerospike_helpers.expressions.resources.ResultType attribute*), 222
BlobBin (*class in aerospike_helpers.expressions.base*), 163
BoolBin (*class in aerospike_helpers.expressions.base*), 164
BOOLEAN (*aerospike_helpers.expressions.resources.ResultType attribute*), 222

C

calc_digest() (*in module aerospike*), 9
cdt_ctx_list_index() (*in module aerospike_helpers.cdt_ctx*), 224
cdt_ctx_list_index_create() (*in module aerospike_helpers.cdt_ctx*), 224
cdt_ctx_list_rank() (*in module aerospike_helpers.cdt_ctx*), 225
cdt_ctx_list_value() (*in module aerospike_helpers.cdt_ctx*), 225
cdt_ctx_map_index() (*in module aerospike_helpers.cdt_ctx*), 225
cdt_ctx_map_key() (*in module aerospike_helpers.cdt_ctx*), 225
cdt_ctx_map_key_create() (*in module aerospike_helpers.cdt_ctx*), 225
cdt_ctx_map_rank() (*in module aerospike_helpers.cdt_ctx*), 225
cdt_ctx_map_value() (*in module aerospike_helpers.cdt_ctx*), 226
CDTInfinite() (*in module aerospike*), 8
CDTWildcard() (*in module aerospike*), 8
Ceil (*class in aerospike_helpers.expressions.arithmetic*), 214
Client (*class in aerospike*), 26, 27, 34, 38, 40, 41, 46, 52, 57, 60
client() (*in module aerospike*), 3
ClientError, 112
close() (*aerospike.Client method*), 26
ClusterChangeError, 114
ClusterError, 114
CmpGeo (*class in aerospike_helpers.expressions.base*), 164
CmpRegex (*class in aerospike_helpers.expressions.base*), 164
code (*aerospike.exception.AerospikeError attribute*), 112
Cond (*class in aerospike_helpers.expressions.base*), 165
connect() (*aerospike.Client method*), 26
contains() (*in module aerospike.predicates*), 98

D

Def (*class in aerospike_helpers.expressions.base*), 165
delete() (*in module aerospike_helpers.operations.operations*), 117
DeviceOverload, 113
DeviceSize (*class in aerospike_helpers.expressions.base*), 166
DigestMod (*class in aerospike_helpers.expressions.base*), 166
Div (*class in aerospike_helpers.expressions.arithmetic*), 215
dumps() (*aerospike.GeoJSON method*), 227

E

ElementExistsError, 113
ElementNotFoundError, 113
Eq (*class in aerospike_helpers.expressions.base*), 166
equals() (*in module aerospike.predicates*), 94
Exclusive (*class in aerospike_helpers.expressions.base*), 166
execute_background() (*aerospike.Query method*), 90
execute_background() (*aerospike.Scan method*), 80
exists() (*aerospike.Client method*), 30
exists_many() (*aerospike.Client method*), 36
EXP_READ_DEFAULT (*in module aerospike*), 22
EXP_READ_EVAL_NO_FAIL (*in module aerospike*), 22
EXP_WRITE_ALLOW_DELETE (*in module aerospike*), 21
EXP_WRITE_CREATE_ONLY (*in module aerospike*), 21
EXP_WRITE_DEFAULT (*in module aerospike*), 21
EXP_WRITE_EVAL_NO_FAIL (*in module aerospike*), 21
EXP_WRITE_POLICY_NO_FAIL (*in module aerospike*), 21
EXP_WRITE_UPDATE_ONLY (*in module aerospike*), 21
ExpiredPassword, 114
expression_read() (*in module aerospike_helpers.operations.expression_operations*), 158
expression_write() (*in module aerospike_helpers.operations.expression_operations*), 158

F

file (*aerospike.exception.AerospikeError attribute*), 112
FilteredOut, 112
FLOAT (*aerospike_helpers.expressions.resources.ResultType attribute*), 222
FloatBin (*class in aerospike_helpers.expressions.base*), 167
Floor (*class in aerospike_helpers.expressions.arithmetic*), 215
ForbiddenError, 113
ForbiddenPassword, 114
foreach() (*aerospike.Query method*), 86
foreach() (*aerospike.Scan method*), 79
func (*aerospike.exception.UDFError attribute*), 115

G

GE (*class in aerospike_helpers.expressions.base*), 167
geo_contains_geojson_point() (*in module aerospike.predicates*), 97
geo_contains_point() (*in module aerospike.predicates*), 97
geo_within_geojson_region() (*in module aerospike.predicates*), 95
geo_within_radius() (*in module aerospike.predicates*), 96
GeoBin (*class in aerospike_helpers.expressions.base*), 167
geodata() (*in module aerospike*), 12
GEOJSON (*aerospike_helpers.expressions.resources.ResultType attribute*), 223
GeoJSON (*class in aerospike*), 226
geojson() (*in module aerospike*), 13
GeoJSON.GeoJSON (*class in aerospike*), 227
geojson_bin() (*in module aerospike.predexp*), 102
geojson_contains() (*in module aerospike.predexp*), 111
geojson_value() (*in module aerospike.predexp*), 103
geojson_var() (*in module aerospike.predexp*), 104
geojson_within() (*in module aerospike.predexp*), 110
get() (*aerospike.Client method*), 30
get_key_digest() (*aerospike.Client method*), 33
get_many() (*aerospike.Client method*), 34
get_node_names() (*aerospike.Client method*), 52
get_nodes() (*aerospike.Client method*), 52
GT (*class in aerospike_helpers.expressions.base*), 167

H

HLL (*aerospike_helpers.expressions.resources.ResultType attribute*), 223
hll_add() (*in module aerospike_helpers.operations.hll_operations*), 155
hll_describe() (*in module aerospike_helpers.operations.hll_operations*), 156
hll_fold() (*in module aerospike_helpers.operations.hll_operations*), 156
hll_get_count() (*in module aerospike_helpers.operations.hll_operations*), 156
hll_get_intersect_count() (*in module aerospike_helpers.operations.hll_operations*), 156
hll_get_similarity() (*in module aerospike_helpers.operations.hll_operations*), 156
hll_get_union() (*in module aerospike_helpers.operations.hll_operations*), 156
hll_get_union_count() (*in module aerospike_helpers.operations.hll_operations*), 157
hll_init() (*in module aerospike_helpers.operations.hll_operations*), 157
hll_refresh_count() (*in module aerospike_helpers.operations.hll_operations*), 157
hll_set_union() (*in module aerospike_helpers.operations.hll_operations*), 157
HLL_WRITE_ALLOW_FOLD (*in module aerospike*), 21
HLL_WRITE_CREATE_ONLY (*in module aerospike*), 21
HLL_WRITE_DEFAULT (*in module aerospike*), 21
HLL_WRITE_NO_FAIL (*in module aerospike*), 21
HLL_WRITE_UPDATE_ONLY (*in module aerospike*), 21
HLLAdd (*class in aerospike_helpers.expressions.hll*), 210
HLLBin (*class in aerospike_helpers.expressions.base*), 168
HLLDescribe (*class in aerospike_helpers.expressions.hll*), 210
HLLGetCount (*class in aerospike_helpers.expressions.hll*), 211
HLLGetIntersectCount (*class in aerospike_helpers.expressions.hll*), 211
HLLGetSimilarity (*class in aerospike_helpers.expressions.hll*), 211
HLLGetUnion (*class in aerospike_helpers.expressions.hll*), 212
HLLGetUnionCount (*class in aerospike_helpers.expressions.hll*), 212
HLLInit (*class in aerospike_helpers.expressions.hll*), 212
HLLMayContain (*class in aerospike_helpers.expressions.hll*), 213

I

IllegalState, 114
in_doubt (*aerospike.exception.AerospikeError attribute*), 112
increment() (*aerospike.Client method*), 40
increment() (*in module aerospike_helpers.operations.operations*), 117
Index Operations, 57
INDEX_GEO2DSPHERE (*in module aerospike*), 23
index_geo2dsphere_create() (*aerospike.Client method*), 59
index_integer_create() (*aerospike.Client method*), 57
index_list_create() (*aerospike.Client method*), 57
index_map_keys_create() (*aerospike.Client method*), 58
index_map_values_create() (*aerospike.Client method*), 58
index_name (*aerospike.exception.IndexError attribute*), 114
INDEX_NUMERIC (*in module aerospike*), 23
index_remove() (*aerospike.Client method*), 59
INDEX_STRING (*in module aerospike*), 23
index_string_create() (*aerospike.Client method*), 57
INDEX_TYPE_LIST (*in module aerospike*), 23
INDEX_TYPE_MAPKEYS (*in module aerospike*), 23
INDEX_TYPE_MAPVALUES (*in module aerospike*), 23
IndexError, 114
IndexFoundError, 114
IndexNameMaxCount, 114
IndexNameMaxLen, 114
IndexNotFound, 114
IndexNotReadable, 114
IndexOOM, 114
Info Operations, 51
info() (*aerospike.Client method*), 52
info_all() (*aerospike.Client method*), 53
info_node() (*aerospike.Client method*), 54
info_random_node() (*aerospike.Client method*), 55
info_single_node() (*aerospike.Client method*), 54
IntAnd (*class in aerospike_helpers.expressions.bitwise_operators*), 219
IntArithmeticRightShift (*class in aerospike_helpers.expressions.bitwise_operators*), 220
IntBin (*class in aerospike_helpers.expressions.base*), 168
IntCount (*class in aerospike_helpers.expressions.bitwise_operators*), 220
INTEGER (*aerospike_helpers.expressions.resources.ResultType attribute*), 222
INTEGER (*in module aerospike*), 17
integer_bin() (*in module aerospike.predexp*), 101
integer_equal() (*in module aerospike.predexp*), 108
integer_greater() (*in module aerospike.predexp*), 109
integer_greatereq() (*in module aerospike.predexp*), 109
integer_less() (*in module aerospike.predexp*), 109
integer_lesseq() (*in module aerospike.predexp*), 109
integer_unequal() (*in module aerospike.predexp*), 110
integer_value() (*in module aerospike.predexp*), 103
integer_var() (*in module aerospike.predexp*), 103
IntLeftScan (*class in aerospike_helpers.expressions.bitwise_operators*), 220
IntLeftShift (*class in aerospike_helpers.expressions.bitwise_operators*), 220
IntNot (*class in aerospike_helpers.expressions.bitwise_operators*), 221
IntOr (*class in aerospike_helpers.expressions.bitwise_operators*), 221
IntRightScan (*class in aerospike_helpers.expressions.bitwise_operators*), 221
IntRightShift (*class in aerospike_helpers.expressions.bitwise_operators*), 222
IntXOr (*class in aerospike_helpers.expressions.bitwise_operators*), 222
InvalidCommand, 114
InvalidCredential, 114
InvalidField, 114
InvalidHostError, 112
InvalidPassword, 114
InvalidPrivilege, 115
InvalidRequest, 112
InvalidRole, 115
InvalidUser, 115
is_connected() (*aerospike.Client method*), 26
IsTombstone (*class in aerospike_helpers.expressions.base*), 168

J

job_info() (*aerospike.Client method*), 50
JOB_QUERY (*in module aerospike*), 16
JOB_SCAN (*in module aerospike*), 16
JOB_STATUS_COMPLETED (*in module aerospike*), 16
JOB_STATUS_INPROGRESS (*in module aerospike*), 16
JOB_STATUS_UNDEF (*in module aerospike*), 16

K

key (*aerospike.exception.RecordError attribute*), 113

KeyBlob (*class in aerospike_helpers.expressions.base*), 169
KeyExists (*class in aerospike_helpers.expressions.base*), 169
KeyInt (*class in aerospike_helpers.expressions.base*), 169
KeyStr (*class in aerospike_helpers.expressions.base*), 169

L

LastUpdateTime (*class in aerospike_helpers.expressions.base*), 170
LE (*class in aerospike_helpers.expressions.base*), 169
Let (*class in aerospike_helpers.expressions.base*), 170
line (*aerospike.exception.AerospikeError attribute*), 112
LIST (*aerospike_helpers.expressions.resources.ResultType attribute*), 222
List Operations, 40
list_append() (*in module aerospike_helpers.operations.list_operations*), 118
list_append_items() (*in module aerospike_helpers.operations.list_operations*), 118
list_bin() (*in module aerospike.predexp*), 102
list_clear() (*in module aerospike_helpers.operations.list_operations*), 119
list_get() (*in module aerospike_helpers.operations.list_operations*), 119
list_get_by_index() (*in module aerospike_helpers.operations.list_operations*), 119
list_get_by_index_range() (*in module aerospike_helpers.operations.list_operations*), 120
list_get_by_rank() (*in module aerospike_helpers.operations.list_operations*), 120
list_get_by_rank_range() (*in module aerospike_helpers.operations.list_operations*), 120
list_get_by_value() (*in module aerospike_helpers.operations.list_operations*), 121
list_get_by_value_list() (*in module aerospike_helpers.operations.list_operations*), 121
list_get_by_value_range() (*in module aerospike_helpers.operations.list_operations*), 121
list_get_by_value_rank_range_relative() (*in module aerospike_helpers.operations.list_operations*), 122
list_get_range() (*in module aerospike_helpers.operations.list_operations*), 123
list_increment() (*in module aerospike_helpers.operations.list_operations*), 123
list_insert() (*in module aerospike_helpers.operations.list_operations*), 124
list_insert_items() (*in module aerospike_helpers.operations.list_operations*), 124
list_iterate_and() (*in module aerospike.predexp*), 105
list_iterate_or() (*in module aerospike.predexp*), 104
LIST_ORDERED (*in module aerospike*), 18
list_pop() (*in module aerospike_helpers.operations.list_operations*), 124
list_pop_range() (*in module aerospike_helpers.operations.list_operations*), 124
list_remove() (*in module aerospike_helpers.operations.list_operations*), 125
list_remove_by_index() (*in module aerospike_helpers.operations.list_operations*), 125
list_remove_by_index_range() (*in module aerospike_helpers.operations.list_operations*), 125
list_remove_by_rank() (*in module aerospike_helpers.operations.list_operations*), 126
list_remove_by_rank_range() (*in module aerospike_helpers.operations.list_operations*), 126
list_remove_by_value() (*in module aerospike_helpers.operations.list_operations*), 127
list_remove_by_value_list() (*in module aerospike_helpers.operations.list_operations*), 127
list_remove_by_value_range() (*in module aerospike_helpers.operations.list_operations*), 127
list_remove_by_value_rank_range_relative() (*in module aerospike_helpers.operations.list_operations*), 128
list_remove_range() (*in module aerospike_helpers.operations.list_operations*), 129

LIST_RETURN_COUNT (*in module aerospike*), 18
LIST_RETURN_INDEX (*in module aerospike*), 18
LIST_RETURN_NONE (*in module aerospike*), 18
LIST_RETURN_RANK (*in module aerospike*), 18
LIST_RETURN_REVERSE_INDEX (*in module aerospike*), 18
LIST_RETURN_REVERSE_RANK (*in module aerospike*), 18
LIST_RETURN_VALUE (*in module aerospike*), 18
list_set() (*in module aerospike_helpers.operations.list_operations*), 129
list_set_order() (*in module aerospike_helpers.operations.list_operations*), 130
list_size() (*in module aerospike_helpers.operations.list_operations*), 130
list_sort() (*in module aerospike_helpers.operations.list_operations*), 130
list_trim() (*in module aerospike_helpers.operations.list_operations*), 130
LIST_UNORDERED (*in module aerospike*), 18
LIST_WRITE_ADD_UNIQUE (*in module aerospike*), 17
LIST_WRITE_DEFAULT (*in module aerospike*), 17
LIST_WRITE_INSERT_BOUNDED (*in module aerospike*), 17
LIST_WRITE_NO_FAIL (*in module aerospike*), 17
LIST_WRITE_PARTIAL (*in module aerospike*), 17
ListAppend (*class in aerospike_helpers.expressions.list*), 174
ListAppendItems (*class in aerospike_helpers.expressions.list*), 175
ListBin (*class in aerospike_helpers.expressions.base*), 171
ListClear (*class in aerospike_helpers.expressions.list*), 175
ListGetByIndex (*class in aerospike_helpers.expressions.list*), 176
ListGetByIndexRange (*class in aerospike_helpers.expressions.list*), 176
ListGetByIndexRangeToEnd (*class in aerospike_helpers.expressions.list*), 177
ListGetByRank (*class in aerospike_helpers.expressions.list*), 177
ListGetByRankRange (*class in aerospike_helpers.expressions.list*), 177
ListGetByRankRangeToEnd (*class in aerospike_helpers.expressions.list*), 178
ListGetByValue (*class in aerospike_helpers.expressions.list*), 178
ListGetByValueList (*class in aerospike_helpers.expressions.list*), 179
ListGetByValueRange (*class in aerospike_helpers.expressions.list*), 179
ListGetByValueRelRankRange (*class in aerospike_helpers.expressions.list*), 180
ListGetByValueRelRankRangeToEnd (*class in aerospike_helpers.expressions.list*), 180
ListIncrement (*class in aerospike_helpers.expressions.list*), 181
ListInsert (*class in aerospike_helpers.expressions.list*), 181
ListInsertItems (*class in aerospike_helpers.expressions.list*), 182
ListRemoveByIndex (*class in aerospike_helpers.expressions.list*), 182
ListRemoveByIndexRange (*class in aerospike_helpers.expressions.list*), 183
ListRemoveByIndexRangeToEnd (*class in aerospike_helpers.expressions.list*), 183
ListRemoveByRank (*class in aerospike_helpers.expressions.list*), 183
ListRemoveByRankRange (*class in aerospike_helpers.expressions.list*), 184
ListRemoveByRankRangeToEnd (*class in aerospike_helpers.expressions.list*), 184
ListRemoveByValue (*class in aerospike_helpers.expressions.list*), 184
ListRemoveByValueList (*class in aerospike_helpers.expressions.list*), 185
ListRemoveByValueRange (*class in aerospike_helpers.expressions.list*), 185
ListRemoveByValueRelRankRange (*class in aerospike_helpers.expressions.list*), 185
ListRemoveByValueRelRankToEnd (*class in aerospike_helpers.expressions.list*), 186
ListSet (*class in aerospike_helpers.expressions.list*), 186
ListSize (*class in aerospike_helpers.expressions.list*), 187
ListSort (*class in aerospike_helpers.expressions.list*), 187
loads() (*aerospike.GeoJSON method*), 227
Log (*class in aerospike_helpers.expressions.arithmetic*), 216
LOG_LEVEL_DEBUG (*in module aerospike*), 23
LOG_LEVEL_ERROR (*in module aerospike*), 23
LOG_LEVEL_INFO (*in module aerospike*), 23
LOG_LEVEL_OFF (*in module aerospike*), 23
LOG_LEVEL_TRACE (*in module aerospike*), 23
LOG_LEVEL_WARN (*in module aerospike*), 23
LT (*class in aerospike_helpers.expressions.base*), 170
LuaFileNotFound, 115

M

MAP (*aerospike_helpers.expressions.resources.ResultType*

attribute), 222
Map Operations, 41
map_bin() (*in module aerospike.predexp*), 102
map_clear() (*in module aerospike_helpers.operations.map_operations*), 131
MAP_CREATE_ONLY (*in module aerospike*), 19
map_decrement() (*in module aerospike_helpers.operations.map_operations*), 131
map_get_by_index() (*in module aerospike_helpers.operations.map_operations*), 131
map_get_by_index_range() (*in module aerospike_helpers.operations.map_operations*), 132
map_get_by_key() (*in module aerospike_helpers.operations.map_operations*), 132
map_get_by_key_index_range_relative() (*in module aerospike_helpers.operations.map_operations*), 132
map_get_by_key_list() (*in module aerospike_helpers.operations.map_operations*), 133
map_get_by_key_range() (*in module aerospike_helpers.operations.map_operations*), 134
map_get_by_rank() (*in module aerospike_helpers.operations.map_operations*), 134
map_get_by_rank_range() (*in module aerospike_helpers.operations.map_operations*), 134
map_get_by_value() (*in module aerospike_helpers.operations.map_operations*), 135
map_get_by_value_list() (*in module aerospike_helpers.operations.map_operations*), 135
map_get_by_value_range() (*in module aerospike_helpers.operations.map_operations*), 136
map_get_by_value_rank_range_relative() (*in module aerospike_helpers.operations.map_operations*), 136
map_increment() (*in module aerospike_helpers.operations.map_operations*), 137
MAP_KEY_ORDERED (*in module aerospike*), 19
MAP_KEY_VALUE_ORDERED (*in module aerospike*), 19
map_put() (*in module aerospike_helpers.operations.map_operations*), 137
map_put_items() (*in module aerospike_helpers.operations.map_operations*), 138
map_remove_by_index() (*in module aerospike_helpers.operations.map_operations*), 138
map_remove_by_index_range() (*in module aerospike_helpers.operations.map_operations*), 138
map_remove_by_key() (*in module aerospike_helpers.operations.map_operations*), 139
map_remove_by_key_index_range_relative() (*in module aerospike_helpers.operations.map_operations*), 139
map_remove_by_key_list() (*in module aerospike_helpers.operations.map_operations*), 140
map_remove_by_key_range() (*in module aerospike_helpers.operations.map_operations*), 140
map_remove_by_rank() (*in module aerospike_helpers.operations.map_operations*), 141
map_remove_by_rank_range() (*in module aerospike_helpers.operations.map_operations*), 141
map_remove_by_value() (*in module aerospike_helpers.operations.map_operations*), 142
map_remove_by_value_list() (*in module aerospike_helpers.operations.map_operations*), 142
map_remove_by_value_range() (*in module aerospike_helpers.operations.map_operations*), 142
map_remove_by_value_rank_range_relative() (*in module aerospike_helpers.operations.map_operations*), 143
MAP_RETURN_COUNT (*in module aerospike*), 19
MAP_RETURN_INDEX (*in module aerospike*), 19
MAP_RETURN_KEY (*in module aerospike*), 20
MAP_RETURN_KEY_VALUE (*in module aerospike*), 20
MAP_RETURN_NONE (*in module aerospike*), 19
MAP_RETURN_RANK (*in module aerospike*), 19
MAP_RETURN_REVERSE_INDEX (*in module aerospike*), 19
MAP_RETURN_REVERSE_RANK (*in module aerospike*), 19
MAP_RETURN_VALUE (*in module aerospike*), 20
map_set_policy() (*in module aerospike_helpers.operations.map_operations*), 144
map_size() (*in module aerospike_helpers.operations.map_operations*), 144

MAP_UNORDERED (*in module aerospike*), 19
MAP_UPDATE (*in module aerospike*), 19
MAP_UPDATE_ONLY (*in module aerospike*), 19
MAP_WRITE_FLAGS_CREATE_ONLY (*in module aerospike*), 18
MAP_WRITE_FLAGS_DEFAULT (*in module aerospike*), 18
MAP_WRITE_FLAGS_NO_FAIL (*in module aerospike*), 19
MAP_WRITE_FLAGS_PARTIAL (*in module aerospike*), 19
MAP_WRITE_FLAGS_UPDATE_ONLY (*in module aerospike*), 18
MapBin (*class in aerospike_helpers.expressions.base*), 171
MapClear (*class in aerospike_helpers.expressions.map*), 187
MapGetByIndex (*class in aerospike_helpers.expressions.map*), 188
MapGetByIndexRange (*class in aerospike_helpers.expressions.map*), 188
MapGetByIndexRangeToEnd (*class in aerospike_helpers.expressions.map*), 189
MapGetByKey (*class in aerospike_helpers.expressions.map*), 189
MapGetByKeyList (*class in aerospike_helpers.expressions.map*), 190
MapGetByKeyRange (*class in aerospike_helpers.expressions.map*), 190
MapGetByKeyRelIndexRange (*class in aerospike_helpers.expressions.map*), 190
MapGetByKeyRelIndexRangeToEnd (*class in aerospike_helpers.expressions.map*), 191
MapGetByRank (*class in aerospike_helpers.expressions.map*), 191
MapGetByRankRange (*class in aerospike_helpers.expressions.map*), 192
MapGetByRankRangeToEnd (*class in aerospike_helpers.expressions.map*), 192
MapGetByValue (*class in aerospike_helpers.expressions.map*), 193
MapGetByValueList (*class in aerospike_helpers.expressions.map*), 193
MapGetByValueRange (*class in aerospike_helpers.expressions.map*), 194
MapGetByValueRelRankRange (*class in aerospike_helpers.expressions.map*), 194
MapGetByValueRelRankRangeToEnd (*class in aerospike_helpers.expressions.map*), 195
MapIncrement (*class in aerospike_helpers.expressions.map*), 195
mapkey_iterate_and() (*in module aerospike.predexp*), 106
mapkey_iterate_or() (*in module aerospike.predexp*), 105
MapPut (*class in aerospike_helpers.expressions.map*), 196
MapPutItems (*class in aerospike_helpers.expressions.map*), 196
MapRemoveByIndex (*class in aerospike_helpers.expressions.map*), 196
MapRemoveByIndexRange (*class in aerospike_helpers.expressions.map*), 197
MapRemoveByIndexRangeToEnd (*class in aerospike_helpers.expressions.map*), 197
MapRemoveByKey (*class in aerospike_helpers.expressions.map*), 197
MapRemoveByKeyList (*class in aerospike_helpers.expressions.map*), 198
MapRemoveByKeyRange (*class in aerospike_helpers.expressions.map*), 198
MapRemoveByKeyRelIndexRange (*class in aerospike_helpers.expressions.map*), 198
MapRemoveByKeyRelIndexRangeToEnd (*class in aerospike_helpers.expressions.map*), 199
MapRemoveByRank (*class in aerospike_helpers.expressions.map*), 199
MapRemoveByRankRange (*class in aerospike_helpers.expressions.map*), 199
MapRemoveByRankRangeToEnd (*class in aerospike_helpers.expressions.map*), 200
MapRemoveByValue (*class in aerospike_helpers.expressions.map*), 200
MapRemoveByValueList (*class in aerospike_helpers.expressions.map*), 200
MapRemoveByValueRange (*class in aerospike_helpers.expressions.map*), 201
MapRemoveByValueRelRankRange (*class in aerospike_helpers.expressions.map*), 201
MapRemoveByValueRelRankRangeToEnd (*class in aerospike_helpers.expressions.map*), 202
MapSize (*class in aerospike_helpers.expressions.map*), 202
mapval_iterate_and() (*in module aerospike.predexp*), 107
mapval_iterate_or() (*in module aerospike.predexp*), 106
Max (*class in aerospike_helpers.expressions.arithmetic*), 216
Min (*class in aerospike_helpers.expressions.arithmetic*), 216
Mod (*class in aerospike_helpers.expressions.arithmetic*), 217
module
    aerospike, 3
    aerospike.exception, 111
    aerospike.predexp, 100
    aerospike.predicates, 94
    aerospike_helpers, 117
    aerospike_helpers.cdt_ctx, 223
    aerospike_helpers.expressions.arithmetic,

213
aerospike_helpers.expressions.base, 163
aerospike_helpers.expressions.bitwise, 203
aerospike_helpers.expressions.bitwise_operations, 219
aerospike_helpers.expressions.hll, 210
aerospike_helpers.expressions.list, 174
aerospike_helpers.expressions.map, 187
aerospike_helpers.expressions.resources, 222
aerospike_helpers.operations.bitwise_operations, 145
aerospike_helpers.operations.expression_operations, 158
aerospike_helpers.operations.hll_operations, 153
aerospike_helpers.operations.list_operations, 118
aerospike_helpers.operations.map_operations, 131
aerospike_helpers.operations.operations, 117
module (*aerospike.exception.UDFError attribute*), 115
msg (*aerospike.exception.AerospikeError attribute*), 112
Mul (*class in aerospike_helpers.expressions.arithmetic*), 217
Multi-Ops, 41

N

NamespaceNotFound, 113
NE (*class in aerospike_helpers.expressions.base*), 171
Not (*class in aerospike_helpers.expressions.base*), 172
NotAuthenticated, 115
null (*in module aerospike*), 23
null() (*in module aerospike*), 8
Numeric Operations, 39

O

operate() (*aerospike.Client method*), 41
operate_ordered() (*aerospike.Client method*), 44
OpNotApplicable, 112
Or (*class in aerospike_helpers.expressions.base*), 172

P

ParamError, 112
POLICY_COMMIT_LEVEL_ALL (*in module aerospike*), 13
POLICY_COMMIT_LEVEL_MASTER (*in module aerospike*), 13
POLICY_EXISTS_CREATE (*in module aerospike*), 14
POLICY_EXISTS_CREATE_OR_REPLACE (*in module aerospike*), 14
POLICY_EXISTS_IGNORE (*in module aerospike*), 14
POLICY_EXISTS_REPLACE (*in module aerospike*), 14
POLICY_EXISTS_UPDATE (*in module aerospike*), 14
POLICY_GEN_EQ (*in module aerospike*), 14
POLICY_GEN_GT (*in module aerospike*), 14
POLICY_GEN_IGNORE (*in module aerospike*), 14
POLICY_KEY_DIGEST (*in module aerospike*), 15
POLICY_KEY_SEND (*in module aerospike*), 15
POLICY_READ_MODE_AP_ALL (*in module aerospike*), 13
POLICY_READ_MODE_AP_ONE (*in module aerospike*), 13
POLICY_READ_MODE_SC_ALLOW_REPLICA (*in module aerospike*), 14
POLICY_READ_MODE_SC_ALLOW_UNAVAILABLE (*in module aerospike*), 14
POLICY_READ_MODE_SC_LINEARIZE (*in module aerospike*), 14
POLICY_READ_MODE_SC_SESSION (*in module aerospike*), 14
POLICY_REPLICA_ANY (*in module aerospike*), 15
POLICY_REPLICA_MASTER (*in module aerospike*), 15
POLICY_REPLICA_PREFER_RACK (*in module aerospike*), 15
POLICY_REPLICA_SEQUENCE (*in module aerospike*), 15
POLICY_RETRY_NONE (*in module aerospike*), 15
POLICY_RETRY_ONCE (*in module aerospike*), 15
Pow (*class in aerospike_helpers.expressions.arithmetic*), 218
predexp() (*aerospike.Query method*), 90
predexp_and() (*in module aerospike.predexp*), 100
predexp_not() (*in module aerospike.predexp*), 101
predexp_or() (*in module aerospike.predexp*), 100
prepend() (*aerospike.Client method*), 39
prepend() (*in module aerospike_helpers.operations.operations*), 117
PRIV_DATA_ADMIN (*in module aerospike*), 24
PRIV_READ (*in module aerospike*), 24
PRIV_READ_WRITE (*in module aerospike*), 24
PRIV_READ_WRITE_UDF (*in module aerospike*), 24
PRIV_SYS_ADMIN (*in module aerospike*), 24
PRIV_USER_ADMIN (*in module aerospike*), 24
PRIV_WRITE (*in module aerospike*), 24
put() (*aerospike.Client method*), 27
PY_BYTES (*in module aerospike*), 17

Q

Query (*class in aerospike*), 84
query() (*aerospike.Client method*), 46
query_apply() (*aerospike.Client method*), 50
QueryError, 114
QueryQueueFull, 114
QueryTimeout, 114

R

range() (*in module aerospike.predicates*), 99

read() (*in module aerospike_helpers.operations.operations*), 118
rec_device_size() (*in module aerospike.predexp*), 108
rec_digest_modulo() (*in module aerospike.predexp*), 107
rec_last_update() (*in module aerospike.predexp*), 107
rec_void_time() (*in module aerospike.predexp*), 108
Record Operations, 26
RecordBusy, 113
RecordError, 113
RecordExistsError, 113
RecordGenerationError, 113
RecordKeyMismatch, 113
RecordNotFound, 113
RecordTooBig, 113
REGEX_EXTENDED (*in module aerospike*), 24
REGEX_ICASE (*in module aerospike*), 24
REGEX_NEWLINE (*in module aerospike*), 24
REGEX_NONE (*in module aerospike*), 24
REGEX_NOSUB (*in module aerospike*), 24
remove() (*aerospike.Client method*), 32
remove_bin() (*aerospike.Client method*), 33
results() (*aerospike.Query method*), 84
results() (*aerospike.Scan method*), 77
ResultType (*class in aerospike_helpers.expressions.resources*), 222
RoleExistsError, 115
RoleViolation, 115

S

Scan (*class in aerospike*), 76
Scan and Query, 45
scan() (*aerospike.Client method*), 46
scan_apply() (*aerospike.Client method*), 49
scan_info() (*aerospike.Client method*), 51
SCAN_PRIORITY (*in module aerospike*), 16
SCAN_STATUS_ABORTED (*in module aerospike*), 16
SCAN_STATUS_COMPLETED (*in module aerospike*), 16
SCAN_STATUS_INPROGRESS (*in module aerospike*), 16
SCAN_STATUS_UNDEF (*in module aerospike*), 16
SecurityNotEnabled, 115
SecurityNotSupported, 115
SecuritySchemeNotSupported, 115
select() (*aerospike.Client method*), 31
select() (*aerospike.Query method*), 84
select() (*aerospike.Scan method*), 76
select_many() (*aerospike.Client method*), 37
SERIALIZER_NONE (*in module aerospike*), 17
SERIALIZER_PYTHON (*in module aerospike*), 17
SERIALIZER_USER (*in module aerospike*), 17
ServerError, 112
ServerFull, 112
set_deserializer() (*in module aerospike*), 10
set_log_handler() (*in module aerospike*), 12
set_log_level() (*in module aerospike*), 12
set_serializer() (*in module aerospike*), 9
set_xdr_filter() (*aerospike.Client method*), 55
SetName (*class in aerospike_helpers.expressions.base*), 172
shm_key() (*aerospike.Client method*), 55
SinceUpdateTime (*class in aerospike_helpers.expressions.base*), 172
StrBin (*class in aerospike_helpers.expressions.base*), 173
STRING (*aerospike_helpers.expressions.resources.ResultType attribute*), 222
String Operations, 38
string_bin() (*in module aerospike.predexp*), 101
string_equal() (*in module aerospike.predexp*), 110
string_regex() (*in module aerospike.predexp*), 111
string_unequal() (*in module aerospike.predexp*), 110
string_value() (*in module aerospike.predexp*), 103
string_var() (*in module aerospike.predexp*), 104
Sub (*class in aerospike_helpers.expressions.arithmetic*), 218

T

ToFloat (*class in aerospike_helpers.expressions.arithmetic*), 218
ToInt (*class in aerospike_helpers.expressions.arithmetic*), 219
touch() (*aerospike.Client method*), 32
touch() (*in module aerospike_helpers.operations.operations*), 118
truncate() (*aerospike.Client method*), 55
TTL (*class in aerospike_helpers.expressions.base*), 173
TTL_DONT_UPDATE (*in module aerospike*), 15
TTL_NAMESPACE_DEFAULT (*in module aerospike*), 15
TTL_NEVER_EXPIRE (*in module aerospike*), 15

U

udf_get() (*aerospike.Client method*), 47
udf_list() (*aerospike.Client method*), 47
udf_put() (*aerospike.Client method*), 46
udf_remove() (*aerospike.Client method*), 46
UDF_TYPE_LUA (*in module aerospike*), 23
UDFError, 115
UDFNotFound, 115
Unknown (*class in aerospike_helpers.expressions.base*), 173
unset_serializers() (*in module aerospike*), 10
UnsupportedFeature, 113
unwrap() (*aerospike.GeoJSON method*), 227
User Defined Functions, 46
UserExistsError, 115

V

Var (*class in aerospike_helpers.expressions.base*), 174
VoidTime (*class in aerospike_helpers.expressions.base*), 174

W

where() (*aerospike.Query method*), 84
wrap() (*aerospike.GeoJSON method*), 227
write() (*in module aerospike_helpers.operations.operations*), 118