# aerospike Documentation

**Ronen Botzer**

**Feb 07, 2023**

# CONTENTS

The Aerospike Python 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.

The Python client is a CPython module built on the Aerospike C client.

### 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.

CHAPTER

# ONE

# LAYOUT

- *aerospike*
  - Constructors for the Client and GeoJSON classes
  - Server-side types
  - Serialization
  - Logging
  - Helper function for calculating key digest
  - Constants
- *aerospike.predicates*
  - Query predicates
- *aerospike.exception*
  - All exception classes
  - Exception hierarchy
- *aerospike_helpers*
  - Bin operations (list, map, bitwise, etc.)
  - Aerospike expressions
  - Batch operations
  - Complex data type context

The *aerospike* module contains these classes:

| Class | Description |
|---|---|
| *aerospike.Client — Client Class* | Aerospike client API |
| *aerospike.Scan — Scan Class* | Contains scan operations of entire sets. |
| *aerospike.Query — Query Class* | Handles queries over secondary indexes. |
| *aerospike.GeoJSON — GeoJSON Class* | Handles GeoJSON type data. |
| *aerospike.KeyOrderedDict — KeyOrderedDict Class* | Key ordered dictionary |

In addition, the *Python Data Mappings* page explains how **Python** types map to **Aerospike Server** types.

**See also:**

The Python Client Manual for a quick guide.

# 1.1 Content

## 1.1.1 aerospike — Aerospike Client for Python

### 1.1.1.1 Overview

`aerospike` is a package which provides a Python client for Aerospike database clusters.

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.

### 1.1.1.2 Methods

#### Constructors

#### Client

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

Creates a new instance of the *Client* class and immediately connects to the cluster.

See *aerospike.Client — Client Class* for more details.

Internally, this is a wrapper function which calls the constructor for the *Client* class. However, the client may also be constructed by calling the constructor directly.

The client takes on many configuration parameters passed in through a dictionary.

**Parameters**
**config** (*dict*) – See *Client Configuration*.

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

Changed in version 2.0.0.

Simple example:

```
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.
# Also sets a top level policy for read operations
config = {
    'hosts':    [ ('127.0.0.1', 3000) ],
    'policies': {'read': {total_timeout': 1000}},
client = aerospike.client(config)
```

Connecting using TLS example:

```
import aerospike
import sys

# NOTE: Use of TLS requires Aerospike Enterprise version >= 3.11
# and client version 2.1.0 or greater
tls_name = "some-server-tls-name"
```

(continues on next page)

(continued from previous page)

```
tls_ip = "127.0.0.1"
tls_port = 4333

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

### 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
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.

## Types

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 infinite 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

```

(continues on next page)

(continued from previous page)

```
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

---

### Serialization

---

**Note:** See *Python Data Mappings*.

---

---

**Note:** If the client's config dictionary has a serializer and deserializer in the *serialization* tuple, then it will take precedence over the ones from *set_serializer()* and *set_deserializer()*.

---

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

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

**Parameters**

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

**See also:**

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

```
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 *Client* instances.

Once registered, all read methods (such as *Client.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 deserializer/serializer available from *Client* instances.

New in version 1.0.53.

### Example

The following example shows the three modes of serialization:

1. Built-in
2. Class-level user functions
3. Instance-level user functions

```
import aerospike
import marshal
import json

# Serializers and deserializers
# Both local and global serializers use json library
# Functions print which one is being used

def classSerializer(obj):
    print("Using class serializer")
    return json.dumps(obj)

def classDeserializer(bytes):
    print("Using class deserializer")
    return json.loads(bytes)

def localSerializer(obj):
    print("Using local serializer")
    return json.dumps(obj)

def localDeserializer(bytes):
    print("Using local deserializer")
    return json.loads(bytes)

# First client has class-level serializer set in aerospike module
aerospike.set_serializer(classSerializer)
aerospike.set_deserializer(classDeserializer)
config = {
    'hosts': [('127.0.0.1', 3000)]
}
client = aerospike.client(config).connect()

# Second client has instance-level serializer set in client config
config['serialization'] = (localSerializer, localDeserializer)
client2 = aerospike.client(config).connect()

# Keys: foo1, foo2, foo3
keys = [('test', 'demo', f'foo{i}') for i in range(1, 4)]
# Tuple is an unsupported type
```

(continues on next page)

(continued from previous page)

```
tupleBin = {'bin': (1, 2, 3)}

# Use the default, built-in serialization (cPickle)
client.put(keys[0], tupleBin)
(_, _, bins) = client.get(keys[0])
print(bins)

# Expected output:
# {'bin': (1, 2, 3)}

# First client uses class-level, user-defined serialization
# No instance-level serializer was declared
client.put(keys[1], tupleBin, serializer=aerospike.SERIALIZER_USER)
(_, _, bins) = client.get(keys[1])
# Notice that json-encoding a tuple produces a list
print(bins)

# Expected output:
# Using class serializer
# Using class deserializer
# {'bin': [1, 2, 3]}

# Second client uses instance-level, user-defined serialization
# Instance-level serializer overrides class-level serializer
client2.put(keys[2], tupleBin, serializer=aerospike.SERIALIZER_USER)
(_, _, bins) = client2.get(keys[2])
print(bins)

# Expected output:
# Using local serializer
# Using local deserializer
# {'bin': [1, 2, 3]}

# Cleanup
client.batch_remove(keys)
client.close()
client2.close()
```

Records `foo1` and `foo2` should have different encodings from each other since they use different serializers. (record `foo3` uses the same encoding as `foo2`) If we read the data for each record using `aql`, it outputs the following data:

```
aql> select bin from test.demo where PK='foo1'
+-------------------------------------------------------------+--------+
| bin                                                         | PK     |
+-------------------------------------------------------------+--------+
| 80 04 95 09 00 00 00 00 00 00 00 4B 01 4B 02 4B 03 87 94 2E | "foo1" |
+-------------------------------------------------------------+--------+
1 row in set (0.000 secs)

OK

aql> select bin from test.demo where PK='foo2'
```

(continues on next page)

(continued from previous page)

```
+--------------------------+--------+
| bin                      | PK     |
+--------------------------+--------+
| 5B 31 2C 20 32 2C 20 33 5D | "foo2" |
+--------------------------+--------+
1 row in set (0.001 secs)

OK
```

### Logging

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

Enables aerospike log handler

**Parameters**

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

---

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

```
import aerospike
```

from __future__ import print_function import aerospike

aerospike.set_log_level(aerospike.LOG_LEVEL_DEBUG) aerospike.set_log_handler(callback)

---

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.

### Other

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
```

(continues on next page)

(continued from previous page)

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

#### 1.1.1.3 Client Configuration

These are the keys and expected values for the `config` dictionary passed to *aerospike.client()*.

Only the *hosts* key is required; the rest of the keys are optional.

**config**

- **hosts** (**list**)
  A list of tuples identifying a node (or multiple nodes) in the cluster.
  The tuple is in this format: `(address, port, [tls-name])`
  address: `str`
  port: `int`
  tls-name: `str`
  The client will connect to the first available node in the list called the *seed node*. From there, it will learn about the cluster and its partition map.
  If `tls-name` is specified, it must match the tls-name specified in the node's server configuration file, as well as the server's CA certificate.
- **user** (**str**)
  (Optional) A defined user with roles in the cluster. See `admin_create_user()`.
- **password** (**str**)
  (Optional) The password will be hashed by the client using bcrypt.
- **lua** (**dict**)
  (Optional) Contains the paths to two types of Lua modules
  **system_path** (**str**)
  The location of the system modules such as `aerospike.lua`
  Default: `/usr/local/aerospike/lua`
  **user_path** (**str**)
  The location of the user's record and stream UDFs .
  Default: `./`
- **policies** (**dict**)
  A `dict` of policies
  **read** (**dict**)
  Contains *Read Policies*.
  **write** (**dict**)
  Contains *Write Policies*.
  **apply** (**dict**)
  Contains *Apply Policies*.
  **operate** (**dict**)
  Contains *Operate Policies*.
  **remove** (**dict**)
  Contains *Remove Policies*.
  **query** (**dict**)
  Contains *Policies*.
  **scan** (**dict**)
  Contains *Policies*.
  **batch** (**dict**)
  Contains *Batch Policies*.
  **info** (**dict**)
  Contains *Info Policies*.
  **total_timeout** (**int**)
  **Deprecated**: set this individually in the *Policies* dictionaries.
  The default connection timeout in milliseconds
  **auth_mode**
  The authentication mode with the server.
  See *Auth Mode Constants* for possible values.
  Default: *aerospike.AUTH_INTERNAL*
  **login_timeout_ms** (**int**)
  Representing the node login timeout in milliseconds.
  Default: `5000`.
  **key**
  **Deprecated**: set this individually in the *Policies* dictionaries.
  Default key policy.
  See *Key Policy Options* for possible values.
  **exists**
  **Deprecated**: set in the *Write Policies* dictionary
  Default exists policy.
  See *Existence Policy Options* for possible values.
  **max_retries** (**int**)

**Deprecated**: set this individually in the *Policies* dictionaries.
Representing the number of times to retry a transaction

**replica**
**Deprecated**: set this in one or all of the following policy dictionaries:
*Read Policies*
*Write Policies*
*Apply Policies*
*Operate Policies*
*Remove Policies*
Default replica policy.
See *Replica Options* for possible values.

**commit_level**
**Deprecated**: set this as needed individually in the following policy dictionaries:
*Write Policies*
*Apply Policies*
*Operate Policies*
*Remove Policies*
Default commit level policy.
See *Commit Level Policy Options* for possible values.
**See also:**
Per-Transaction Consistency Guarantees.

- **shm** (`dict`)
  Contains optional shared-memory cluster tending parameters
  Shared-memory cluster tending is on if the `dict` is provided. If multiple clients are instantiated and talking to the same cluster the *shm* cluster-tending should be used.

  **max_nodes** (`int`)
  Maximum number of nodes allowed. Pad this value so new nodes can be added without configuration changes.
  Default: `16`

  **max_namespaces** (`int`)
  Maximum number of namespaces allowed.
  Pad this value so new namespaces can be added without configuration changes.
  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 *`Client.shm_key()`* .
  It is still possible to specify a key when using **use_shared_connection** = *True*.
  Default: `0xA8000000`
  **See also:**
  Shared Memory

- **use_shared_connection** (`bool`)
  Indicates whether this instance should share its connection to the Aerospike cluster with other client instances in the same process.
  Default: `False`

- **tls** (`dict`)
  Contains optional TLS configuration parameters.

  ---
  **Note:** TLS usage requires Aerospike Enterprise Edition. See TLS.
  ---

  **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** (*int*)
  Configures the client to encode Python booleans as the native Python boolean type, an integer, or the server boolean type.
  Use one of the *Send Bool Constants* constant values.
  See *Python Data Mappings* for more information.
  Default: *aerospike.AS_BOOL*
- **serialization** (*tuple*)
  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.
  It's important to set this value to a few seconds less than the server's proto-fd-idle-ms, so the client does not attempt to use a socket that has already been reaped by the server.
  The value is limited to 24 hours (86400 seconds).
  Default:
  `0` (disabled) for non-TLS connections
  `55` 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*)
  **Deprecated**: set in the *Write Policies* dictionary
  Compress data for transmission if the object size is greater than a given number of bytes
  Default: `0`, meaning 'never compress'
- **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.

Default: `0`
- **rack_aware** (`bool`)
  Track server rack data.
  This is useful for:
  Directing read operations to run on the same rack as the client.
  Lowering cloud provider costs when nodes are distributed across different availability zones (represented as racks).
  In order to enable this functionality:
  `rack_id` needs to be set to the local rack's ID
  The client config's *Read Policies* needs to be set to *POLICY_REPLICA_PREFER_RACK*
  The server rack configuration must 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`.
- **fail_if_not_connected** (`bool`)
  Flag to signify fail on cluster init if seed node and all peers are not reachable.
  Default: `True`

#### 1.1.1.4 Constants

##### Operators

Operators for the single-record, multi-operation transaction method *Client.operate()*.

---

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

---

##### Policy Options

##### 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 success only after successfully committing all replicas

`aerospike.`**POLICY_COMMIT_LEVEL_MASTER**
  Return success after successfully committing the master replica

##### 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.

### 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.

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

New in version 3.7.0.

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.

### Existence Policy Options

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

aerospike.**POLICY_EXISTS_CREATE**
: Only create a record given it doesn't exist

aerospike.**POLICY_EXISTS_CREATE_OR_REPLACE**
: Replace a record completely if it exists, otherwise create it

aerospike.**POLICY_EXISTS_IGNORE**
: Update a record if it exists, otherwise create it

aerospike.**POLICY_EXISTS_REPLACE**
: Only replace a record completely if it exists

aerospike.**POLICY_EXISTS_UPDATE**
: Only update a record if it exists

### 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

### 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

### Replica Options

Specifies which partition replica to read from.

`aerospike.`**POLICY_REPLICA_SEQUENCE**
: Always try node containing master partition first.

  If connection fails and the client is configured to retry, it will try the 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.

### 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.

### 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.

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.

> **Warning:** This mode should only be used for testing purposes because it is not secure authentication.

### 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 *Client.scan_info()*

aerospike.**SCAN_STATUS_COMPLETED**

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

aerospike.**SCAN_STATUS_INPROGRESS**

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

aerospike.**SCAN_STATUS_UNDEF**

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

New in version 1.0.39.

### Job Constants

aerospike.**JOB_SCAN**

Scan job type argument for the module parameter of *Client.job_info()*

aerospike.**JOB_QUERY**

Query job type argument for the module parameter of *Client.job_info()*

### Job Statuses

aerospike.**JOB_STATUS_UNDEF**

aerospike.**JOB_STATUS_INPROGRESS**

aerospike.**JOB_STATUS_COMPLETED**

New in version 1.0.50.

### 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.

### Send Bool Constants

Specifies how the Python client will write Python booleans.

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

  This is Python's native boolean type.

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

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

  This is the Aerospike server's boolean type.

### List

### 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.

### 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.

### List Order

Flags used by list order.

aerospike.**LIST_UNORDERED**

List is not ordered. This is the default.

aerospike.**LIST_ORDERED**

Ordered list.

### 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.

### Maps

### 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.

### 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.

### 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.

### 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 keys and values next to each other:

```
[key1, value1, key2, value2, ...]
```

### Bitwise

### 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.

### 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.

### 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.

### 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.

### Write Expression 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.

### Read Expression 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.

### 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_BOOL**
(int): 17

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

### Miscellaneous

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

  New in version 1.0.54.

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.

### 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**

### 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.

aerospike.**PRIV_TRUNCATE**

User can truncate data only. Requires server 6.0+

aerospike.**PRIV_UDF_ADMIN**

User can perform user defined function(UDF) administration actions. Examples include create/drop UDF. Global scope only. Global scope only. Requires server version 6.0+

aerospike.**PRIV_SINDEX_ADMIN**

User can perform secondary index administration actions. Examples include create/drop index. Global scope only. Requires server version 6.0+

### Regex Flag Values

Flags used by the `aerospike_operation_helpers.expressions.base.CmpRegex` Aerospike expression. See *aerospike_helpers.expressions package* for more information.

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.

### 1.1.2 aerospike.Client — Client Class

#### 1.1.2.1 Overview

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.

#### Boilerplate Code For Examples

Assume every in-line example runs this code beforehand:

> **Warning:** Only run example code on a brand new Aerospike server. This code deletes all records in the `demo` set!

```
# Imports
import aerospike
from aerospike import exception as ex
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()
    client.truncate('test', "demo", 0)
except ex.ClientError as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    sys.exit(1)

# Record key tuple: (namespace, set, key)
keyTuple = ('test', 'demo', 'key')
```

Basic example:

```
# Write a record
client.put(keyTuple, {'name': 'John Doe', 'age': 32})

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

#### 1.1.2.2 Methods

### Connection

**class** aerospike.**Client**

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

If there is currently no connection to the cluster, connect to it. 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.

You may call *connect()* again after closing the connection.

### Record Operations

**class** aerospike.**Client**

**put**(*key*, *bins: dict*[, *meta: dict*[, *policy: dict*[, *serializer=aerospike.SERIALIZER_PYTHON* ]]])

Create a new record, or remove / add bins to a record.

**Parameters**

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **bins** (*dict*) – contains bin name-value pairs of the record.
- **meta** (*dict*) – record metadata to be set. see *Metadata Dictionary*.
- **policy** (*dict*) – see *Write Policies*.

- **serializer** – 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*.

Example:

```
# Insert a record with bin1
client.put(keyTuple, {'bin1': 4})

# Insert another bin named bin2
client.put(keyTuple, {'bin2': "value"})

# Remove bin1 from this record
client.put(keyTuple, {'bin2': aerospike.null()})

# Removing the last bin should delete this record
client.put(keyTuple, {'bin1': aerospike.null()})
```

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

Check if a record with a given key exists in the cluster.

Returns the record's key and metadata in a tuple.

If the record does not exist, the tuple's metadata will be None.

**Parameters**

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

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

**Raises**
a subclass of *AerospikeError*.

```
# Check non-existent record
(key, meta) = client.exists(keyTuple)

print(key) # ('test', 'demo', 'key', bytearray(b'...'))
print(meta) # None

# Check existing record
client.put(keyTuple, {'bin1': 4})
(key, meta) = client.exists(keyTuple)

print(key) # ('test', 'demo', 'key', bytearray(b'...'))
print(meta) # {'ttl': 2592000, 'gen': 1}
```

Changed in version 2.0.3.

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

Returns a record with a given key.

**Parameters**

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

**Returns**
a *Record Tuple*.

**Raises**
*RecordNotFound*.

```
# Get nonexistent record
try:
    client.get(keyTuple)
except ex.RecordNotFound as e:
    print("Error: {0} [{1}]".format(e.msg, e.code))
    # Error: 127.0.0.1:3000 AEROSPIKE_ERR_RECORD_NOT_FOUND [2]

# Get existing record
client.put(keyTuple, {'bin1': 4})
(key, meta, bins) = client.get(keyTuple)

print(key) # ('test', 'demo', None, bytearray(b'...'))
print(meta) # {'ttl': 2592000, 'gen': 1}
print(bins) # {'bin1': 4}
```

Changed in version 2.0.0.

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

Returns specific bins of a record.

If a bin does not exist, it will not show up 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*.

**Raises**
*RecordNotFound*.

```
# Record to select from
client.put(keyTuple, {'bin1': 4, 'bin2': 3})

# Only get bin1
(key, meta, bins) = client.select(keyTuple, ['bin1'])

# Similar output to get()
print(key) # ('test', 'demo', 'key', bytearray(b'...'))
print(meta) # {'ttl': 2592000, 'gen': 1}
print(bins) # {'bin1': 4}

# Get all bins
(key, meta, bins) = client.select(keyTuple, ['bin1', 'bin2'])
```

(continues on next page)

(continued from previous page)

```
print(bins) # {'bin1': 4, 'bin2': 3}

# Get nonexistent bin
(key, meta, bins) = client.select(keyTuple, ['bin3'])
print(bins) # {}
```

Changed in version 2.0.0.

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

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

**Parameters**

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

**Raises**

a subclass of *AerospikeError*.

```
# Insert record and get its metadata
client.put(keyTuple, bins = {"bin1": 4})
(key, meta) = client.exists(keyTuple)
print(meta) # {'ttl': 2592000, 'gen': 1}

# Explicitly set TTL to 120
# and increment generation
client.touch(keyTuple, 120)

# Record metadata should be updated
(key, meta) = client.exists(keyTuple)
print(meta) # {'ttl': 120, 'gen': 2}
```

**remove**(*key[meta: dict[, policy: dict]]*)

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

**Parameters**

- **key** (*tuple*) – a *Key Tuple* associated with the record.
- **meta** (*dict*) – contains the expected generation of the record in a key called `"gen"`.
- **policy** (*dict*) – see *Remove Policies*. May be passed as a keyword argument.

**Raises**

a subclass of *AerospikeError*.

```
# Insert a record
client.put(keyTuple, {"bin1": 4})

# Try to remove it with the wrong generation
try:
    client.remove(keyTuple, meta={'gen': 5}, policy={'gen': aerospike.POLICY_
↪GEN_EQ})
```

(continues on next page)

(continued from previous page)

```
except ex.AerospikeError as e:
    # Error: AEROSPIKE_ERR_RECORD_GENERATION [3]
    print("Error: {0} [{1}]".format(e.msg, e.code))

# Remove it ignoring generation
client.remove(keyTuple)
```

**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

```
digest = client.get_key_digest("test", "demo", "key")
print(digest.hex())
# 3bd475bd0c73f210b67ea8379330 0eeae576285d
```

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

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

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*) – record metadata to be set. See *Metadata Dictionary*.
- **policy** (*dict*) – optional *Write Policies*.

**Raises**
a subclass of *AerospikeError*.

```
# Insert record
bins = {"bin1": 0, "bin2": 1}
client.put(keyTuple, bins)

# Remove bin1
client.remove_bin(keyTuple, ['bin1'])

# Only bin2 shold remain
(keyTuple, meta, bins) = client.get(keyTuple)
print(bins)
# {'bin2': 1}
```

### Batch Operations

**class** aerospike.**Client**

**get_many**(*keys*[, *policy: dict* ]) → [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*) – see *Batch Policies*.

**Returns**
a `list` of *Record Tuple*.

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

```
# Keys
keyTuples = [
    ('test', 'demo', '1'),
    ('test', 'demo', '2'),
    ('test', 'demo', '3'),
]
# Only insert two records with the first and second key
client.put(keyTuples[0], {'bin1': 'value'})
client.put(keyTuples[1], {'bin1': 'value'})

# Try to get records with all three keys
records = client.get_many(keyTuples)
# The third record tuple should have 'meta' and 'bins' set to none
# Because there is no third record
print(records[0])
print(records[1])
print(records[2])

# Expected output:
# (('test', 'demo', '1', bytearray(...)), {'ttl': 2592000, 'gen': 1}, {'bin1': 'value'})
# (('test', 'demo', '2', bytearray(...)), {'ttl': 2592000, 'gen': 1}, {'bin1': 'value'})
# (('test', 'demo', '3', bytearray(...)), None, None)
```

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

Batch-read metadata for multiple keys.

Any record that does not exist will have a `None` value for metadata in their tuple.

**Parameters**

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

**Returns**
a `list` of (key, metadata) `tuple` for each record.

```
# Keys
# Only insert two records with the first and second key
keyTuples = [
    ('test', 'demo', '1'),
    ('test', 'demo', '2'),
    ('test', 'demo', '3'),
]
client.put(keyTuples[0], {'bin1': 'value'})
client.put(keyTuples[1], {'bin1': 'value'})

# Check for existence of records using all three keys
keyMetadata = client.exists_many(keyTuples)
print(keyMetadata[0])
print(keyMetadata[1])
print(keyMetadata[2])

# (('test', 'demo', '1', bytearray(...)), {'ttl': 2592000, 'gen': 1})
# (('test', 'demo', '2', bytearray(...)), {'ttl': 2592000, 'gen': 1})
# (('test', 'demo', '3', bytearray(...)), None)
```

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

Batch-read specific bins from multiple records.

Any record that does not exist will have a `None` value for metadata and bins in its tuple.

**Parameters**

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

**Returns**

a `list` of *Record Tuple*.

```
# Insert 4 records with these keys
keyTuples = [
    ('test', 'demo', 1),
    ('test', 'demo', 2),
    ('test', 'demo', 3),
    ('test', 'demo', 4)
]
# Only records 1, 2, 4 have a bin called bin2
client.put(keyTuples[0], {'bin1': 20, 'bin2': 40})
client.put(keyTuples[1], {'bin1': 11, 'bin2': 50})
client.put(keyTuples[2], {'bin1': 50,              'bin3': 20})
client.put(keyTuples[3], {'bin1': 87, 'bin2': 76, 'bin3': 40})

# Get all 4 records and filter out every bin except bin2
records = client.select_many(keyTuples, ['bin2'])
for record in records:
    print(record)

# (('test', 'demo', 1, bytearray(...)), {'ttl': 2592000, 'gen': 1}, {'bin2': 40})
# (('test', 'demo', 2, bytearray(...)), {'ttl': 2592000, 'gen': 1}, {'bin2': 50})
```

(continues on next page)

(continued from previous page)

```
# (('test', 'demo', 3, bytearray(...)), {'ttl': 2592000, 'gen': 1}, {})
# (('test', 'demo', 4, bytearray(...)), {'ttl': 2592000, 'gen': 1}, {'bin2': 76})
```

**batch_get_ops**(*keys*, *ops*, *policy: dict*) → [key, meta, bins]

Batch-read multiple records, and return them as a `list`.

Any record that does not exist will have a exception type value as metadata and `None` value as bins in the record tuple.

**Parameters**

- **keys** (*list*) – a list of *Key Tuple*.
- **ops** (*list*) – a list of operations to apply.
- **policy** (*dict*) – see *Batch Policies*.

**Returns**

a `list` of *Record Tuple*.

**Raises**

a *ClientError* if the batch is too big.

```
# Insert records for 3 players and their scores
keyTuples = [("test", "demo", i) for i in range(1, 4)]
bins = [
    {"scores": [1, 4, 3, 10]},
    {"scores": [20, 1, 4, 28]},
    {"scores": [50, 20, 10, 20]},
]
for keyTuple, bin in zip(keyTuples, bins):
    client.put(keyTuple, bin)

# Get highest scores for each player
from aerospike_helpers.operations import list_operations
ops = [
    list_operations.list_get_by_rank("scores", -1, aerospike.LIST_RETURN_VALUE)
]
records = client.batch_get_ops(keyTuples, ops)

# Print results
for _, _, bins in records:
    print(bins)
# {'scores': 10}
# {'scores': 28}
# {'scores': 50}
```

The following batch methods will return a `BatchRecords` object with a `result` value of `0` if one of the following is true:

- All transactions are successful.
- One or more transactions failed because:
  - A record was filtered out by an expression
  - The record was not found

Otherwise if one or more transactions failed, the `BatchRecords` object will have a `result` value equal to an as_status error code.

In any case, the `BatchRecords` object has a list of batch records called `batch_records`, and each batch record contains the result of that transaction.

**batch_write**(*batch_records: BatchRecords*[, *policy_batch: dict* ]) → *BatchRecords*

Write/read multiple records for specified batch keys in one batch call.

This method allows different sub-commands for each key in the batch. The resulting status and operated bins are set in `batch_records.results` and `batch_records.record`.

**Parameters**

- **batch_records** (*BatchRecords*) – A `BatchRecords` object used to specify the operations to carry out.
- **policy_batch** (*dict*) – aerospike batch policy *Batch Policies*.

**Returns**

A reference to the batch_records argument of type *BatchRecords*.

**Raises**

A subclass of *AerospikeError*.

```
from aerospike_helpers.batch import records as br
from aerospike_helpers.operations import operations as op

# Keys
# Only insert two records with the first and second key
keyTuples = [
    ('test', 'demo', 'Robert'),
    ('test', 'demo', 'Daniel'),
    ('test', 'demo', 'Patrick'),
]
client.put(keyTuples[0], {'id': 100, 'balance': 400})
client.put(keyTuples[1], {'id': 101, 'balance': 200})
client.put(keyTuples[2], {'id': 102, 'balance': 300})

# Apply different operations to different keys
batchRecords = br.BatchRecords(
    [
        # Remove Robert from system
        br.Remove(
            key = keyTuples[0],
        ),
        # Modify Daniel's ID and balance
        br.Write(
            key = keyTuples[1],
            ops = [
                op.write("id", 200),
                op.write("balance", 100),
                op.read("id"),
            ],
        ),
        # Read Patrick's ID
        br.Read(
```

(continues on next page)

(continued from previous page)

```
            key = keyTuples[2],
            ops=[
                op.read("id")
            ],
            policy=None
        ),
    ]
)

client.batch_write(batchRecords)

# batch_write modifies its BatchRecords argument.
# Results for each BatchRecord will be set in the result, record, and in_doubt␣
↪fields.
for batchRecord in batchRecords.batch_records:
    print(batchRecord.result)
    print(batchRecord.record)
# Note how written bins return None if their values aren't read
# And removed records have an empty bins dictionary
# 0
# (('test', 'demo', 'Robert', bytearray(b'...')), {'ttl': 4294967295, 'gen': 0}, {})
# 0
# (('test', 'demo', 'Daniel', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {'id': 200,
↪ 'balance': None})
# 0
# (('test', 'demo', 'Patrick', bytearray(b'...')), {'ttl': 2592000, 'gen': 1}, {'id':␣
↪102})
```

**Note:** Requires server version >= 6.0.0.

**See also:**

More information about the batch helpers *aerospike_helpers.batch package*

**batch_operate**(*keys: list*, *ops: list*[, *policy_batch: dict* ][, *policy_batch_write: dict* ]) → *BatchRecords*

Perform the same read/write transactions on multiple keys.

**Parameters**

- **keys** (*list*) – The keys to operate on.
- **ops** (*list*) – List of operations to apply.
- **policy_batch** (*dict*) – See *Batch Policies*.
- **policy_batch_write** (*dict*) – See *Batch Write Policies*.

**Returns**

an instance of *BatchRecords*.

**Raises**

A subclass of *AerospikeError*.

```
from aerospike_helpers.operations import operations as op
```

(continues on next page)

(continued from previous page)

```
# Insert 3 records
keys = [("test", "demo", f"employee{i}") for i in range(1, 4)]
bins = [
    {"id": 100, "balance": 200},
    {"id": 101, "balance": 400},
    {"id": 102, "balance": 300}
]
for key, bin in zip(keys, bins):
    client.put(key, bin)

# Increment ID by 100 and balance by 500 for all employees
# NOTE: batch_operate ops must include a write op
# get_batch_ops or get_many can be used for all read ops use cases.
ops = [
    op.increment("id", 100),
    op.increment("balance", 500),
    op.read("balance")
]

batchRecords = client.batch_operate(keys, ops)
print(batchRecords.result)
# 0

# Print each individual transaction's results
# and record if it was read from
for batchRecord in batchRecords.batch_records:
    print(f"{batchRecord.result}: {batchRecord.record}")
# 0: (('test', 'demo', 'employee1', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {'id
↪': None, 'balance': 700})
# 0: (('test', 'demo', 'employee2', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {'id
↪': None, 'balance': 900})
# 0: (('test', 'demo', 'employee3', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {'id
↪': None, 'balance': 800})
```

**Note:** Requires server version >= 6.0.0.

**batch_apply**(*keys: list*, *module: str*, *function: str*, *args: list*[, *policy_batch: dict* ][, *policy_batch_apply: dict* ]) → *BatchRecords*

Apply UDF (user defined function) on multiple keys.

**Parameters**

- **keys** (*list*) – The keys to operate on.
- **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_batch** (*dict*) – See *Batch Policies*.
- **policy_batch_apply** (*dict*) – See *Batch Apply Policies*.

**Returns**
an instance of *BatchRecords*.

**Raises**
A subclass of *AerospikeError*.

```
# Insert 3 records
keys = [("test", "demo", f"employee{i}") for i in range(1, 4)]
bins = [
    {"id": 100, "balance": 200},
    {"id": 101, "balance": 400},
    {"id": 102, "balance": 300}
]
for key, bin in zip(keys, bins):
    client.put(key, bin)

# Apply a user defined function (UDF) to a batch
# of records using batch_apply.
client.udf_put("batch_apply.lua")

args = ["balance", 0.5, 100]
batchRecords = client.batch_apply(keys, "batch_apply", "tax", args)

print(batchRecords.result)
# 0

for batchRecord in batchRecords.batch_records:
    print(f"{batchRecord.result}: {batchRecord.record}")
# 0: (('test', 'demo', 'employee1', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {
↪'SUCCESS': 0})
# 0: (('test', 'demo', 'employee2', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {
↪'SUCCESS': 100})
# 0: (('test', 'demo', 'employee3', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {
↪'SUCCESS': 50})
```

```
-- Deduct tax and fees from bin
function tax(record, binName, taxRate, fees)
    if aerospike:exists(record) then
        record[binName] = record[binName] * (1 - taxRate) - fees
        aerospike:update(record)
    else
        record[binName] = 0
        aerospike:create(record)
    end
    return record[binName]
end
```

**Note:** Requires server version >= 6.0.0.

**batch_remove**(*keys: list*[, *policy_batch: dict* ][, *policy_batch_remove: dict* ]) → *BatchRecords*

**Note:** Requires server version >= 6.0.0.

Remove multiple records by key.

**Parameters**

- **keys** (*list*) – The keys to remove.
- **policy_batch** (*dict*) – Optional aerospike batch policy *Batch Policies*.
- **policy_batch_remove** (*dict*) – Optional aerospike batch remove policy *Batch Remove Policies*.

**Returns**
an instance of *BatchRecords*.

**Raises**
A subclass of *AerospikeError*.

```
# Insert 3 records
keys = [("test", "demo", f"employee{i}") for i in range(1, 4)]
bins = [
    {"id": 100, "balance": 200},
    {"id": 101, "balance": 400},
    {"id": 102, "balance": 300}
]
for key, bin in zip(keys, bins):
    client.put(key, bin)

batchRecords = client.batch_remove(keys)

# A result of 0 means success
print(batchRecords.result)
# 0
for batchRecord in batchRecords.batch_records:
    print(batchRecord.result)
    print(batchRecord.record)
# 0: (('test', 'demo', 'employee1', bytearray(b'...')), {'ttl': 4294967295, 'gen': 0},
↪{})
# 0: (('test', 'demo', 'employee2', bytearray(b'...')), {'ttl': 4294967295, 'gen': 0},
↪{})
# 0: (('test', 'demo', 'employee3', bytearray(b'...')), {'ttl': 4294967295, 'gen': 0},
↪{})
```

### 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: dict*[, *policy: dict* ]])
Append a string 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 bin value.
- **meta** (*dict*) – record metadata to be set. See *Metadata Dictionary*.
- **policy** (*dict*) – optional *Operate Policies*.

**Raises**
a subclass of *AerospikeError*.

```
client.put(keyTuple, {'bin1': 'Martin Luther King'})
client.append(keyTuple, 'bin1', ' jr.')
(_, _, bins) = client.get(keyTuple)
print(bins) # Martin Luther King jr.
```

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

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 bin value.
- **meta** (*dict*) – record metadata to be set. See *Metadata Dictionary*.
- **policy** (*dict*) – optional *Operate Policies*.

**Raises**
a subclass of *AerospikeError*.

```
client.put(keyTuple, {'bin1': 'Freeman'})
client.prepend(keyTuple, 'bin1', ' Gordon ')
(_, _, bins) = client.get(keyTuple)
print(bins) # Gordon Freeman
```

### 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: dict*[, *policy: dict* ]])

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*) – record metadata to be set. See *Metadata Dictionary*.
- **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*.

```
# Start with 100 lives
client.put(keyTuple, {'lives': 100})

# Gain health
client.increment(keyTuple, 'lives', 10)
(key, meta, bins) = client.get(keyTuple)
print(bins) # 110

# Take damage
client.increment(keyTuple, 'lives', -90)
(key, meta, bins) = client.get(keyTuple)
print(bins) # 20
```

### List Operations

**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.

### Map Operations

**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.

### Single-Record Transactions

**class** aerospike.**Client**

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

Performs an atomic transaction, with multiple bin operations, against a single record with a given *key*.

Starting with Aerospike server version 3.6.0, non-existent bins are not 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. (In Aerospike server versions prior to 3.6.0, non-existent bins being read will have a `None` value. )

**Parameters**

- **key** (*tuple*) – a *Key Tuple* associated with the record.

- **operations** (*list*) – See *aerospike_helpers.operations package*.
- **meta** (*dict*) – record metadata to be set. See *Metadata Dictionary*.
- **policy** (*dict*) – optional *Operate Policies*.

**Returns**
a *Record Tuple*.

**Raises**
a subclass of *AerospikeError*.

```
from aerospike_helpers.operations import operations

# Add name, update age, and return attributes
client.put(keyTuple, {'age': 25, 'career': 'delivery boy'})
ops = [
    operations.increment("age", 1000),
    operations.write("name", "J."),
    operations.prepend("name", "Phillip "),
    operations.append("name", " Fry"),
    operations.read("name"),
    operations.read("career"),
    operations.read("age")
]
(key, meta, bins) = client.operate(key, ops)

print(key) # ('test', 'demo', None, bytearray(b'...'))
# The generation should only increment once
# A transaction is *atomic*
print(meta) # {'ttl': 2592000, 'gen': 2}
print(bins) # Will display all bins selected by read operations
# {'name': 'Phillip J. Fry', 'career': 'delivery boy', 'age': 1025}
```

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

Changed in version 2.1.3.

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

Performs an atomic transaction, with multiple bin operations, against a single record with a given *key*. The results will be 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.
- **operations** (*list*) – See *aerospike_helpers.operations package*.
- **meta** (*dict*) – record metadata to be set. See *Metadata Dictionary*.
- **policy** (*dict*) – optional *Operate Policies*.

**Returns**
a *Record Tuple*.

**Raises**
a subclass of *AerospikeError*.

```
from aerospike_helpers.operations import operations

# Add name, update age, and return attributes
client.put(keyTuple, {'age': 25, 'career': 'delivery boy'})
ops = [
    operations.increment("age", 1000),
    operations.write("name", "J."),
    operations.prepend("name", "Phillip "),
    operations.append("name", " Fry"),
    operations.read("name"),
    operations.read("career"),
    operations.read("age")
]
(key, meta, bins) = client.operate_ordered(keyTuple, ops)

# Same output for key and meta as operate()
# But read operations are outputted as bin-value pairs
print(bins)
# [('name': 'Phillip J. Fry'), ('career': 'delivery boy'), ('age': 1025)]
```

Changed in version 2.1.3.

### User Defined Functions

**class** aerospike.**Client**

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

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:** To run this example, do not run the boilerplate code.

---

```
import aerospike

config = {
    'hosts': [ ('127.0.0.1', 3000)],
    'lua': { 'user_path': '/path/to/lua/user_path'}
}
client = aerospike.client(config).connect()
# Register the UDF module and copy it to the Lua 'user_path'
client.udf_put('/path/to/my_module.lua')
client.close()
```

**udf_remove**(*module*[, *policy: dict* ])

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: dict* ]) → []

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*.

```
print(client.udf_list())
# [
#    {'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}
# ]
```

**udf_get**(*module*[, *language=aerospike.UDF_TYPE_LUA*[, *policy: dict* ] ]) → 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: dict* ])

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.

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

Deprecated since version 7.0.0: *aerospike.Query* should be used instead.

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 *Policies*.
- **options** (*dict*) – the *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*.

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

Initiate a query and apply a record UDF to each record matched by the query.

This method blocks until the query is complete.

**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*.

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

Return the status of a job running in the background.

The returned `dict` contains these keys:

- `"status"`: see *Job Statuses* for possible values.
- `"records_read"`: number of scanned records.
- `"progress_pct"`: progress percentage of the job

**Parameters**

- **job_id** (*int*) – the job ID returned by *scan_apply()* or *query_apply()*.
- **module** – one of *Job Constants*.
- **policy** – optional *Info Policies*.

**Returns**
dict

**Raises**
a subclass of *AerospikeError*.

**scan_info**(*scan_id*) → dict

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

Return the status of a scan running in the background.

The returned `dict` contains these keys:

- `"status"`: see *Scan Constants* for possible values.
- `"records_scanned"`: number of scanned records.
- `"progress_pct"`: progress percentage of the job

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

**Returns**
dict

**Raises**
a subclass of *AerospikeError*.

### Info Operations

**class** aerospike.**Client**

**get_node_names**() → []

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

**Returns**
a list of node info dictionaries.

**Raises**
a subclass of *AerospikeError*.

```
# Assuming two nodes
nodes = client.get_node_names()
print(nodes)
# [{'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*.

```
# Assuming two nodes
nodes = client.get_nodes()
print(nodes)
# [('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_single_node**(*command*, *host*[, *policy: dict* ]) → str

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

**Parameters**

- **command** (*str*) – the info command. See Info Command Reference.
- **host** (*str*) – 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.

**info_all**(*command[, policy: dict]]*) → {}

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*) – see Info Command Reference.
- **policy** (*dict*) – optional *Info Policies*.

**Return type**
dict

**Raises**
a subclass of *AerospikeError*.

```
response = client.info_all("namespaces")
print(response)
# {'BB9020011AC4202': (None, 'test\n')}
```

New in version 3.0.0.

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

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

**Parameters**

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

**Return type**
str

**Raises**
a subclass of *AerospikeError*.

Changed in version 6.0.0.

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

Deprecated since version 6.0.0: Use *info_single_node()* to send a request to a single node.

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

**Parameters**

- **command** (*str*) – the info command. See Info Command Reference.
- **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*.

Changed in version 3.0.0.

> **Warning:** for client versions < 3.0.0 `info_node()` will not work when using TLS

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

Deprecated since version 3.0.0: Use *info_all()* to send a request to the entire cluster.

Send an info command to all nodes in the cluster, and optionally filter responses to only include certain nodes.

**Parameters**

- **command** (*str*) – the info command. See Info Command Reference
- **hosts** (*list*) – a `list` containing `(address, port)` tuples. If specified, only send the command to these hosts. Example: `[('127.0.0.1', 3000)]`
- **policy** (*dict*) – optional *Info Policies*.

**Return type**
`dict`

**Raises**
a subclass of *AerospikeError*.

```
response = client.info(command)
client.close()
# {'BB9581F41290C00': (None, '127.0.0.1:3000\n'), 'BC3581F41290C00': (None, '127.0.0.
↪1:3010\n')}
```

---

**Note:** 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
```

---

Changed in version 3.0.0.

**set_xdr_filter**(*data_center*, *namespace*, *expression_filter*[, *policy: dict* ]) → 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** (*AerospikeExpression*) – 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.

**get_expression_base64**(*expression*) → str

Get the base64 representation of a compiled aerospike expression.

See *aerospike_helpers.expressions package* for more details on expressions.

**Parameters**
**expression** (*AerospikeExpression*) – the compiled expression.

**Raises**
a subclass of *AerospikeError*.

```
from aerospike_helpers import expressions as exp

# Compile expression
expr = exp.Eq(exp.IntBin("bin1"), 6).compile()

base64 = client.get_expression_base64(expr)
print(base64)
# kwGTUQKkYmluMQY=
```

Changed in version 7.0.0.

**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: dict* ])

Remove all records in the namespace / set whose last updated time is older than the given time.

This method is many orders of magnitude faster than deleting records one at a time. See Truncate command reference.

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 to truncate.
- **set** (*str*) – The set to truncate. Pass in None to truncate a namespace instead.
- **nanos** (*long*) – A cutoff threshold where records last updated before the threshold will be removed. Units are in nanoseconds since the 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*) – See *Info Policies*.

**Return type**
Status indicating the success of the operation.

**Raises**
a subclass of *AerospikeError*.

**Note:** Requires Aerospike server version >= 3.12

```
import time

client.put(("test", "demo", "key1"), {"bin": 4})

time.sleep(1)
# Take threshold time
current_time = time.time()
time.sleep(1)

client.put(("test", "demo", "key2"), {"bin": 5})

threshold_ns = int(current_time * 10**9)
# Remove all items in set `demo` created before threshold time
# Record using key1 should be removed
client.truncate('test', 'demo', threshold_ns)

# Remove all items in namespace
# client.truncate('test', None, 0)
```

### Index Operations

**class** aerospike.**Client**

**index_string_create**(*ns*, *set*, *bin*, *index_name*[, *policy: dict* ])
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: dict* ])

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: dict* ])

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: dict* ])

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: dict* ])

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: dict* ])

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*.

**get_cdtctx_base64**(*ctx: list*) → str

Get the base64 representation of aerospike CDT ctx.

See *aerospike_helpers.cdt_ctx module* for more details on CDT context.

**Parameters**

**ctx** (*list*) – Aerospike CDT context: generated by aerospike CDT ctx helper *aerospike_helpers*.

**Raises**

a subclass of *AerospikeError*.

```
import aerospike
from aerospike_helpers import cdt_ctx

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

ctxs = [cdt_ctx.cdt_ctx_list_index(0)]
ctxs_base64 = client.get_cdtctx_base64(ctxs)
print("Base64 encoding of ctxs:", ctxs_base64)

client.close()
```

Changed in version 7.1.1.

### Admin Operations

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.

**class** aerospike.**Client**

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

Create a custom role containing a `list` of privileges, as well as an optional whitelist and quotas.

**Parameters**

- **role** (*str*) – The name of the role.
- **privileges** (*list*) – A list of *Privilege Objects*.
- **policy** (*dict*) – See *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 `0` for no limit.
- **write_quota** (*int*) – Maximum write per second limit, Pass in `0` for no limit.

**Raises**

One of the *AdminError* subclasses.

**admin_set_whitelist**(*role*, *whitelist*[, *policy: dict* ])

Add a 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 this to None will clear the whitelist for that role.
- **policy** (*dict*) – See *Admin Policies*.

**Raises**
One of the *AdminError* subclasses.

**admin_set_quotas**(*role*[, *read_quota*[, *write_quota*[, *policy: dict* ] ] ])

Add quotas to a role.

**Parameters**

- **role** (*str*) – the name of the role.
- **read_quota** (*int*) – Maximum reads per second limit. Pass in 0 for no limit.
- **write_quota** (*int*) – Maximum write per second limit. Pass in 0 for no limit.
- **policy** (*dict*) – See *Admin Policies*.

**Raises**
one of the *AdminError* subclasses.

**admin_drop_role**(*role*[, *policy: dict* ])

Drop a custom role.

**Parameters**

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

**Raises**
one of the *AdminError* subclasses.

**admin_grant_privileges**(*role*, *privileges*[, *policy: dict* ])

Add privileges to a role.

**Parameters**

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

**Raises**
one of the *AdminError* subclasses.

**admin_revoke_privileges**(*role*, *privileges*[, *policy: dict* ])

Remove privileges from a role.

**Parameters**

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

**Raises**
one of the *AdminError* subclasses.

**admin_get_role**(*role*[, *policy: dict* ]) → {}

Get a dict of privileges, whitelist, and quotas associated with a role.

**Parameters**

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

**Returns**
a *Role Objects*.

**Raises**
one of the *AdminError* subclasses.

**admin_get_roles**([*policy: dict* ]) → {}

Get the names of all roles and their attributes.

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

**Returns**
a dict of *Role Objects* keyed by role names.

**Raises**
one of the *AdminError* subclasses.

**admin_query_role**(*role*[, *policy: dict* ]) → []

Get the list of privileges associated with a role.

**Parameters**

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

**Returns**
a list of *Privilege Objects*.

**Raises**
one of the *AdminError* subclasses.

**admin_query_roles**([*policy: dict* ]) → {}

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: dict* ])

Create a user 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: dict* ])

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: dict* ])

Change the password of a user.

This operation can only be performed by that same user.

**Parameters**

- **username** (*str*) – the username of the user.
- **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: dict* ])

Set the password of a user 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: dict* ])

Add roles to a user.

**Parameters**

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

**Raises**
one of the *AdminError* subclasses.

**admin_revoke_roles**(*username*, *roles*[, *policy: dict* ])

Remove roles from a user.

**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: dict* ]) → []

Return the list of roles granted to the specified user.

**Parameters**

- **username** (*str*) – the username of the user.
- **policy** (*dict*) – optional *Admin Policies*.

**Returns**
a `list` of role names.

**Raises**
one of the *AdminError* subclasses.

**admin_query_users**([*policy: dict* ]) → {}

Get the roles of all users.

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

**Returns**
a `dict` of roles keyed by username.

**Raises**
one of the *AdminError* subclasses.

### Scan and Query Constructors

**class** aerospike.**Client**

**scan**(*namespace*[, *set* ]) → *Scan*

Deprecated since version 7.0.0: *aerospike.Query* should be used instead.

Returns a *aerospike.Scan* object to scan all records in a namespace / set.

If set is omitted or set to `None`, the object returns all 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 in a namespace.

See *aerospike.Query — Query Class* for more details.

**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.

#### 1.1.2.3 Tuples

### Key Tuple

**key**

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

```
(namespace, set, primary key[, digest])
```

- **namespace** (**str**)
  Name of the namespace.

  This must be preconfigured on the cluster.
- **set** (**str**)
  Name of the set.

  The set be created automatically if it does not exist.
- **primary key** (**str**, **int** or **bytearray**)
  The value by which the client-side application identifies the record.
- **digest**
  The record's RIPEMD-160 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.

The following code example shows:

- How to use the key tuple in a *put* operation
- How to fetch the key tuple in a *get* operation

```
import aerospike

# NOTE: change this to your Aerospike server's seed node address
seedNode = ('127.0.0.1', 3000)
config = config = {'hosts': [seedNode]}
client = aerospike.client(config).connect()

# The key tuple comprises the following:
namespaceName = 'test'
setName = 'setname'
primaryKeyName = 'pkname'
keyTuple = (namespaceName, setName, primaryKeyName)

# Insert a record
recordBins = {'bin1':0, 'bin2':1}
client.put(keyTuple, recordBins)

# Now fetch that record
(key, meta, bins) = client.get(keyTuple)

# The key should be in the second format
```

(continues on next page)

(continued from previous page)

```
# Notice how there is no primary key
# and there is the record's digest
print(key)

# Expected output:
# ('test', 'setname', None, bytearray(b'b\xc7[\xbb\xa4K\xe2\x9al\xd12!&\xbf<\xd9\
↪xf9\x1bPo'))

# Cleanup
client.remove(keyTuple)
client.close()
```

**See also:**

Data Model: Keys and Digests.

## Record Tuple

### record

The record tuple which is returned by various read operations. It has the structure:

`(key, meta, bins)`

- **key** (**tuple**)
  See *Key Tuple*.
- **meta** (**dict**)
  Contains record metadata with the following key-value pairs:
  **gen** (**int**)
  Generation value
  **ttl** (**int**)
  Time-to-live value
- **bins** (**dict**)
  Contains bin-name/bin-value pairs.

We reuse the code example in the key-tuple section and print the `meta` and `bins` values that were returned from `client.get()`:

```
import aerospike

# NOTE: change this to your Aerospike server's seed node address
seedNode = ('127.0.0.1', 3000)
config = {'hosts': [seedNode]}
client = aerospike.client(config).connect()

namespaceName = 'test'
setName = 'setname'
primaryKeyName = 'pkname'
keyTuple = (namespaceName, setName, primaryKeyName)

# Insert a record
recordBins = {'bin1':0, 'bin2':1}
client.put(keyTuple, recordBins)

# Now fetch that record
(key, meta, bins) = client.get(keyTuple)
```

(continues on next page)

(continued from previous page)

```
# Generation is 1 because this is the first time we wrote the record
print(meta)

# Expected output:
# {'ttl': 2592000, 'gen': 1}

# The bin-value pairs we inserted
print(bins)
{'bin1': 0, 'bin2': 1}

client.remove(keyTuple)
client.close()
```

**See also:**

Data Model: Record.

#### 1.1.2.4 Metadata Dictionary

The metadata dictionary has the following key-value pairs:

- "ttl" (int): record time to live in seconds. See *TTL Constants*.
- "gen" (int): record generation

#### 1.1.2.5 Policies

##### Write Policies

**policy**

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

- **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: `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.

  ---

- **compression_threshold** (`int`)
  Compress data for transmission if the object size is greater than a given number of bytes. Default: `0`, meaning 'never compress'

### Read Policies

**policy**

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

- **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: 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: 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.

### Operate Policies

**`policy`**

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

- **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: `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.

### Apply Policies

**policy**

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

- **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: `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.

  ---

### Remove Policies

**policy**

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

- **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: `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.

### Batch Policies

**`policy`**

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

- **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: 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: 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`

- **allow_inline_ssd** (*bool*)
  Allow batch to be processed immediately in the server's receiving thread for SSD namespaces. If false, the batch will always be processed in separate service threads. Server versions < 6.0 ignore this field. Inline processing can introduce the possibility of unfairness because the server can process the entire batch before moving onto the next command.
  Default: `False`
- **send_set_name** (*bool*)

  .. deprecated:: in client version 7.0.0, the client ignores this policy and always sends set name to the server.

  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`
- **expressions** *list*

  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

  **Note:** Requires Aerospike server version >= 5.2.

- **respond_all_keys** *bool*
  Should all batch keys be attempted regardless of errors. This field is used on both the client and server. The client handles node specific errors and the server handles key specific errors.
  If `True`, every batch key is attempted regardless of previous key specific errors. Node specific errors such as timeouts stop keys to that node, but keys directed at other nodes will continue to be processed.
  If `False`, the server will stop the batch to its node on most key specific errors. The exceptions are `AEROSPIKE_ERR_RECORD_NOT_FOUND` and `AEROSPIKE_FILTERED_OUT` which never stop the batch. The client will stop the entire batch on node specific errors for sync commands that are run in sequence (`concurrent` == false). The client will not stop the entire batch for async commands or sync commands run in parallel.
  Server versions < 6.0 do not support this field and treat this value as false for key specific errors.
  Default: `True`

### Batch Write Policies

**policy**

A *dict* of optional batch write policies, which are applicable to `batch_write()`, `batch_operate()` and *Write*.

- **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*

- **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

### Batch Apply Policies

**policy**

A `dict` of optional batch apply policies, which are applicable to `batch_apply()`, and *Apply*.

- **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*

- **ttl int**

  Time to live (expiration) of the record in seconds.

  0 which means that the record will adopt the default TTL value from the namespace.

  0xFFFFFFFF (also, -1 in a signed 32 bit int) which means that the record will get an internal "void_time" of zero, and thus will never expire.

  0xFFFFFFFE (also, -2 in a signed 32 bit int) which means that the record

  ttl will not change when the record is updated.
  Note that the TTL value will be employed ONLY on write/update calls.

  Default: 0

- **expressions** `list`

  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

### Batch Remove Policies

**policy**

A dict of optional batch remove policies, which are applicable to batch_remove(), and *Remove*.

- **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*
- **generation int**

  Generation of the record.

  Default: 0
- **durable_delete (bool)**

  Perform durable delete

  Default: False
- **expressions list**

  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

### Batch Read Policies

**policy**

A dict of optional batch read policies, which are applicable to *Read*.

- **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
- **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*
- **expressions list**

  Compiled aerospike expressions *aerospike_helpers* used for filtering records within a transaction.

  Default: None

### Info Policies

**policy**

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

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

### Admin Policies

**policy**

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

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

### List Policies

**policy**

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

- **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
}
```

### 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* .

- **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
}
```

### Bit Policies

**policy**

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

---

**Note:** Requires server version >= 4.6.0

---

- **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
}
```

### HyperLogLog Policies

**policy**

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

**Note:** Requires server version >= 4.9.0

- **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
}
```

#### 1.1.2.6 Misc

### Role Objects

**Role**

A dict describing attributes associated with a specific role:

- "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.

### Privilege Objects

**privilege**

A dict describing a privilege and where it applies to:

- "code": one of the *Privileges* values
- "ns": **optional** str specifying the namespace where the privilege applies.
  If not specified, the privilege applies globally.
- "set": **optional** str specifying the set within the namespace where the privilege applies.
  If not specified, the privilege applies to the entire namespace.

Example:

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

### Partition Objects

**partition_filter**

A dict of partition information used by the client to perform partition queries or scans. Useful for resuming terminated queries and querying particular partitions or records.

- "begin": **Optional** int signifying which partition to start at.
  Default: 0 (the first partition)
- "count": **Optional** int signifying how many partitions to process.
  Default: 4096 (all partitions)
- "digest": **Optional** dict containing the keys "init" and "value" signifying whether the digest has been calculated, and the digest value.

  "init": bool Whether the digest has been calculated.
  "value": bytearray The bytearray value of the digest, should be 20 characters long.
  Default: {} (will start from first record in partition)
- "partition_status": **Optional** dict containing partition_status tuples. These can be used to resume a query/scan.
  Default: {} (all partitions)

Default: {} (All partitions will be queried/scanned).

```
# Example of a query policy using partition_filter.

# partition_status is most easily used to resume a query
# and can be obtained by calling Query.get_partitions_status()
partition_status = {
    0: {0, False, Flase, bytearray([0]*20)}...
}

policy = {
    "partition_filter": {
        "partition_status": partition_status,
        "begin": 0,
        "count": 4096
    },
}
```

**partition_status**

**Note:** Requires Aerospike server version >= 6.0.

A `dict` of partition status information used by the client to set the partition status of a partition query or scan.

This is useful for resuming either of those.

The dictionary contains these key-value pairs:

- `"retry"`: `str` represents the overall retry status of this partition query. (i.e. Does this query/scan need to be retried?)

  This maps to a boolean value.

- `"done"`: `str` represents whether all partitions were finished.

  This maps to a boolean value.

In addition, the dictionary contains keys of the partition IDs (`int`), and each partition ID is mapped to a dictionary containing the status details of a partition.

Each partition ID has a dictionary with the following keys:

- `"id"`: `int` represents a partition ID number
- `"init"`: `bool` represents whether the digest being queried was calculated.
- `"retry"`: `bool` represents whether this partition should be retried.
- `"digest"`: `bytearray` represents the digest of the record being queried.
  Should be 20 characters long.
- `"bval"`: `int` is used in conjunction with `"digest"` to determine the last record received by a partition query.

Default: `{}` (All partitions will be queried).

```
# Example of a query policy using partition_status.

# Here is the form of partition_status.
# partition_status = {
#     0: (0, False, Flase, bytearray([0]*20), 0)...
# }
partition_status = query.get_partitions_status()

policy = {
    "partition_filter": {
        "partition_status": partition_status,
        "begin": 0,
        "count": 4096
    },
}
```

### 1.1.3 aerospike.Scan — Scan Class

Deprecated since version 7.0.0: *aerospike.Query* should be used instead.

#### 1.1.3.1 Overview

The Scan object is used to return all the records in a specified set (which can be omitted 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.

#### 1.1.3.2 Methods

**class** aerospike.**Scan**

Deprecated since version 7.0.0: *aerospike.Query* should be used instead.

**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 unsupported 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), bool.

**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
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 *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
↪')),
```

(continues on next page)

(continued from previous page)

```
    { 'gen': 52, 'ttl': 2592000},
    { 'id': 2}),
  ( ( 'test',
      'test',
      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:** As of client 7.0.0 and with server >= 6.0 results and the scan policy "partition_filter" see *Partition Objects* can be used to specify which partitions/records results will scan. See the example below.

```
# This is an example of scaning partitions 1000 - 1003.
import aerospike


scan = client.scan("test", "demo")

policy = {
    "partition_filter": {
        "begin": 1000,
        "count": 4
    },
}

# NOTE that these will only be non 0 if there are records in partitions 1000 -␣
↪1003
# results will be the records in partitions 1000 - 1003
results = scan.results(policy=policy)
```

**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 *Policies*.
- **options** (*dict*) – the *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. If the scan is using the "partition_filter" scan policy the callback will receive two arguments The first is a `int` representing partition id, the second is the same *Record Tuple* as a normal callback.

```
import aerospike
import pprint
```

(continues on next page)

(continued from previous page)

```
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)

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.

```
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))
```

(continues on next page)

(continued from previous page)

```
print(len(keys)) # this will be 100 if the number of matching records > 100
client.close()
```

---

**Note:** As of client 7.0.0 and with server >= 6.0 foreach and the scan policy "partition_filter" see *Partition Objects* can be used to specify which partitions/records foreach will scan. See the example below.

```
# This is an example of scaning partitions 1000 - 1003.
import aerospike


partitions = []

def callback(part_id, input_tuple):
    print(part_id)
    partitions.append(part_id)

scan = client.scan("test", "demo")

policy = {
    "partition_filter": {
        "begin": 1000,
        "count": 4
    },
}

scan.foreach(callback, policy)


# NOTE that these will only be non 0 if there are records in partitions 1000 -␣
↪1003
# should be 4
print(len(partitions))

# should be [1000, 1001, 1002, 1003]
print(partitions)
```

---

**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.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.

```
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
```

(continues on next page)

(continued from previous page)

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

---

**paginate**()

Makes a scan instance a paginated scan. Call this if you are using the "max_records" scan policy and you need to scan data in pages.

---

**Note:** Calling .paginate() on a scan instance causes it to save its partition state. This can be retrieved later using .get_partitions_status(). This can also be done using the partition_filter policy.

---

```
# scan 3 pages of 1000 records each.

import aerospike

pages = 3
page_size = 1000
policy = {"max_records": 1000}

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

scan.paginate()

# NOTE: The number of pages queried and records returned per page can differ
# if record counts are small or unbalanced across nodes.
for page in range(pages):
    records = scan.results(policy=policy)

    print("got page: " + str(page))

    if scan.is_done():
        print("all done")
        break

# This id can be used to paginate queries.
```

**is_done**()

If using scan pagination, did the previous paginated or partition_filter scan using this scan instance return all records?

**Returns**

A `bool` signifying whether this paginated scan instance has returned all records.

```
import aerospike

policy = {"max_records": 1000}

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

(continues on next page)

(continued from previous page)

```
scan.paginate()

records = scan.results(policy=policy)

if scan.is_done():
    print("all done")

# This id can be used to monitor the progress of a paginated scan.
```

**get_partitions_status**()

Get this scan instance's partition status. That is which partitions have been queried and which have not. The returned value is a `dict` with partition id, `int`, as keys and `tuple` as values. If the scan instance is not tracking its partitions, the returned `dict` will be empty.

---

**Note:** A scan instance must have had .paginate() called on it in order retrieve its partition status. If .paginate() was not called, the scan instance will not save partition status.

---

**Returns**

a `tuple` of form (id: `int`, init: class`bool`, done: class`bool`, digest: `bytearray`). See *Partition Objects* for more information.

```
# This is an example of resuming a scan using partition status.
import aerospike


for i in range(15):
    key = ("test", "demo", i)
    bins = {"id": i}
    client.put(key, bins)

records = []
resumed_records = []

def callback(input_tuple):
    record, _, _ = input_tuple

    if len(records) == 5:
        return False

    records.append(record)

scan = client.scan("test", "demo")
scan.paginate()

scan.foreach(callback)

# The first scan should stop after 5 records.
assert len(records) == 5
```

(continues on next page)

(continued from previous page)

```
partition_status = scan.get_partitions_status()

def resume_callback(part_id, input_tuple):
    record, _, _ = input_tuple
    resumed_records.append(record)

scan_resume = client.scan("test", "demo")

policy = {
    "partition_filter": {
        "partition_status": partition_status
    },
}

scan_resume.foreach(resume_callback, policy)

# should be 15
total_records = len(records) + len(resumed_records)
print(total_records)

# cleanup
for i in range(15):
    key = ("test", "demo", i)
    client.remove(key)
```

#### 1.1.3.3 Policies

**policy**

A `dict` of optional scan policies which are applicable to *`Scan.results()`* and *`Scan.foreach()`*. See *Policies*.

- **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).

  ---

  **Note:** Requires Aerospike server version >= 6.0

  ---

- **partition_filter** `dict`

  A dictionary of partition information used by the client
  to perform partiton scans. Useful for resuming terminated scans and
  scaning particular partitons/records.

  See *Partition Objects* for more information.

  Default: `{}` (All partitions will be scanned).

#### 1.1.3.4 Options

**options**

A dict of optional scan options which are applicable to *Scan.foreach()*.

- **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 *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.

### 1.1.4 aerospike.Query — Query Class

#### 1.1.4.1 Overview

##### Constructing A Query

The query object is used for executing queries over a secondary index of a specified set. It can be created by calling *aerospike.Client.query()*.

A query without a secondary index filter will apply to all records in the namespace, similar to a *Scan*.

Otherwise, the query can optionally be assigned one of the secondary index filters in *aerospike.predicates* to filter out records using their bin values. These secondary index filters are applied to the query using *where()*. In this case, if the set is initialized to None, then the query will only apply to records without a set.

---

**Note:** The secondary index filters in *aerospike.predicates* are **not** the same as the deprecated predicate expressions. For more details, read this guide.

---

##### Writing Using Query

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 aerospike_helpers.operations.

### Query Aggregations

A stream UDF may be applied with *apply()*. It will aggregate results out of the records streaming back from the query.

### Getting Results From Query

The returned bins can be filtered by using `select()`.

Finally, the query is invoked using one of these methods:

- *foreach()*
- *results()*
- *execute_background()*

**See also:**

Queries and Managing Queries.

#### 1.1.4.2 Fields

**class** aerospike.**Query**

**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)

---

**Note:** Requires server version >= 6.0.0

---

**records_per_second** (**int**)
Limit the scan to process records at records_per_second. Requires server version >= 6.0.0

Default: `0` (no limit)

**ttl** (**int**)
The time-to-live (expiration) of the record in seconds.

There are also special values that can be set in the record TTL:

**0** (**TTL_NAMESPACE_DEFAULT**)
Which means that the record will adopt the default TTL value from the namespace.

**0xFFFFFFFF** (**TTL_NEVER_EXPIRE**)
(also, `-1` in a signed 32 bit int) Which means that the record will never expire.

**0xFFFFFFFE** (**TTL_DONT_UPDATE**)
(also, `-2` in a signed 32 bit int) Which means that the record ttl will not change when the record is updated.

---

**Note:** Note that the TTL value will be employed ONLY on background query writes.

---

Requires server version >= 6.0.0

Default: `0` (record will adopt the default TTL value from the namespace)

#### 1.1.4.3 Methods

Assume this boilerplate code is run before all examples below:

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

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

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

# Remove old indices
try:
    client.index_remove("test", "scoreIndex")
    client.index_remove("test", "eloIndex")
except ex.AerospikeError as e:
    # Ignore if no indices found
    pass

# Insert 4 records
keyTuples = [("test", "demo", f"player{i}") for i in range(4)]
bins = [
    {"score": 100, "elo": 1400},
    {"score": 20, "elo": 1500},
    {"score": 10, "elo": 1100},
    {"score": 200, "elo": 900}
]
for keyTuple, bin in zip(keyTuples, bins):
    client.put(keyTuple, bin)

query = client.query('test', 'demo')

# Queries require a secondary index for each bin name
client.index_integer_create("test", "demo", "score", "scoreIndex")
client.index_integer_create("test", "demo", "elo", "eloIndex")
```

**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*[, *ctx* ])

Set a where *predicate* for the query.

You can only assign at most one predicate to the query. If this function isn't called, the query will behave similar to *aerospike.Scan*.

**Parameters**

- **predicate** (*tuple*) – the `tuple` produced by either *equals()* or *between()*.
- **ctx** (*list*) – the `list` produced by one of the *aerospike_helpers.cdt_ctx* methods.

**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 *Policies*.
- **options** (*dict*) – optional *Options*.

**Returns**

a `list` of *Record Tuple*.

```
from aerospike import predicates

query.select('score')
query.where(predicates.equals('score', 100))

records = query.results()
# Matches one record
print(records)
# [(('test', 'demo', None, bytearray(b'...')), {'ttl': 2592000, 'gen': 1}, {'score':␣
↪100})]
```

---

**Note:** As of client 7.0.0 and with server >= 6.0 results and the query policy "partition_filter" see *Partition Objects* can be used to specify which partitions/records results will query. See the example below.

```
# This is an example of querying partitions 1000 - 1003.
import aerospike


query = client.query("test", "demo")

policy = {
    "partition_filter": {
        "begin": 1000,
        "count": 4
    },
}
```

# NOTE that these will only be non 0 if there are records in partitions 1000 - 1003 # results will be the records in partitions 1000 - 1003 results = query.results(policy=policy)

---

**foreach**(*callback*[, *policy*[, *options* ] ])

Invoke the *callback* function for each of the records streaming back from the query.

A *Record Tuple* is passed as the argument to the callback function. If the query is using the "partition_filter" query policy the callback will receive two arguments The first is a `int` representing partition id, the second is the same *Record Tuple* as a normal callback.

**Parameters**

- **callback** (*callable*) – the function to invoke for each record.
- **policy** (*dict*) – optional *Policies*.
- **options** (*dict*) – optional *Options*.

```
# Callback function
# Calculates new elo for a player
def updateElo(record):
    keyTuple, _, bins = record
    # Add score to elo
    bins["elo"] = bins["elo"] + bins["score"]
    client.put(keyTuple, bins)

query.foreach(updateElo)

# Player elos should be updated
records = client.get_many(keyTuples)
for _, _, bins in records:
    print(bins)
# {'score': 100, 'elo': 1500}
# {'score': 20, 'elo': 1520}
# {'score': 10, 'elo': 1110}
# {'score': 200, 'elo': 1100}
```

**Note:** To stop the stream return `False` from the callback function.

```
# Adds record keys from a stream to a list
# But limits the number of keys to "lim"
def limit(lim: int, result: list):
    # Integers are immutable
    # so a list (mutable) is used for the counter
    c = [0]
    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

from aerospike import predicates as p

keys = []
query.foreach(limit(2, keys))
print(len(keys)) # 2
```

**Note:** As of client 7.0.0 and with server >= 6.0 foreach and the query policy "partition_filter" see *Partition Objects* can be used to specify which partitions/records foreach will query. See the example below.

```
# This is an example of querying partitions 1000 - 1003.
import aerospike


partitions = []

def callback(part_id, input_tuple):
    print(part_id)
    partitions.append(part_id)

query = client.query("test", "demo")

policy = {
    "partition_filter": {
        "begin": 1000,
        "count": 4
    },
}

query.foreach(callback, policy)


# NOTE that these will only be non 0 if there are records in partitions 1000 -␣
↪1003
# should be 4
print(len(partitions))

# should be [1000, 1001, 1002, 1003]
print(partitions)
```

**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 unsupported 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), `bool`.

**See also:**

Developing Stream UDFs

Example: find the first name distribution of users who are 21 or older using a query aggregation:

```
-- Filter function
-- Filters records with a bin value >= a threshold
local function is_greater_than_or_equal(binname, threshold)
    return function(rec)
        if rec[binname] < threshold then
            return false
        end
        return true
    end
end

-- Creates an aggregate function that counts the number of times a specific bin␣
↪value is found
local function count(bin_name)
    return function(counts_map, rec)
        -- Does record have that specific bin?
        if rec[bin_name] then
            -- Account for that bin value
            local bin_value = rec[bin_name]
            counts_map[bin_value] = (counts_map[bin_value] or 0) + 1
        end
        -- No changes to bin value counts
        return counts_map
    end
end

-- Helper function for reduce
local function add_values(val1, val2)
    return val1 + val2
end

-- Combines count maps into one
-- Need this function when the database runs multiple aggregations in parallel
local function reduce_groups(a, b)
    return map.merge(a, b, add_values)
end

-- First filter records with a bin binname that has value >= threshold (if␣
↪those arguments are passed in)
-- Then count the number of times a value in bin "binname_to_group" is found
function group_count(stream, binname_to_group, binname, threshold)
    if binname and threshold then
        local filter = is_greater_than_or_equal(binname, threshold)
        return stream : filter(filter) : aggregate(map{}, count(binname_to_
↪group)) : reduce(reduce_groups)
    else
        -- Don't filter records in this case
        return stream : aggregate(map{}, count(binname_to_group)) :␣
↪reduce(reduce_groups)
    end
end
```

Assume the example code above is in a file called “example.lua”, and is the same folder as the following

script.

```
import aerospike

config = {'hosts': [('127.0.0.1', 3000)],
            'lua': {'system_path':'/usr/local/aerospike/lua/',
                    'user_path':'./'}}
client = aerospike.client(config).connect()
client.udf_put("example.lua")

# Remove index if it already exists
from aerospike import exception as ex
try:
    client.index_remove("test", "ageIndex")
except ex.IndexNotFound:
    pass

bins = [
    {"name": "Jeff", "age": 20},
    {"name": "Derek", "age": 24},
    {"name": "Derek", "age": 21},
    {"name": "Derek", "age": 29},
    {"name": "Jeff", "age": 29},
]
keys = [("test", "users", f"user{i}") for i in range(len(bins))]
for key, recordBins in zip(keys, bins):
    client.put(key, recordBins)

client.index_integer_create("test", "users", "age", "ageIndex")

query = client.query('test', 'users')
query.apply('example', 'group_count', ['name', 'age', 21])
names = query.results()

# we expect a dict (map) whose keys are names, each with a count value
print(names)
# One of the Jeffs is excluded because he is under 21
# [{'Derek': 3, 'Jeff': 1}]

# Cleanup
client.index_remove("test", "ageIndex")
client.batch_remove(keys)
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.

---

**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.job_info()` to track the status of the `aerospike.JOB_QUERY` , as it runs in the background.

```
# EXAMPLE 1: Increase everyone's score by 100

from aerospike_helpers.operations import operations
ops = [
    operations.increment("score", 100)
]
query.add_ops(ops)
id = query.execute_background()

# Allow time for query to complete
import time
time.sleep(3)

for key in keyTuples:
    _, _, bins = client.get(key)
    print(bins)
# {"score": 200, "elo": 1400}
# {"score": 120, "elo": 1500}
# {"score": 110, "elo": 1100}
# {"score": 300, "elo": 900}

# EXAMPLE 2: Increase score by 100 again for those with elos > 1000
# Use write policy to select players by elo
import aerospike_helpers.expressions as expr
eloGreaterOrEqualTo1000 = expr.GE(expr.IntBin("elo"), 1000).compile()
writePolicy = {
    "expressions": eloGreaterOrEqualTo1000
}
id = query.execute_background(policy=writePolicy)

time.sleep(3)

for i, key in enumerate(keyTuples):
    _, _, bins = client.get(key)
    print(bins)
```

(continues on next page)

(continued from previous page)

```
# {"score": 300, "elo": 1400} <--
# {"score": 220, "elo": 1500} <--
# {"score": 210, "elo": 1100} <--
# {"score": 300, "elo": 900}

# Cleanup and close the connection to the Aerospike cluster.
for key in keyTuples:
    client.remove(key)
client.close()
```

**paginate**()

Makes a query instance a paginated query. Call this if you are using the max_records and you need to query data in pages.

---

**Note:** Calling .paginate() on a query instance causes it to save its partition state. This can be retrieved later using .get_partitions_status(). This can also been done by using the partition_filter policy.

---

```
# After inserting 4 records...
# Query 3 pages of 2 records each.

pages = 3
page_size = 2

query.max_records = 2
query.paginate()

# NOTE: The number of pages queried and records returned per page can differ
# if record counts are small or unbalanced across nodes.
for page in range(pages):
    records = query.results()
    print("got page: " + str(page))

    # Print records in each page
    for record in records:
        print(record)

    if query.is_done():
        print("all done")
        break
# got page: 0
# (('test', 'demo', None, bytearray(b'HD\xd1\xfa$L\xa0\xf5\xa2~\xd6\x1dv\x91\x9f\
↪xd6\xfa\xad\x18\x00')), {'ttl': 2591996, 'gen': 1}, {'score': 20, 'elo': 1500})
# (('test', 'demo', None, bytearray(b'f\xa4\t"\xa9uc\xf5\xce\x97\xf0\x16\x9eI\xab\
↪x89Q\xb8\xef\x0b')), {'ttl': 2591996, 'gen': 1}, {'score': 10, 'elo': 1100})
# got page: 1
# (('test', 'demo', None, bytearray(b'\xb6\x9f\xf5\x7f\xfarb.IeaVc\x17n\xf4\x9b\
↪xad\xa7T')), {'ttl': 2591996, 'gen': 1}, {'score': 200, 'elo': 900})
# (('test', 'demo', None, bytearray(b'j>@\xfe\xe0\x94\xd5?\n\xd7\xc3\xf2\xd7\x045\
↪xbc*\x07 \x1a')), {'ttl': 2591996, 'gen': 1}, {'score': 100, 'elo': 1400})
# got page: 2
# all done
```

**is_done()**

If using query pagination, did the previous paginated or partition_filter query using this query instance return all records?

**Returns**

A `bool` signifying whether this paginated query instance has returned all records.

**get_partitions_status()**

Get this query instance's partition status. That is which partitions have been queried and which have not. The returned value is a `dict` with partition id, `int`, as keys and `tuple` as values. If the query instance is not tracking its partitions, the returned `dict` will be empty.

---

**Note:** A query instance must have had .paginate() called on it, or been used with a partition filter, in order retrieve its partition status. If .paginate() was not called, or partition_filter was not used, the query instance will not save partition status.

---

**Returns**

a `tuple` of form (id: `int`, init: class`bool`, done: class`bool`, digest: `bytearray`). See *Partition Objects* for more information.

```
# Only read 2 records

recordCount = 0
def callback(record):
    global recordCount
    if recordCount == 2:
        return False
    recordCount += 1

    print(record)

# Query is set to read ALL records
query = client.query("test", "demo")
query.paginate()
query.foreach(callback)
# (('test', 'demo', None, bytearray(b'...')), {'ttl': 2591996, 'gen': 1}, {'score': 10,
↪'elo': 1100})
# (('test', 'demo', None, bytearray(b'...')), {'ttl': 2591996, 'gen': 1}, {'score': 20,
↪'elo': 1500})


# Use this to resume query where we left off
partition_status = query.get_partitions_status()

# Callback must include partition_id parameter
# if partition_filter is included in policy
def resume_callback(partition_id, record):
    print(partition_id, "->", record)

policy = {
    "partition_filter": {
        "partition_status": partition_status
```

(continues on next page)

(continued from previous page)

```
    },
}

query.foreach(resume_callback, policy)
# 1096 -> (('test', 'demo', None, bytearray(b'...')), {'ttl': 2591996, 'gen': 1}, {
↪'score': 100, 'elo': 1400})
# 3690 -> (('test', 'demo', None, bytearray(b'...')), {'ttl': 2591996, 'gen': 1}, {
↪'score': 200, 'elo': 900})
```

#### 1.1.4.4 Policies

**policy**

A `dict` of optional query policies which are applicable to *Query.results()* and *Query.foreach()*. See *Policies*.

- **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`

- **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`

- **short_query** `bool`

  Is query expected to return less than 100 records.
  If True, the server will optimize the query for a small record set.
  This field is ignored for aggregation queries, background queries
  and server versions less than 6.0.0.

  Mututally exclusive with records_per_second
  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.

- **partition_filter** `dict`

  A dictionary of partition information used by the client
  to perform partiton queries. Useful for resuming terminated queries and
  querying particular partitons/records.

  See *Partition Objects* for more information.

  Default: `{}` (All partitions will be queried).

  **Note:** Requires Aerospike server version >= 6.0

#### 1.1.4.5 Options

**`options`**

A `dict` of optional query options which are applicable to *Query.foreach()* and *Query.results()*.

- **nobins** `bool`

  Whether to return the *bins* portion of the *Record Tuple*.

  Default `False`.

New in version 3.0.0.

### 1.1.5 aerospike.GeoJSON — GeoJSON Class

#### 1.1.5.1 Overview

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, which allows queries for points inside any given shapes using:

- *geo_within_geojson_region()*
- *geo_within_radius()*

It also enables queries for regions that contain a given point using:

- *geo_contains_geojson_point()*
- *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 a write operation, such as *put()*.

When reading a record from the server, bins with geospatial data will be deserialized into a *GeoJSON* instance.

**See also:**

Geospatial Index and Query.

#### Example

```
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)

# Expected output:
# {"type": "Point", "coordinates": [-80.604333, 28.608389]}

# Alternatively, create the GeoJSON point from a string
loc = aerospike.geojson('{"type": "Point", "coordinates": [-80.604333, 28.608389]}')

# Create and store a user record.
bins = {'pad_id': 1,
        'loc': loc}
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)

# Expected output:
# {'pad_id': 1, 'loc': '{"type": "Point", "coordinates": [-80.604333, 28.608389]}'}

# Cleanup
client.remove(('test', 'pads', 'launchpad1'))
client.close()
```

#### 1.1.5.2 Methods

**class** aerospike.**GeoJSON**

**class GeoJSON**([*geo_data*])

Optionally initializes an object with a GeoJSON `str` or a `dict` of geospatial data.

See *Example* for usage.

**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.

### 1.1.6 aerospike.KeyOrderedDict — KeyOrderedDict Class

**class** aerospike.**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
```

(continues on next page)

(continued from previous page)

```
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': []})

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.

### 1.1.7 aerospike.predicates — Query Predicates

These methods are secondary index filters that can be applied to the *aerospike.Query* class.

#### 1.1.7.1 Bin 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()*.

```
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()*.

```
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()
```

#### 1.1.7.2 GeoJSON Predicates

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

---

```
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",
                 '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

---

```
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))
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

```
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

```
import aerospike
from aerospike import GeoJSON
```

(continues on next page)

(continued from previous page)

```
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.

#### 1.1.7.3 Map and List Predicates

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

---

```
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'␣
```

(continues on next page)

(continued from previous page)

```
↪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

---

```
import aerospike
from aerospike import predicates as p

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()
```

(continues on next page)

(continued from previous page)

```
print(res)
client.close()
```

### 1.1.8 aerospike.exception — Aerospike Exceptions

#### 1.1.8.1 Example

This is a simple example on how to catch an exception thrown by the Aerospike client:

```
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.

#### 1.1.8.2 Base Class

**exception** aerospike.exception.**AerospikeError**

The parent class of all exceptions raised by the Aerospike client.

An exception of this class type must have the following attributes:

**code**

The associated status code.

**msg**

The human-readable error message.

**file**

File where the exception occurred.

**line**

Line in the file where the exception occurred.

**in_doubt**

`True` if it is possible that the operation succeeded. See *In Doubt Status*.

In addition to accessing these attributes by their names, they can also be checked by calling `exc.args[i]`, where `exc` is the exception object and `i` is the index of the attribute in the order they appear above. For example, run `exc.args[4]` to get the `in_doubt` flag.

Inherits from `exceptions.Exception`.

#### 1.1.8.3 Client Errors

**exception** aerospike.exception.**ClientError**

Exception class for client-side errors, often due to misconfiguration or misuse of the API methods.

**exception** aerospike.exception.**InvalidHostError**

Subclass of *ClientError*.

**exception** aerospike.exception.**ParamError**

The operation was not performed because of invalid parameters.

Subclass of *ClientError*.

#### 1.1.8.4 Server Errors

**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 expression was false.

Subclass of *ServerError*.

**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*.

#### 1.1.8.5 Record Errors

**exception** aerospike.exception.**RecordError**

The parent class for record and bin exceptions exceptions associated with read and write operations.

**key**

The key identifying the record.

**bin**

(Optional) the bin associated with the error.

Subclass of *ServerError*.

**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 either a read or a write with the 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 a 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*.

#### 1.1.8.6 Index Errors

**exception** aerospike.exception.**IndexError**

The parent class for indexing exceptions.

**index_name**

The name of the index associated with the error.

Subclass of *ServerError*.

**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*.

#### 1.1.8.7 Query Errors

**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*.

#### 1.1.8.8 Cluster Errors

**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*.

#### 1.1.8.9 Admin Errors

**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*.

#### 1.1.8.10 UDF Errors

**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*.

#### 1.1.8.11 Exception Hierarchy

```
AerospikeError (*)
 +-- TimeoutError (9)
 +-- ClientError (-1)
 |    +-- InvalidHost (-4)
 |    +-- ParamError (-2)
 +-- ServerError (1)
      +-- InvalidRequest (4)
      +-- ServerFull (8)
      +-- AlwaysForbidden (10)
      +-- UnsupportedFeature (16)
      +-- DeviceOverload (18)
      +-- NamespaceNotFound (20)
      +-- ForbiddenError (22)
      +-- ElementNotFoundError (23)
      +-- ElementExistsError (24)
      +-- RecordError (*)
      |    +-- RecordKeyMismatch (19)
      |    +-- RecordNotFound (2)
      |    +-- RecordGenerationError (3)
      |    +-- RecordExistsError (5)
```

(continues on next page)

(continued from previous page)

```
    |    +-- 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)
```

#### 1.1.8.12 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.

### 1.1.9 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.)

#### 1.1.9.1 Subpackages

##### 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** (*str*) – 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** (*str*) – 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** (*str*) – 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** (*str*) – 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: Optional[int] = 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** (*str*) – 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 context operation 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: str*, *value*, *policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a list append operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *values*, *policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a list append items operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *index*, *return_type*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *index*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *rank*, *return_type*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *rank*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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. This should be one of the *List Return Types* values
- **ctx** (*list*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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()* and *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: str*, *value*, *return_type*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value_list*, *return_type*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *return_type*, *value_begin*, *value_end*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list]* = *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.

**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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *operate_ordered()*.The format of the dictionary should be considered an internal detail, and subject to change.

---

**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]
# Found 5
# Rank 0 = 5
# No upper limit, so get all elements >= 5
(5, 0, None) = [5,9,11,15]
# Only get 2 elements
(5, 0, 2) = [5, 9]
```

(continues on next page)

(continued from previous page)

```
# Relative rank -1 = 4
# since 4 is just below 5 in the list
# Then get all elements >= 4
(5, -1, None) = [4, 5, 9, 11, 15]
# Only get 3 elements
(5, -1, 3) = [4, 5, 9]

# 3 not in list
# But next greater value is 4, so set 4's rank to 0
# Then rank 0 = 4, rank 1 = 5, rank 2 = 9, rank 3 = 11
(3,3, None) = [11, 15]
# Rank 0 = 4, Rank -1 = 0
# We will only get down to the lowest rank item
(3,-3, None) = [0, 4,5,9,11,15]
(3, 0, None) = [4,5,9,11,15]
```

aerospike_helpers.operations.list_operations.**list_get_range**(*bin_name: str*, *index*, *count*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *value*, *policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a list increment operation.

The list increment operation increments an item at index: *index* in 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *value*, *policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a list insert operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *values*, *policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a list insert items operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *ctx: Optional[list] = None*)

Creates a list pop operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *count*, *ctx: Optional[list] = None*)

Creates a list pop range operation.

The list pop 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, to be removed and returned
- **ctx** (*list*) – An optional list of nested CDT *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *ctx: Optional[list] = None*)

Creates a 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *return_type*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *rank*, *return_type*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *rank*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value*, *return_type*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value_list*, *return_type*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *return_type*, *value_begin=None*, *value_end=None*, *inverted=False*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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.

**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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *operate_ordered()*.The format of the dictionary should be considered an internal detail, and subject to change.

---

**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]
# We can only go down to lowest rank item
(3,-3, None) = [0,4,5,9,11,15]
(3, 0, None) = [4,5,9,11,15]
```

aerospike_helpers.operations.list_operations.**list_remove_range**(*bin_name: str*, *index*, *count*, *ctx: Optional[list] = None*)

Creates a 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *index*, *value*, *policy: Optional[dict] = None*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *list_order*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *sort_flags: int = 0*, *ctx: Optional[list] = 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** (*int*) – *List Sort Flags* modifying the sorting behavior (default `aerospike.DEFAULT_LIST_SORT`).
- **ctx** (*list*) – An optional list of nested CDT *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *index*, *count*, *ctx: Optional[list] = 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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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 context operation 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: str*, *ctx: Optional[list] = None*)

Creates a map_clear operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *key*, *amount*, *map_policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a map_decrement operation.

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** (*dict*) – Optional *map_policy dictionary* specifies the mode of writing items to the Map, and dictates the map order if there is no Map at the *bin_name*
- **ctx** (*list*) – An optional list of nested CDT *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *index*, *return_type*, *ctx: Optional[list] = None*)

Creates a map_get_by_index operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *index_start*, *get_amt*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_get_by_index_range operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *key*, *return_type*, *ctx: Optional[list] = None*)

Creates a map_get_by_key operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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.

**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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

---

**Note:** This operation requires server version 4.3.0 or greater.

---

Examples for a key ordered map `{0: 6, 6: 12, 10: 18, 15: 24}` and return type of `aerospike.MAP_RETURN_KEY`. See *map_remove_by_key_index_range_relative()* for in-depth explanation.

```
(value, offset, count) = [returned keys]
(5, 0, None) = [6, 10, 15]
(5, 0, 2) = [6, 10]
(5,-1, None) = [0, 6, 10, 15]
```

(continues on next page)

(continued from previous page)

```
(5, -1, 3) = [0, 6, 10]
(3, 2, None) = [15]
(3, 5, None) = []
```

aerospike_helpers.operations.map_operations.**map_get_by_key_list**(*bin_name: str*, *key_list*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_get_by_key_list operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *key_range_start*, *key_range_end*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_get_by_key_range operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *rank*, *return_type*, *ctx: Optional[list] = None*)

Creates a map_get_by_rank operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *rank_start*, *get_amt*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_get_by_rank_range operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *value*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_get_by_value operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *key_list*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_get_by_value_list operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value_start*, *value_end*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_get_by_value_range operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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.

**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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

---

**Note:** This operation requires server version 4.3.0 or greater.

---

Examples for map `{0: 6, 10: 18, 6: 12, 15: 24}` and return type of `aerospike.MAP_RETURN_KEY`. See *map_remove_by_value_rank_range_relative()* for in-depth explanation.

```
(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]
```

aerospike_helpers.operations.map_operations.**map_increment**(*bin_name: str*, *key*, *amount*, *map_policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a map_increment operation.

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** (*dict*) – Optional *map_policy dictionary* specifies the mode of writing items to the Map, and dictates the map order if there is no Map at the *bin_name*
- **ctx** (*list*) – An optional list of nested CDT *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *key*, *value*, *map_policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a map_put operation.

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** (*dict*) – Optional *map_policy dictionary* specifies the mode of writing items to the Map, and dictates the map order if there is no Map at the *bin_name*
- **ctx** (*list*) – An optional list of nested CDT *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *item_dict*, *map_policy: Optional[dict] = None*, *ctx: Optional[list] = None*)

Creates a map_put_items operation.

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** (*dict*) – Optional *map_policy dictionary* specifies the mode of writing items to the Map, and dictates the map order if there is no Map at the *bin_name*
- **ctx** (*list*) – An optional list of nested CDT *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *index*, *return_type*, *ctx: Optional[list] = None*)

Creates a map_remove_by_index operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *index_start*, *remove_amt*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_remove_by_index_range operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *key*, *return_type*, *ctx: Optional[list] = None*)

Creates a map_remove_by_key operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *key*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list] = 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.

**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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

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]

# Next greatest key is 6
# Rank 0 = 6
(5, 0, None) = [6, 10, 15]

# Only delete 2 keys
(5, 0, 2) = [6, 10]

# Rank 0 = 6
# Rank -1 = 0
# Delete elements starting from key 0
(5,-1, None) = [0, 6, 10, 15]
```

(continues on next page)

(continued from previous page)

```
(5, -1, 3) = [0, 6, 10]

# Rank 0 = 6, rank 1 = 10, rank 2 = 15
(3, 2, None) = [15]

# No items with relative rank higher than 2
(3, 5, None) = []
```

aerospike_helpers.operations.map_operations.**map_remove_by_key_list**(*bin_name: str*, *key_list*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_remove_by_key operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *key_range_start*, *key_range_end*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_remove_by_key_range operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *rank*, *return_type*, *ctx: Optional[list] = None*)

Creates a map_remove_by_rank operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *rank_start*, *remove_amt*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_remove_by_rank_range operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_remove_by_value operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value_list*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_remove_by_value_list operation.

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 *cdt_ctx* context operation objects.

**Returns**
A dictionary usable in *operate()* and *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: str*, *value_start*, *value_end*, *return_type*, *inverted=False*, *ctx: Optional[list] = None*)

Creates a map_remove_by_value_range operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *value*, *offset*, *return_type*, *count=None*, *inverted=False*, *ctx: Optional[list]* = *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.

**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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in operate or operate_ordered.The format of the dictionary should be considered an internal detail, and subject to change.

---

**Note:** This operation requires server version 4.3.0 or greater.

---

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]

# Remove and return all keys with values >= 6
(6, 0, None) = [0, 6, 10, 15]

# No key with value 5
```

(continues on next page)

(continued from previous page)

```
# The next greater value is 6
# So rank 0 = 6, and remove and return 2 keys
# starting with the one with value 6
(5, 0, 2) = [0, 6]

# No key with value 7
# The next greater value is 12
# So rank 0 = 12, rank -1 = 6
# We only remove and return one key
(7, -1, 1) = [0]

# Same scenario but remove and return three keys
(7, -1, 3) = [0, 6, 10]
```

aerospike_helpers.operations.map_operations.**map_set_policy**(*bin_name: str*, *policy*, *ctx: Optional[list] = None*)

Creates a map_set_policy_operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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: str*, *ctx: Optional[list] = None*)

Creates a map_size operation.

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 *cdt_ctx* context operation objects.

**Returns**

A dictionary usable in *operate()* and *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.operate()` and `aerospike.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:

```
import aerospike
from aerospike_helpers.operations import bitwise_operations

# Configure the client.
config = {"hosts": [("127.0.0.1", 3000)]}
# Create a client and connect it to the cluster.
client = aerospike.client(config).connect()

key = ("test", "demo", "foo")
five_ones_bin_name = "bitwise1"
five_one_blob = bytearray([1] * 5)
# Write the record.
client.put(key, {five_ones_bin_name: five_one_blob})

# EXAMPLE 1: resize the five_ones bin to a bytesize of 10.

ops = [bitwise_operations.bit_resize(five_ones_bin_name, 10)]

_, _, bins = client.get(key)
print("5 bytes: ", bins)
# 5 bytes:  {'bitwise1': b''}

_, _, _ = client.operate(key, ops)

_, _, newbins = client.get(key)
print("After resize to 10 bytes: ", newbins)
# After resize to 10 bytes:  {'bitwise1': b''}

# EXAMPLE 2: shrink the five_ones bin to a bytesize of 5 from the front.

ops = [
    bitwise_operations.bit_resize(
        five_ones_bin_name, 5, resize_flags=aerospike.BIT_RESIZE_FROM_FRONT
    )
]

_, _, _ = client.operate(key, ops)
_, _, newbins = client.get(key)
print("After resize to 5 bytes again: ", newbins)
# After resize to 5 bytes again:  {'bitwise1': b''}

# Cleanup and close the connection to the Aerospike cluster.
client.remove(key)
client.close()
```

Example:

```
import aerospike
from aerospike import exception as e
from aerospike_helpers.operations import bitwise_operations

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

key = ('test', 'demo', 'bit_example')
five_one_blob = bytearray([1] * 5)
five_one_bin = 'bitwise1'

# Precleanup
_, meta = client.exists(key)
if meta != None:
    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)
]
_, _, results = client.operate(key, ops)
print(results)
# {'bitwise1': b'{{{{{'}

# Example 2: modify bits using the 'or' op, then read bits
# 0 = offset
# 8 = num bits to OR
# value to OR: 0xff
# value size is 1 byte
ops = [
    bitwise_operations.bit_or(five_one_bin, 0, 8, 1, bytearray([255]), bit_policy),
    bitwise_operations.bit_get(five_one_bin, 0, 40)
]
_, _, results = client.operate(key, ops)
print(results)
# {'bitwise1': b'ÿ{{{{'}

# Example 3: modify bits using the 'remove' op, then read bits'
# offset = 0
# # bytes to remove = 2
# Number of bits remaining = 24
ops = [
    bitwise_operations.bit_remove(five_one_bin, 0, 2, bit_policy),
    bitwise_operations.bit_get(five_one_bin, 0, 24)
]
_, _, results = client.operate(key, ops)
```

(continues on next page)

(continued from previous page)

```
print(results)
# {'bitwise1': b'{{{'}

client.close()
```

**See also:**

Bits (Data Types).

aerospike_helpers.operations.bitwise_operations.**bit_add**(*bin_name: str*, *bit_offset*, *bit_size*, *value*, *sign*, *action*, *policy=None*)

Creates a bit_add_operation.

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*) – The *bit_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: str*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_and_operation.

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, bytearray*) – Bytes to be used in and operation.
- **policy** (*dict*) – The *bit_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: str*, *bit_offset*, *bit_size*)

Creates a bit_count_operation to be used with `aerospike.operate()` or `aerospike.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: str*, *bit_offset*, *bit_size*)

Creates a bit_get_operation.

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: str*, *bit_offset*, *bit_size*, *sign*)

Creates a bit_get_int_operation.

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: str*, *byte_offset*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_insert_operation.

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, bytearray*) – The value to be inserted.
- **policy** (*dict*) – The *bit_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: str*, *bit_offset*, *bit_size*, *value*)

Creates a bit_lscan_operation.

Server returns an integer representing the bit offset of the first occurrence 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: str*, *bit_offset*, *bit_size*, *shift*, *policy=None*)

Creates a bit_lshift_operation.

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*) – The *bit_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: str*, *bit_offset*, *bit_size*, *policy=None*)

Creates a bit_not_operation.

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*) – The *bit_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: str*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_or_operation.

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*) – The *bit_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: str*, *byte_offset*, *byte_size*, *policy=None*)

Creates a bit_remove_operation to be used with `aerospike.operate()` or `aerospike.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*) – The *bit_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: str*, *byte_size*, *policy=None*, *resize_flags: int = 0*)

Creates a bit_resize_operation to be used with `aerospike.operate()` or `aerospike.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*) – The *bit_policy* dictionary. default: None.

- **resize_flags** (*int*) – *Bitwise Resize Flags* modifying the resize behavior (default `aerospike.BIT_RESIZE_DEFAULT`), such as `aerospike.BIT_RESIZE_GROW_ONLY | aerospike.BIT_RESIZE_FROM_FRONT`.

**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: str*, *bit_offset*, *bit_size*, *value*)

Creates a bit_rscan_operation.

Server returns an integer representing the bit offset of the last occurrence 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: str*, *bit_offset*, *bit_size*, *shift*, *policy=None*)

Creates a bit_rshift_operation.

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*) – The *bit_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: str*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_set_operation to be used with `aerospike.operate()` or `aerospike.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, bytearray*) – The value to be set.
- **policy** (*dict*) – The *bit_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: str*, *bit_offset*, *bit_size*, *value*, *sign*, *action*, *policy=None*)

Creates a bit_subtract_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*) – The *bit_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: str*, *bit_offset*, *bit_size*, *value_byte_size*, *value*, *policy=None*)

Creates a bit_xor_operation.

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*) – The *bit_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:

```
import aerospike
from aerospike_helpers.operations import hll_operations as hll_ops
from aerospike_helpers.operations import operations

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.
client = aerospike.client(config).connect()

# Create customer keys
TEST_NS = "test"
TEST_SET = "demo"
customerNames = ["Amy", "Farnsworth", "Scruffy"]
keys = []
for customer in customerNames:
    keys.append((TEST_NS, TEST_SET, customer))

itemsViewedPerCustomer = [
    # [item1, item2, ... item500]
    list("item%s" % str(i) for i in range(0, 500)), # Amy
    list("item%s" % str(i) for i in range(0, 750)), # Farnsworth
    list("item%s" % str(i) for i in range(250, 1000)), # Scruffy
]

for key, itemsViewed in zip(keys, itemsViewedPerCustomer):
    customerName = key[2]
    ops = [
        operations.write("name", customerName),
        hll_ops.hll_add("viewed", itemsViewed, NUM_INDEX_BITS, NUM_MH_BITS),
    ]
    client.operate(key, ops)

# Find out how many items viewed Amy, Farnsworth, and Scruffy have in common.
farnsworthRecord = client.get(keys[1])
```

(continues on next page)

(continued from previous page)

```
scruffyRecord = client.get(keys[2])
farnsworthViewedItems = farnsworthRecord[2]["viewed"]
scruffyViewedItems = scruffyRecord[2]["viewed"]
viewed = [farnsworthViewedItems, scruffyViewedItems]
ops = [
    hll_ops.hll_get_intersect_count("viewed", viewed)
]
# Pass in Amy's key
_, _, res = client.operate(keys[0], ops)
print("Estimated items viewed intersection:", res["viewed"])
# Estimated items viewed intersection: 251
# Actual intersection: 250

# Find out how many unique products Amy, Farnsworth, and Scruffy have viewed.
ops = [hll_ops.hll_get_union_count("viewed", viewed)]
_, _, res = client.operate(keys[0], ops)

print("Estimated items viewed union:", res["viewed"])
# Estimated items viewed union: 1010
# Actual union: 1000

# Find the similarity of Amy, Farnsworth, and Scruffy's product views.
ops = [hll_ops.hll_get_similarity("viewed", viewed)]
_, _, res = client.operate(keys[0], ops)

print("Estimated items viewed similarity: %f%%" % (res["viewed"] * 100))
# Estimated items viewed similarity: 24.888393%
# Actual similarity: 25%
```

aerospike_helpers.operations.hll_operations.**hll_add**(*bin_name: str*, *values*, *index_bit_count=None*, *mh_bit_count=None*, *policy=None*)

Creates a hll_add operation.

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.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**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 between 4 and 16 inclusive.
- **mh_bit_count** – An optional number of min hash bits. Must be between 4 and 58 inclusive.
- **policy** (*dict*) – An optional dictionary of *HyperLogLog policies*.

aerospike_helpers.operations.hll_operations.**hll_describe**(*bin_name*)

Creates a hll_describe operation.

Server returns index and minhash bit counts used to create HLL bin in a list of integers. The list size is 2.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**Parameters**
**bin_name** (*str*) – The name of the bin to be operated on.

aerospike_helpers.operations.hll_operations.**hll_fold**(*bin_name: str*, *index_bit_count*)

Creates a hll_fold operation.

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.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**Parameters**

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index_bit_count** – number of index bits. Must be between 4 and 16 inclusive.

aerospike_helpers.operations.hll_operations.**hll_get_count**(*bin_name*)

Creates a hll_get_count operation.

Server returns estimated count of elements in the HLL bin.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**Parameters**
**bin_name** (*str*) – The name of the bin to be operated on.

aerospike_helpers.operations.hll_operations.**hll_get_intersect_count**(*bin_name: str*, *hll_list*)

Creates a hll_get_intersect_count operation.

Server returns estimate of elements that would be contained by the intersection of these HLL objects.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**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: str*, *hll_list*)

Creates a hll_get_similarity operation.

Server returns estimated similarity of the HLL objects. Server returns a float.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**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: str*, *hll_list*)

Creates a hll_get_union operation.

Server returns an HLL object that is the union of all specified HLL objects in hll_list with the HLL bin.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**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: str*, *hll_list*)

Creates a hll_get_union_count operation.

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.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**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: str*, *index_bit_count=None*, *mh_bit_count=None*, *policy=None*)

Creates a hll_init operation.

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.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**Parameters**

- **bin_name** (*str*) – The name of the bin to be operated on.
- **index_bit_count** – An optional number of index bits. Must be between 4 and 16 inclusive.
- **mh_bit_count** – An optional number of min hash bits. Must be between 4 and 58 inclusive.
- **policy** (*dict*) – An optional dictionary of *HyperLogLog policies*.

aerospike_helpers.operations.hll_operations.**hll_refresh_count**(*bin_name: str*)

Creates a hll_refresh_count operation.

Server updates the cached count if it is stale. Server returns the count.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*. :param bin_name: The name of the bin to be operated on. :type bin_name: str

aerospike_helpers.operations.hll_operations.**hll_set_union**(*bin_name: str*, *hll_list*, *policy=None*)

Creates a hll_set_union operation.

Server sets the union of all specified HLL objects with the HLL bin. Server returns nothing.

Returns a dictionary to be used with *aerospike.Client.operate()* and *aerospike.Client.operate_ordered()*.

**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 *HyperLogLog policies*.

### 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: str*, *expression: _BaseExpr*, *expression_read_flags: int = 0*)

Create an expression read operation dictionary.

Reads and returns the value produced by the evaluated expression.

**Parameters**

- **bin_name** (*str*) – 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 *aerospike_helpers.expressions package*.
- **expression_read_flags** (*int*) – *Read Expression Flags* (default `aerospike.EXP_READ_DEFAULT`)

**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: str*, *expression: _BaseExpr*, *expression_write_flags: int = 0*)

Create an expression write operation dictionary.

Writes the value produced by the evaluated expression to the supplied bin.

**Parameters**

- **bin_name** (*str*) – The name of the bin to write to.
- **expression** – A compiled Aerospike expression, see *aerospike_helpers.expressions package*.
- **expression_write_flags** (*int*) – *Write Expression Flags* such as `aerospike.EXP_WRITE_UPDATE_ONLY | aerospike.EXP_WRITE_POLICY_NO_FAIL` (default `aerospike.EXP_WRITE_DEFAULT`).

**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}
```

## aerospike_helpers.expressions package

Classes for the creation and use of Aerospike expressions.

### Overview

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.

**Note:** See Aerospike Expressions.

In the Python client, Aerospike expressions are built using a series of classes that represent comparison 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.

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, these transactions will filter the results.

Example:

```
import aerospike
from aerospike_helpers import expressions as exp

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

# Write player records to database
keys = [("test", "demo", i) for i in range(1, 5)]
records = [
            {'user': "Chief"  , 'scores': [6, 12, 4, 21], 'kd': 1.2},
            {'user': "Arbiter", 'scores': [5, 10, 5, 8] , 'kd': 1.0},
            {'user': "Johnson", 'scores': [8, 17, 20, 5], 'kd': 0.9},
            {'user': "Regret" , 'scores': [4, 2, 3, 5]  , 'kd': 0.3}
        ]
for key, record in zip(keys, records):
    client.put(key, record)

# Example #1: Get players with a K/D ratio >= 1.0

kdGreaterThan1 = exp.GE(exp.FloatBin("kd"), 1.0).compile()
policy = {"expressions": kdGreaterThan1}
# For more details on get_many() usage, see the documentation
records = client.get_many(keys, policy)

for record in records:
    print(record[2])
# {'user': 'Chief', 'scores': [6, 12, 4, 21], 'kd': 1.2}
# {'user': 'Arbiter', 'scores': [5, 10, 5, 8], 'kd': 1.0}
# None
# None

# Example #2: Get player with scores higher than 20
# By nesting expressions, we can create complicated filters

# Get top score
getTopScore = exp.ListGetByRank(
                None,
                aerospike.LIST_RETURN_VALUE,
                exp.ResultType.INTEGER,
                -1,
                exp.ListBin("scores")
                )
# ...then compare it
```

(continues on next page)

(continued from previous page)

```
scoreHigherThan20 = exp.GE(getTopScore, 20).compile()
policy = {"expressions": scoreHigherThan20}
records = client.get_many(keys, policy)

for record in records:
    print(record[2])
# {'user': 'Chief', 'scores': [6, 12, 4, 21], 'kd': 1.2}
# None
# {'user': 'Johnson', 'scores': [8, 17, 20, 5], 'kd': 0.9}
# None
```

Currently, Aerospike expressions are supported for: - Record operations - Batch operations - Transactions - UDF apply methods (apply, scan apply, and query apply) - Query invoke methods (foreach, results, execute background) - Scan invoke methods (same as query invoke methods)

### Filter Behavior

This section describes the behavior of methods when a record is filtered out by an expression.

**For:**

- Record operations
- Numeric operations
- String operations
- Single record transactions

An exception *FilteredOut* is thrown.

**For:**

- *get_many()*
- *exists_many()*
- *select_many()*

The filtered out record's `meta` and `bins` are both set to `None` .

For:

- *batch_write()* (records filtered out by a batch or batch record policy)
- *batch_operate()* (records filtered out by a batch or batch write policy)
- *batch_apply()* (records filtered out by a batch or batch apply policy)

The filtered out record's:

- `BatchRecord.record` is set to `None`
- `BatchRecord.result` is set to `27`

For *batch_get_ops()*, the filtered out record's:

- `meta` is set to *FilteredOut*.
- `bins` is set to `None`.

### Terminology

Aerospike expressions are evaluated server side, and expressions used for filtering are called **filter expressions**. They 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 list during evaluation 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 that an expression returns an **expression**, this means that the data type returned may vary (usually depending on the `return_type` parameter).

---

**Note:** Currently, Aerospike expressions for the python client do not support comparing `as_python_bytes` blobs.

Comparisons between constant map values and map expressions are also unsupported.

---

### Expression Type Aliases

The following documentation uses type aliases that map to standard Python types.

Table 1: Aliases to Standard Python Types

| Alias | Type |
|---|---|
| AerospikeExpression | _BaseExpr |
| 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] |
| TypeCTX | 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

Assume all in-line examples run this code beforehand:

```
import aerospike
import aerospike_helpers.expressions as exp
```

### aerospike_helpers.expressions.base module

The expressions base module provide expressions for:

- declaring variables, using variables, and control-flow
- comparison operators
- applying logical operators to one or more 'boolean expressions'
- returning the value of (in-memory) record metadata
- returning the value from storage, such as bin data or the record's key

**class** aerospike_helpers.expressions.base.**And**(**exprs: _BaseExpr*)

Create an "and" operator that applies to a variable amount of expressions.

**__init__**(**exprs: _BaseExpr*)

**Parameters**
***exprs** (*_BaseExpr*) – Variable amount of expressions to be ANDed together.

**Returns**
(boolean value)

Example:

```
# (a > 5 || a == 0) && b < 3
expr = exp.And(
        exp.Or(
          exp.GT(exp.IntBin("a"), 5),
          exp.Eq(exp.IntBin("a"), 0)),
        exp.LT(exp.IntBin("b"), 3)).compile()
```

**class** aerospike_helpers.expressions.base.**BinExists**(*bin: str*)

Create an expression that returns True if bin exists.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – bin name.

**Returns**
(boolean value): True if bin exists, False otherwise.

Example:

```
#Bin "a" exists in record.
expr = exp.BinExists("a").compile()
```

**class** aerospike_helpers.expressions.base.**BinType**(*bin: str*)

Create an expression that returns the type of a bin as one of the aerospike *bin types*

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – bin name.

**Returns**
(integer value): returns the bin type.

Example:

```
# bin "a" == type string.
expr = exp.Eq(exp.BinType("a"), aerospike.AS_BYTES_STRING).compile()
```

**class** aerospike_helpers.expressions.base.**BlobBin**(*bin: str*)

Create an expression that returns a bin as a blob. Returns the unknown-value if the bin is not a blob.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(blob bin)

Example:

```
#. Blob bin "a" == bytearray([0x65, 0x65])
expr = exp.Eq(exp.BlobBin("a"), bytearray([0x65, 0x65])).compile()
```

**class** aerospike_helpers.expressions.base.**BoolBin**(*bin: str*)

Create an expression that returns a bin as a boolean. Returns the unknown-value if the bin is not a boolean.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(boolean bin)

Example:

```
# Boolean bin "a" is True.
expr = exp.BoolBin("a").compile()
```

**class** aerospike_helpers.expressions.base.**CmpGeo**(*expr0: TypeGeo*, *expr1: TypeGeo*)

Create a point within region or region contains point expression.

**__init__**(*expr0: TypeGeo*, *expr1: TypeGeo*)

**Parameters**

- **expr0** (*TypeGeo*) – Left expression in comparison.
- **expr1** (*TypeGeo*) – Right expression in comparison.

**Returns**
(boolean value)

Example:

```
# Geo bin "point" is within geo bin "region".
expr = exp.CmpGeo(exp.GeoBin("point"), exp.GeoBin("region")).compile()
```

class aerospike_helpers.expressions.base.**CmpRegex**(*options: int*, *regex_str: str*, *cmp_str: Union[_BaseExpr, str]*)

Create an expression that performs a regex match on a string bin or value expression.

**__init__**(*options: int*, *regex_str: str*, *cmp_str: Union[_BaseExpr, str]*)

**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.
import aerospike
expr = exp.CmpRegex(aerospike.REGEX_ICASE | aerospike.REGEX_NEWLINE, "prefix.
↪*suffix",             exp.StrBin("a")).compile()
```

class aerospike_helpers.expressions.base.**Cond**(**exprs: _BaseExpr*)

Conditionally select an expression from a variable number of condition/action pairs, followed by a default expression action.

Takes a set of test-expression/action-expression pairs and evaluates each test expression, one at a time. If a test returns `True`, `Cond` evaluates the corresponding action expression and returns its value, after which `Cond` doesn't evaluate any of the other tests or expressions. If all tests evaluate to `False`, the default action expression is evaluated and returned.

`Cond` is strictly typed, so all actions-expressions must evaluate to the same type or the *Unknown* expression.

Requires server version 5.6.0+.

**__init__**(**exprs: _BaseExpr*)

**Parameters**
***exprs** (*_BaseExpr*) – bool exp1, action exp1, bool exp2, action exp2, ..., action-default

**Returns**
(boolean value)

Example:

```
from aerospike_helpers.expressions.arithmetic import Add, Sub, Mul

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

(continues on next page)

(continued from previous page)

```
# Create a client and connect it to the cluster
client = aerospike.client(config).connect()
client.truncate('test', "demo", 0)

# Store 2 bin integers and use expressions to perform arithmetic
# Results will only be calculated and returned, not stored

keyTuple = ('test', 'demo', 'key')
client.put(keyTuple, {"operation": "add", "val1": 40, "val2": 30})

# Determine operation to perform
# If operation is unknown, return -1
expr = exp.Cond(
            exp.Eq(exp.StrBin("operation"), "add"),
                Add(exp.IntBin("val1"), exp.IntBin("val2")),
            exp.Eq(exp.StrBin("operation"), "subtract"),
                Sub(exp.IntBin("val1"), exp.IntBin("val2")),
            exp.Eq(exp.StrBin("operation"), "multiply"),
                Mul(exp.IntBin("val1"), exp.IntBin("val2")),
-1).compile()

from aerospike_helpers.operations import expression_operations as expr_ops
ops = [
    # Bin "results" doesn't actually exist in the server
    # The name is only used to return the results
    expr_ops.expression_read("results", expr)
]
record = client.operate(keyTuple, ops)
print(record)
# (('test', 'demo', 'key', bytearray(b'...')), {'ttl': 2592000, 'gen': 1}, {'results':␣
↪70})

client.put(keyTuple, {"operation": "divide"})

record = client.operate(keyTuple, ops)
print(record)
# Divide isn't supported, so we get -1
# (('test', 'demo', 'key', bytearray(b'...')), {'ttl': 2592000, 'gen': 2}, {'results': -
↪1})
```

**class** aerospike_helpers.expressions.base.**Def**(*var_name: str*, *expr: _BaseExpr*)

Assign variable to an expression that can be accessed later with *Var*. Requires server version 5.6.0+.

**__init__**(*var_name: str*, *expr: _BaseExpr*)

**Parameters**

- **var_name** (*str*) – Variable name.
- **expr** (*_BaseExpr*) – Variable is set to result of this expression.

**Returns**

(a variable name expression pair)

Example:

```
# for int bin "a", 5 < a < 10
expr = exp.Let(exp.Def("x", exp.IntBin("a")),
        exp.And(
            exp.LT(5, exp.Var("x")),
            exp.LT(exp.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__**()

**Returns**
(integer value): Uncompressed storage size of the record.

Example:

```
# Record device size >= 100 KB.
expr = exp.GE(exp.DeviceSize(), 100 * 1024).compile()
```

**class** aerospike_helpers.expressions.base.**DigestMod**(*mod: int*)

Create an expression that returns record digest modulo as integer.

**__init__**(*mod: int*)

**Parameters**
**mod** (*int*) – Divisor used to divide the digest to get a remainder.

**Returns**
(integer value): Value in range 0 and mod (exclusive).

Example:

```
# Records that have digest(key) % 3 == 1.
expr = exp.Eq(exp.DigestMod(3), 1).compile()
```

**class** aerospike_helpers.expressions.base.**Eq**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

Create an equals, (==) expression.

**__init__**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

**Parameters**

- **expr0** (*TypeComparisonArg*) – Left argument to ==.
- **expr1** (*TypeComparisonArg*) – Right argument to ==.

**Returns**
(boolean value)

Example:

```
# Integer bin "a" == 11
expr = exp.Eq(exp.IntBin("a"), 11).compile()
```

**class** aerospike_helpers.expressions.base.**Exclusive**(**exprs: _BaseExpr*)

Create an expression that returns True if only one of the expressions are True.

__init__(*exprs: _BaseExpr)

**Parameters**
***exprs** (*_BaseExpr*) – Variable amount of expressions to be checked.

**Returns**
(boolean value)

Example:

```
# exclusive(a == 0, b == 0)
expr = exp.Exclusive(
                exp.Eq(exp.IntBin("a"), 0),
                exp.Eq(exp.IntBin("b"), 0)).compile()
```

class aerospike_helpers.expressions.base.**FloatBin**(*bin: str*)

Create an expression that returns a bin as a float. Returns the unknown-value if the bin is not a float.

__init__(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(float bin)

Example:

```
# Float bin "a" > 2.71.
expr = exp.GT(exp.FloatBin("a"), 2.71).compile()
```

class aerospike_helpers.expressions.base.**GE**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

Create a greater than or equal to (>=) expression.

__init__(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

**Parameters**

- **expr0** (*TypeComparisonArg*) – Left argument to >=.
- **expr1** (*TypeComparisonArg*) – Right argument to >=.

**Returns**
(boolean value)

Example:

```
# Integer bin "a" >= 88.
expr = exp.GE(exp.IntBin("a"), 88).compile()
```

class aerospike_helpers.expressions.base.**GT**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

Create a greater than (>) expression.

__init__(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

**Parameters**

- **expr0** (*TypeComparisonArg*) – Left argument to >.
- **expr1** (*TypeComparisonArg*) – Right argument to >.

**Returns**
(boolean value)

Example:

```
# Integer bin "a" > 8.
expr = exp.GT(exp.IntBin("a"), 8).compile()
```

**class** aerospike_helpers.expressions.base.**GeoBin**(*bin: str*)

Create an expression that returns a bin as a geojson. Returns the unknown-value if the bin is not a geojson.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(geojson bin)

Example:

```
#GeoJSON bin "a" contained by GeoJSON bin "b".
expr = exp.CmpGeo(GeoBin("a"), exp.GeoBin("b")).compile()
```

**class** aerospike_helpers.expressions.base.**HLLBin**(*bin: str*)

Create an expression that returns a bin as a HyperLogLog. Returns the unknown-value if the bin is not a HyperLogLog.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(HyperLogLog bin)

Example:

```
# Does HLL bin "a" have a hll_count > 1000000.
from aerospike_helpers.expressions import hll
count = hll.HLLGetCount(hll.HLLBin("a"))
expr = exp.GT(count, 1000000).compile()
```

**class** aerospike_helpers.expressions.base.**IntBin**(*bin: str*)

Create an expression that returns a bin as an integer. Returns the unknown-value if the bin is not an integer.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(integer bin)

Example:

```
# Integer bin "a" == 200.
expr = exp.Eq(exp.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__**()

**Returns**
(boolean value): True if the record is a tombstone, false otherwise.

Example:

```
# Detect deleted records that are in tombstone state.
expr = exp.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__**()

**Returns**
(blob value): Blob value of the key if the key is a blob.

Example:

```
# blob record key <= bytearray([0x65, 0x65]).
expr = exp.GE(exp.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__**()

**Returns**
(boolean value): True if the record has a stored key, false otherwise.

Example:

```
# Key exists in record meta data.
expr = exp.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__**()

**Returns**
(integer value): Integer value of the key if the key is an integer.

Example:

```
# Integer record key >= 10000.
expr = exp.GE(exp.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__()

**Returns**
(string value): string value of the key if the key is an string.

Example:

```
# string record key == "aaa".
expr = exp.Eq(exp.KeyStr(), "aaa").compile()
```

class aerospike_helpers.expressions.base.**LE**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

Create a less than or equal to (<=) expression.

__init__(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

**Parameters**

- **expr0** (*TypeComparisonArg*) – Left argument to <=.
- **expr1** (*TypeComparisonArg*) – Right argument to <=.

**Returns**
(boolean value)

Example:

```
# Integer bin "a" <= 1.
expr = exp.LE(exp.IntBin("a"), 1).compile()
```

class aerospike_helpers.expressions.base.**LT**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

Create a less than (<) expression.

__init__(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

**Parameters**

- **expr0** (*TypeComparisonArg*) – Left argument to <.
- **expr1** (*TypeComparisonArg*) – Right argument to <.

**Returns**
(boolean value)

Example:

```
# Integer bin "a" < 1000.
expr = exp.LT(exp.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__()

**Returns**
(integer value): When the record was last updated.

Example:

```
# Record last update time >= 2020-01-15.
expr = exp.GE(exp.LastUpdateTime(), 1577836800).compile()
```

**class** aerospike_helpers.expressions.base.**Let**(**exprs: _BaseExpr*)

Defines variables to be used within the `Let` expression's scope. The last argument can be any expression and should make use of the defined variables. The `Let` 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: _BaseExpr*)

**Parameters**

- ***exprs** (*_BaseExpr*) – Variable number of *Def* expressions
- **expression.** (*followed by a scoped*) –

**Returns**
(result of scoped expression)

Example:

```
# for int bin "a", 5 < a < 10
expr = exp.Let(exp.Def("x", exp.IntBin("a")),
    exp.And(
        exp.GT(5, exp.Var("x")),
        exp.LT(exp.Var("x"), 10))).compile()
```

**class** aerospike_helpers.expressions.base.**ListBin**(*bin: str*)

Create an expression that returns a bin as a list. Returns the unknown-value if the bin is not a list.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(list bin)

Example:

```
from aerospike_helpers.expressions import list as list_exprs

# Check that list bin "listBin" contains at least one item with value 42.
list42Count = list_exprs.ListGetByValue(
    ctx=None,
    return_type=aerospike.LIST_RETURN_COUNT,
    value=42,
    bin=exp.ListBin("listBin")
)
expr = exp.GT(list42Count, 0).compile()
```

**class** aerospike_helpers.expressions.base.**MapBin**(*bin: str*)

Create an expression that returns a bin as a map. Returns the unknown-value if the bin is not a map.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(map bin)

Example:

```
from aerospike_helpers.expressions import map as map_exprs

# Map bin "a" size > 7.
expr = exp.GT(map_exprs.MapSize(None, exp.MapBin("a")), 7).compile()
```

**class** aerospike_helpers.expressions.base.**NE**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

Create a not equals (not ==) expressions.

**__init__**(*expr0: TypeComparisonArg*, *expr1: TypeComparisonArg*)

**Parameters**

- **expr0** (*TypeComparisonArg*) – Left argument to *not* ==.
- **expr1** (*TypeComparisonArg*) – Right argument to *not* ==.

**Returns**
(boolean value)

Example:

```
# Integer bin "a" not == 13.
expr = exp.NE(exp.IntBin("a"), 13).compile()
```

**class** aerospike_helpers.expressions.base.**Not**(**exprs: _BaseExpr*)

Create a "not" (not) operator expression.

**__init__**(**exprs: _BaseExpr*)

**Parameters**
***exprs** (*_BaseExpr*) – Variable amount of expressions to be negated.

**Returns**
(boolean value)

Example:

```
# not (a == 0 or a == 10)
expr = exp.Not(exp.Or(
            exp.Eq(exp.IntBin("a"), 0),
            exp.Eq(exp.IntBin("a"), 10))).compile()
```

**class** aerospike_helpers.expressions.base.**Or**(**exprs: _BaseExpr*)

Create an "or" operator that applies to a variable amount of expressions.

**__init__**(**exprs: _BaseExpr*)

**Parameters**
***exprs** (*_BaseExpr*) – Variable amount of expressions to be ORed together.

**Returns**
(boolean value)

Example:

```
# (a == 0 || b == 0)
expr = exp.Or(
        exp.Eq(exp.IntBin("a"), 0),
        exp.Eq(exp.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__**()

**Returns**
(string value): Name of the set this record belongs to.

Example:

```
# Record set name == "myset".
expr = exp.Eq(exp.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__**()

**Returns**
(integer value): Number of milliseconds since last updated.

Example:

```
# Record last updated more than 2 hours ago.
expr = exp.GT(exp.SinceUpdateTime(), 2 * 60 * 60 * 1000).compile()
```

**class** aerospike_helpers.expressions.base.**StrBin**(*bin: str*)

Create an expression that returns a bin as a string. Returns the unknown-value if the bin is not a string.

**__init__**(*bin: str*)

**Parameters**
**bin** (*str*) – Bin name.

**Returns**
(string bin)

Example:

```
# String bin "a" == "xyz".
expr = exp.Eq(exp.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__**()

**Returns**
(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 = exp.LT(exp.TTL(), 60 * 60).compile()
```

**class** aerospike_helpers.expressions.base.**Unknown**

Create an 'Unknown' expression, which allows an operation expression ('read expression' or 'write expression') to be aborted.

This expression returns a special 'unknown' trilean value which, when returned by an operation expression, will result in an error code 26 *OpNotApplicable*. These failures can be ignored with the policy flags *aerospike.EXP_READ_EVAL_NO_FAIL* for read expressions and *aerospike.EXP_WRITE_EVAL_NO_FAIL* for write expressions. This would then allow subsequent operations in the transaction to proceed.

This expression is only useful from a *Cond* conditional expression within an operation expression, and should be avoided in filter-expressions, where it might trigger an undesired move into the storage-data phase.

If a test-expression within the Cond yields the special 'unknown' trilean value, then the Cond will also immediately yield the 'unknown' value and further test-expressions will not be evaluated.

Note that this special 'unknown' trilean value is the same value returned by any failed expression.

**__init__**()

**Returns**
(unknown value)

Example:

```
from aerospike_helpers.expressions.arithmetic import Add

# Declare variable for balance bin
exp.Let(exp.Def("bal", exp.IntBin("balance")),
    # If IntBin("balance") >= 50, get "balance" + 50.
    exp.Cond(
        exp.GE(exp.Var("bal"), 50),
            Add(exp.Var("bal"), 50),
        # Otherwise, fail the expression via Unknown().
        exp.Unknown()
    )
)
```

**class** aerospike_helpers.expressions.base.**Var**(*var_name: str*)

Retrieve expression value from a variable previously defined with *Def*. Requires server version 5.6.0+.

**__init__**(*var_name: str*)

**Parameters**
**var_name** (*str*) – Variable name.

**Returns**
(value stored in variable)

Example:

```
# for int bin "a", 5 < a < 10
expr = exp.Let(exp.Def("x", exp.IntBin("a")),
        exp.And(
            exp.LT(5, exp.Var("x")),
            exp.LT(exp.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__()

**Returns**
(integer value): Expiration time in nanoseconds since 1970-01-01.

Example:

```
# Record expires on 2021-01-01.
expr = exp.And(
        exp.GE(exp.VoidTime(), 1609459200),
        exp.LT(exp.VoidTime(), 1609545600)).compile()
```

### aerospike_helpers.expressions.list module

List expressions contain expressions for reading and modifying Lists. Most of these operations are from the standard *List API*.

**class** aerospike_helpers.expressions.list.**ListAppend**(*ctx: TypeCTX*, *policy: TypePolicy*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that appends value to end of list.

__init__(*ctx: TypeCTX*, *policy: TypePolicy*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *List policies*.
- **value** (*TypeValue*) – Value or value expression to append to list.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
List expression.

Example:

```
# Check if length of list bin "a" is > 5 after appending 1 item.
listAppendedBy3 = exp.ListAppend(None, None, 3, exp.ListBin("a"))
expr = exp.GT(exp.ListSize(None, listAppendedBy3), 5).compile()
```

**class** aerospike_helpers.expressions.list.**ListAppendItems**(*ctx: TypeCTX*, *policy: TypePolicy*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that appends a list of items to the end of a list.

__init__(*ctx: TypeCTX*, *policy: TypePolicy*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *List policies*.
- **value** (*TypeValue*) – List or list expression of items to be appended.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
List expression.

Example:

```
# Check if length of list bin "a" is > 5 after appending multiple items.
listAppendedByTwoItems = exp.ListAppendItems(None, None, [3, 2], exp.ListBin("a
↪"))
expr = exp.GT(
        exp.ListSize(None, listAppendedByTwoItems),
        5).compile()
```

**class** aerospike_helpers.expressions.list.**ListClear**(*ctx: TypeCTX*, *bin: TypeBinName*)

Create an expression that removes all items in a list.

**__init__**(*ctx: TypeCTX*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
List expression.

Example:

```
# Clear list value of list nested in list bin "a" index 1.
from aerospike_helpers import cdt_ctx
ctx = [cdt_ctx.cdt_ctx_list_index(1)]
expr = exp.ListClear(ctx, "a").compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByIndex**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that selects list item identified by index and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the value at index 0 in list bin "a". (assume this value is an integer)
expr = exp.ListGetByIndex(None, aerospike.LIST_RETURN_VALUE, ResultType.INTEGER,
↪ 0,
    exp.ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByIndexRange**(*ctx: TypeCTX*, *return_type: int*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that selects "count" list items starting at specified index and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get elements at indexes 3, 4, 5, 6 in list bin "a".
expr = exp.ListGetByIndexRange(None, aerospike.LIST_RETURN_VALUE, 3, 4, exp.
↪ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByIndexRangeToEnd**(*ctx: TypeCTX*, *return_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

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: TypeCTX*, *return_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **index** (*TypeIndex*) – Integer or integer expression of index to start getting elements at.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get element 5 to end from list bin "a".
expr = exp.ListGetByIndexRangeToEnd(None, aerospike.LIST_RETURN_VALUE, 5, exp.
↪ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByRank**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that selects list item identified by rank and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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).
- **rank** (*TypeRank*) – Rank integer or integer expression of element to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
from aerospike_helpers.expressions.resources import ResultType
# Get the smallest element in list bin "a".
expr = exp.ListGetByRank(None, aerospike.LIST_RETURN_VALUE, ResultType.INTEGER,␣
↪0,
    exp.ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByRankRange**(*ctx: TypeCTX*, *return_type: int*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that selects "count" list items starting at specified rank and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the 3 smallest elements in list bin "a".
expr = exp.ListGetByRankRange(None, aerospike.LIST_RETURN_VALUE, 0, 3, exp.
↪ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByRankRangeToEnd**(*ctx: TypeCTX*, *return_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

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: TypeCTX*, *return_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **rank** (*TypeRank*) – Rank integer or integer expression of first element to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the three largest elements in list bin "a".
expr = exp.ListGetByRankRangeToEnd(None, aerospike.LIST_RETURN_VALUE, -3, exp.
↪ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByValue**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that selects list items identified by value and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **value** (*TypeValue*) – Value or value expression of element to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the index of the element with value, 3, in list bin "a".
expr = exp.ListGetByValue(None, aerospike.LIST_RETURN_INDEX, 3, exp.ListBin("a
↪")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByValueList**(*ctx: TypeCTX*, *return_type: int*, *value: TypeListValue*, *bin: TypeBinName*)

Create an expression that selects list items identified by values and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value: TypeListValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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.

- **value** (*TypeListValue*) – List or list expression of values of elements to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the indexes of the the elements in list bin "a" with values [3, 6, 12].
expr = exp.ListGetByValueList(None, aerospike.LIST_RETURN_INDEX, [3, 6, 12],␣
↪exp.ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByValueRange**(*ctx: TypeCTX*, *return_type: int*, *value_begin: TypeValue*, *value_end: TypeValue*, *bin: TypeBinName*)

Create an expression that selects list items identified by value range and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value_begin: TypeValue*, *value_end: TypeValue*, *bin: TypeBinName*)

Create an expression that selects list items identified by value range and returns selected data specified by return_type.

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **value_begin** (*TypeValue*) – Value or value expression of first element to get.
- **value_end** (*TypeValue*) – Value or value expression of ending element.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get rank of values between 3 (inclusive) and 7 (exclusive) in list bin "a".
expr = exp.ListGetByValueRange(None, aerospike.LIST_RETURN_RANK, 3, 7, exp.
↪ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListGetByValueRelRankRange**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

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: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.

- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **value** (*TypeValue*) – Value or value expression to get items relative to.
- **rank** (*TypeRank*) – Rank integer expression. rank relative to "value" to start getting elements.
- **count** (*TypeCount*) – Integer value or integer value expression, how many elements to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# [6, 12, 4, 21]
expr = exp.ListGetByValueRelRankRange(None, aerospike.LIST_RETURN_VALUE, 3, 1,␣
↪2,
    exp.ListBin("a")).compile()
# The next greater value after 3 in the list is 4
# Rank 0 = 4
# So we only fetch two values starting at rank 1
# [12, 6]
```

class aerospike_helpers.expressions.list.**ListGetByValueRelRankRangeToEnd**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that selects list items nearest to value and greater by relative rank

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *List Return Types* values.
- **value** (*TypeValue*) – Value or value expression to get items relative to.
- **rank** (*TypeRank*) – Rank integer expression. rank relative to "value" to start getting elements.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# [6, 12, 4, 21]
expr = exp.ListGetByValueRelRankRangeToEnd(None, aerospike.LIST_RETURN_VALUE, 3,
↪ 1,
    exp.ListBin("a")).compile()
# Rank 0: 4
```

(continues on next page)

(continued from previous page)

```
# We only fetch values with rank 1 or more (i.e the rest of the list)
# Expected results: [6, 12, 21]
```

class aerospike_helpers.expressions.list.**ListIncrement**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that increments list[index] by value.

**__init__**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *List policies*.
- **index** (*TypeIndex*) – Index of value to increment.
- **value** (*TypeValue*) – Value or value expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
List expression.

Example:

```
# Check if incremented value in list bin "a" is the largest in the list.
# Rank of -1 == largest element
largestListValue = exp.ListGetByRank(None, aerospike.LIST_RETURN_VALUE,␣
↪ResultType.INTEGER, -1)
listIncrementedAtIndex1 = exp.ListIncrement(None, None, 1, 5, exp.ListBin("a"))
listItemAtIndex1 = exp.ListGetByIndex(None, aerospike.LIST_RETURN_VALUE,␣
↪ResultType.INTEGER, 1,
    listIncrementedAtIndex1)
expr = exp.Eq(
    largestListValue,
    listItemAtIndex1
).compile()
```

class aerospike_helpers.expressions.list.**ListInsert**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that inserts value to specified index of list.

**__init__**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *List policies*.
- **index** (*TypeIndex*) – Target index for insertion, integer or integer expression.
- **value** (*TypeValue*) – Value or value expression to be inserted.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
List expression.

Example:

```
# Check if list bin "a" has length > 5 after insert.
listInsertedBy3At0 = exp.ListInsert(None, None, 0, 3, exp.ListBin("a"))
expr = exp.GT(exp.ListSize(None, listInsertedBy3At0), 5).compile()
```

**class** aerospike_helpers.expressions.list.**ListInsertItems**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *values: TypeListValue*, *bin: TypeBinName*)

Create an expression that inserts each input list item starting at specified index of list.

**__init__**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *values: TypeListValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *List policies*.
- **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 expression, such as *MapBin* or *ListBin*.

**Returns**
List expression.

Example:

```
# Check if list bin "a" has length > 5 after inserting items.
listInsertedByTwoItems = exp.ListInsertItems(None, None, 0, [4, 7], exp.ListBin(
↪"a"))
expr = exp.GT(exp.ListSize(None, listInsertedByTwoItems), 5).compile()
```

**class** aerospike_helpers.expressions.list.**ListRemoveByIndex**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that removes "count" list items starting at specified index.

**__init__**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **index** (*TypeIndex*) – Index integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Get size of list bin "a" after index 3 has been removed.
expr = exp.ListSize(None, exp.ListRemoveByIndex(None, 3, exp.ListBin("a"))).
↪compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByIndexRange**(*ctx: TypeCTX*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that removes "count" list items starting at specified index.

**__init__**(*ctx: TypeCTX*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Get size of list bin "a" after index 3, 4, and 5 have been removed.
expr = exp.ListSize(None, exp.ListRemoveByIndexRange(None, 3, 3, exp.ListBin("a
↪"))).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByIndexRangeToEnd**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that removes list items starting at specified index to the end of list.

**__init__**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **index** (*TypeIndex*) – Starting index integer or integer expression of elements to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove all elements starting from index 3 in list bin "a".
expr = exp.ListRemoveByIndexRangeToEnd(None, 3, exp.ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByRank**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that removes list item identified by rank.

**__init__**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove smallest value in list bin "a".
expr = exp.ListRemoveByRank(None, 0, exp.ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListRemoveByRankRange**(*ctx: TypeCTX*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that removes "count" list items starting at specified rank.

**__init__**(*ctx: TypeCTX*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove the 3 smallest items from list bin "a".
expr = exp.ListRemoveByRankRange(None, 0, 3, exp.ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListRemoveByRankRangeToEnd**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that removes list items starting at specified rank to the last ranked item.

**__init__**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to start removing at.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove the 2 largest elements from List bin "a".
# Assume list bin contains [6, 12, 4, 21]
expr = exp.ListRemoveByRankRangeToEnd(None, -2, exp.ListBin("a")).compile()
# Expected results: [6, 4]
```

**class** aerospike_helpers.expressions.list.**ListRemoveByValue**(*ctx: TypeCTX*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that removes list items identified by value.

**__init__**(*ctx: TypeCTX*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **value** (*TypeValue*) – Value or value expression to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# See if list bin "a", with `3` removed, is equal to list bin "b".
listRemoved3 = exp.ListRemoveByValue(None, 3, exp.ListBin("a"))
expr = exp.Eq(listRemoved3, exp.ListBin("b")).compile()
```

**class** aerospike_helpers.expressions.list.**ListRemoveByValueList**(*ctx: TypeCTX*, *values: TypeListValue*, *bin: TypeBinName*)

Create an expression that removes list items identified by values.

**__init__**(*ctx: TypeCTX*, *values: TypeListValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **values** (*TypeListValue*) – List of values or list expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove elements with values [1, 2, 3] from list bin "a".
expr = exp.ListRemoveByValueList(None, [1, 2, 3], exp.ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListRemoveByValueRange**(*ctx: TypeCTX*, *begin: TypeValue*, *end: TypeValue*, *bin: TypeBinName*)

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: TypeCTX*, *begin: TypeValue*, *end: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **begin** (*TypeValue*) – Begin value or value expression for range.
- **end** (*TypeValue*) – End value or value expression for range.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove list of items with values >= 3 and < 7 from list bin "a".
expr = exp.ListRemoveByValueRange(None, 3, 7, exp.ListBin("a")).compile()
```

class aerospike_helpers.expressions.list.**ListRemoveByValueRelRankRange**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that removes list items nearest to value and greater by relative rank with a count limit.

**__init__**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **value** (*TypeValue*) – Start value or value expression.
- **rank** (*TypeRank*) – Rank integer or integer expression.
- **count** (*TypeCount*) – How many elements to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove 2 elements greater than 4
# Assume list in bin a is: [6, 12, 4, 21]
exp.ListRemoveByValueRelRankRange(None, 4, 1, 2, exp.ListBin("a"))
# Expected results: [4, 21]
```

class aerospike_helpers.expressions.list.**ListRemoveByValueRelRankToEnd**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that removes list items nearest to value and greater by relative rank.

**__init__**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **value** (*TypeValue*) – Start value or value expression.
- **rank** (*TypeRank*) – Rank integer or integer expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Remove elements larger than 4 by relative rank in list bin "a".
# Assume list in bin a is: [6, 12, 4, 21]
expr = exp.ListRemoveByValueRelRankToEnd(None, 4, 1, exp.ListBin("a")).compile()
```

(continues on next page)

(continued from previous page)

```
# Expected results: [4]
# All elements starting with and after the rank are removed
```

**class** aerospike_helpers.expressions.list.**ListSet**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that sets item value at specified index in list.

**__init__**(*ctx: TypeCTX*, *policy: TypePolicy*, *index: TypeIndex*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *List policies*.
- **index** (*TypeIndex*) – index of value to set.
- **value** (*TypeValue*) – value or value expression to set index in list to.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
List expression.

Example:

```
# Get smallest element in list bin "a" after setting index 1 to 10.
listSetAtIndex1 = exp.ListSet(None, None, 1, 10, exp.ListBin("a"))
expr = exp.ListGetByRank(None, aerospike.LIST_RETURN_VALUE, ResultType.INTEGER,␣
↪0,
    listSetAtIndex1).compile()
```

**class** aerospike_helpers.expressions.list.**ListSize**(*ctx: TypeCTX*, *bin: TypeBinName*)

Create an expression that returns list size.

**__init__**(*ctx: TypeCTX*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Integer expression.

Example:

```
#Take the size of list bin "a".
expr = exp.ListSize(None, exp.ListBin("a")).compile()
```

**class** aerospike_helpers.expressions.list.**ListSort**(*ctx: TypeCTX*, *order: int*, *bin: TypeBinName*)

Create an expression that sorts a list.

**__init__**(*ctx: TypeCTX*, *order: int*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.

- **order** (*int*) – Optional flags modifying the behavior of list_sort. This should be constructed by bitwise or'ing together values from *List Sort Flags*.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
list expression.

Example:

```
# Get value of sorted list bin "a".
expr = exp.ListSort(None, aerospike.LIST_SORT_DEFAULT, "a").compile()
```

### aerospike_helpers.expressions.map module

Map expressions contain expressions for reading and modifying Maps. Most of these operations are from the standard *Map API*.

**class** aerospike_helpers.expressions.map.**MapClear**(*ctx: TypeCTX*, *bin: TypeBinName*)

Create an expression that removes all items in map.

**__init__**(*ctx: TypeCTX*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Clear map bin "b".
expr = exp.MapClear(None, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByIndex**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that selects map item identified by index and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map 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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the value at index 0 in map bin "b". (assume this value is an integer)
expr = exp.MapGetByIndex(None, aerospike.MAP_RETURN_VALUE, ResultType.INTEGER,␣
↪0, MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByIndexRange**(*ctx: TypeCTX*, *return_type: int*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that selects "count" map items starting at specified index and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get elements at indexes 3, 4, 5, 6 in map bin "b".
expr = exp.MapGetByIndexRange(None, aerospike.MAP_RETURN_VALUE, 3, 4, MapBin("b
↪")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByIndexRangeToEnd**(*ctx: TypeCTX*, *return_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that selects map items starting at specified index to the end of map and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **index** (*TypeIndex*) – Integer or integer expression of index to start getting elements at.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get element at index 5 to end from map bin "b".
expr = exp.MapGetByIndexRangeToEnd(None, aerospike.MAP_RETURN_VALUE, 5, MapBin(
↪"b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByKey**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *key: TypeKey*, *bin: TypeBinName*)

Create an expression that selects map item identified by key and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *key: TypeKey*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the value at key "key0" in map bin "b". (assume the value at key0 is an␣
↪integer)
expr = exp.MapGetByKey(None, aerospike.MAP_RETURN_VALUE, ResultType.INTEGER,
↪"key0",
    exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByKeyList**(*ctx: TypeCTX*, *return_type: int*, *keys: TypeKeyList*, *bin: TypeBinName*)

Create an expression that selects map items identified by keys and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *keys: TypeKeyList*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **keys** (*TypeKeyList*) – List of key values or list expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get elements at keys "key3", "key4", "key5" in map bin "b".
expr = exp.MapGetByKeyList(None, aerospike.MAP_RETURN_VALUE, ["key3", "key4",
↪"key5"],
    exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByKeyRange**(*ctx: TypeCTX*, *return_type: int*, *begin: TypeKey*, *end: TypeKey*, *bin: TypeBinName*)

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.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *begin: TypeKey*, *end: TypeKey*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **begin** (*TypeKey*) – Key value or expression.
- **end** (*TypeKey*) – Key value or expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get elements at keys "key3", "key4", "key5", "key6" in map bin "b".
expr = exp.MapGetByKeyRange(None, aerospike.MAP_RETURN_VALUE, "key3", "key7",␣
↪exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByKeyRelIndexRange**(*ctx: TypeCTX*, *return_type: int*, *key: TypeKey*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

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.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *key: TypeKey*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
expr = exp.MapGetByKeyRelIndexRange(None, aerospike.MAP_RETURN_VALUE, "key2", 0,
↪ 2,
    exp.MapBin("b")).compile()
```

(continues on next page)

(continued from previous page)

```
# [2, 3]
expr = exp.MapGetByKeyRelIndexRange(None, aerospike.MAP_RETURN_VALUE, "key2", 1,
↪ 2,
    exp.MapBin("b")).compile()
# [3, 10]
```

class aerospike_helpers.expressions.map.**MapGetByKeyRelIndexRangeToEnd**(*ctx: TypeCTX*, *return_type: int*, *key: TypeKey*, *index: TypeIndex*, *bin: TypeBinName*)

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.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *key: TypeKey*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **key** (*TypeKey*) – Key value or value expression.
- **index** (*TypeIndex*) – Index integer or integer value expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**

Expression.

Example:

```
# Get elements with keys larger than "key2" from map bin "b".
expr = exp.MapGetByKeyRelIndexRangeToEnd(None, aerospike.MAP_RETURN_VALUE, "key2
↪", 1,
    exp.MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByRank**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that selects map items identified by rank and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the smallest element in map bin "b".
expr = exp.MapGetByRank(None, aerospike.MAP_RETURN_VALUE, aerospike.ResultType.
↪INTEGER, 0,
    MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByRankRange**(*ctx: TypeCTX*, *return_type: int*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that selects "count" map items starting at specified rank and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the 3 smallest elements in map bin "b".
expr = exp.MapGetByRankRange(None, aerospike.MAP_RETURN_VALUE, 0, 3, exp.MapBin(
↪"b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByRankRangeToEnd**(*ctx: TypeCTX*, *return_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that selects map items starting at specified rank to the last ranked item and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **rank** (*TypeRank*) – Rank integer or integer expression of first element to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the three largest elements in map bin "b".
expr = exp.MapGetByRankRangeToEnd(None, aerospike.MAP_RETURN_VALUE, -3, MapBin(
↪"b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByValue**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that selects map items identified by value and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **value** (*TypeValue*) – Value or value expression of element to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the rank of the element with value, 3, in map bin "b".
expr = exp.MapGetByValue(None, aerospike.MAP_RETURN_RANK, 3, exp.MapBin("b")).
↪compile()
```

**class** aerospike_helpers.expressions.map.**MapGetByValueList**(*ctx: TypeCTX*, *return_type: int*, *value: TypeListValue*, *bin: TypeBinName*)

Create an expression that selects map items identified by values and returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value: TypeListValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **value** (*TypeListValue*) – List or list expression of values of elements to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the indexes of the the elements in map bin "b" with values [3, 6, 12].
expr = exp.MapGetByValueList(None, aerospike.MAP_RETURN_INDEX, [3, 6, 12], exp.
↪MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByValueRange**(*ctx: TypeCTX*, *return_type: int*, *value_begin: TypeValue*, *value_end: TypeValue*, *bin: TypeBinName*)

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.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value_begin: TypeValue*, *value_end: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **value_begin** (*TypeValue*) – Value or value expression of first element to get.
- **value_end** (*TypeValue*) – Value or value expression of ending element.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get elements with values between 3 and 7 from map bin "b".
expr = exp.MapGetByValueRange(None, aerospike.MAP_RETURN_VALUE, 3, 7, exp.
↪MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapGetByValueRelRankRange**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

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: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **value** (*TypeValue*) – Value or value expression to get items relative to.
- **rank** (*TypeRank*) – Rank integer expression. rank relative to "value" to start getting elements.
- **count** (*TypeCount*) – Integer value or integer value expression, how many elements to get.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# {"key1": 1, "key2": 2, "key3": 3, "key4": 10}
# Get next two largest values greater than a value of 1
expr = exp.MapGetByValueRelRankRange(None, aerospike.MAP_RETURN_VALUE, 1, 1, 2,␣
↪exp.MapBin("b")).compile()
# [2, 3]
```

**class** aerospike_helpers.expressions.map.**MapGetByValueRelRankRangeToEnd**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that selects map items nearest to value and greater by relative rank, Expression returns selected data specified by return_type.

**__init__**(*ctx: TypeCTX*, *return_type: int*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **return_type** (*int*) – Value specifying what should be returned from the operation. This should be one of the *Map Return Types* values.
- **value** (*TypeValue*) – Value or value expression to get items relative to.
- **rank** (*TypeRank*) – Rank integer expression. rank relative to "value" to start getting elements.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Expression.

Example:

```
# Get the values of all elements in map bin "b" larger than 3.
expr = exp.MapGetByValueRelRankRangeToEnd(None, aerospike.MAP_RETURN_VALUE, 3,␣
↪1, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapIncrement**(*ctx: TypeCTX*, *policy: TypePolicy*, *key: TypeKey*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that increments a map value, by value, for all items identified by key. Valid only for numbers.

**__init__**(*ctx: TypeCTX*, *policy: TypePolicy*, *key: TypeKey*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *Map policies*.
- **key** (*TypeKey*) – Key value or value expression element to increment.
- **value** (*TypeValue*) – Increment element by value expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Increment element at 'vageta' in map bin "b" by 9000.
expr = exp.MapIncrement(None, None, 'vageta', 9000, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapPut**(*ctx: TypeCTX*, *policy: TypePolicy*, *key: TypeKey*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that writes key/val to map bin.

**__init__**(*ctx: TypeCTX*, *policy: TypePolicy*, *key: TypeKey*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *Map policies*.
- **key** (*TypeKey*) – Key value or value expression to put into map.
- **value** (*TypeValue*) – Value or value expression to put into map.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Put {"key": 27} into map bin "b".
expr = exp.MapPut(None, None, "key", 27, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapPutItems**(*ctx: TypeCTX*, *policy: TypePolicy*, *map: map*, *bin: TypeBinName*)

Create an expression that writes each map item to map bin.

**__init__**(*ctx: TypeCTX*, *policy: TypePolicy*, *map: map*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **policy** (*TypePolicy*) – Optional dictionary of *Map policies*.
- **map** (*map*) – Map or map expression of items to put into target map.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Put {27: 'key27', 28: 'key28'} into map bin "b".
expr = exp.MapPutItems(None, None, {27: 'key27', 28: 'key28'}, exp.MapBin("b")).
↪compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByIndex**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that removes map item identified by index.

**__init__**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **index** (*TypeIndex*) – Index integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove element with smallest key from map bin "b".
expr = exp.MapRemoveByIndex(None, 0, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByIndexRange**(*ctx: TypeCTX*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that removes count map items starting at specified index.

**__init__**(*ctx: TypeCTX*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Get size of map bin "b" after index 3, 4, and 5 have been removed.
expr = exp.MapSize(None, exp.MapRemoveByIndexRange(None, 3, 3, exp.MapBin("b
↪"))).compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByIndexRangeToEnd**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that removes map items starting at specified index to the end of map.

**__init__**(*ctx: TypeCTX*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **index** (*TypeIndex*) – Starting index integer or integer expression of elements to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove all elements starting from index 3 in map bin "b".
expr = exp.MapRemoveByIndexRangeToEnd(None, 3, exp.MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKey**(*ctx: TypeCTX*, *key: TypeKey*, *bin: TypeBinName*)

Create an expression that removes a map item identified by key.

**__init__**(*ctx: TypeCTX*, *key: TypeKey*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **key** (*TypeKey*) – Key value or value expression of key to element to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove element at key 1 in map bin "b".
expr = exp.MapRemoveByKey(None, 1, exp.MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKeyList**(*ctx: TypeCTX*, *keys: List[Union[_BaseExpr, Any]]*, *bin: TypeBinName*)

Create an expression that removes map items identified by keys.

**__init__**(*ctx: TypeCTX*, *keys: List[Union[_BaseExpr, Any]]*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **key** (*List[TypeKey]*) – List of key values or a list expression of keys to elements to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove elements at keys [1, 2] in map bin "b".
expr = exp.MapRemoveByKeyList(None, [1, 2], exp.MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByKeyRange**(*ctx: TypeCTX*, *begin: TypeValue*, *end: TypeValue*, *bin: TypeBinName*)

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: TypeCTX*, *begin: TypeValue*, *end: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **begin** (*TypeValue*) – Begin value expression.

- **end** (*TypeValue*) – End value expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove elements at keys between 1 and 10 in map bin "b".
expr = exp.MapRemoveByKeyRange(None, 1, 10, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByKeyRelIndexRange**(*ctx: TypeCTX*, *key: TypeKey*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that removes map items nearest to key and greater by index with a count limit.

**__init__**(*ctx: TypeCTX*, *key: TypeKey*, *index: TypeIndex*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove the next two items after key1
# {"key1": 1, "key2": 2, "key3": 3, "key4": 10}
expr = exp.MapRemoveByKeyRelIndexRange(None, "key1", 1, 2, exp.MapBin("b")).
↪compile()
# {"key1": 1, "key4": 10}
```

**class** aerospike_helpers.expressions.map.**MapRemoveByKeyRelIndexRangeToEnd**(*ctx: TypeCTX*, *key: TypeKey*, *index: TypeIndex*, *bin: TypeBinName*)

Create an expression that removes map items nearest to key and greater by index.

**__init__**(*ctx: TypeCTX*, *key: TypeKey*, *index: TypeIndex*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **key** (*TypeKey*) – Key value or expression for key to start removing from.
- **index** (*TypeIndex*) – Index integer or integer expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# {"key1": 1, "key2": 2, "key3": 3, "key4": 10}
expr = exp.MapGetByKeyRelIndexRangeToEnd(None, aerospike.MAP_RETURN_VALUE, "key2
↪", 1,              exp.MapBin("b")).compile()
# [3, 10]
```

**class** aerospike_helpers.expressions.map.**MapRemoveByRank**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that removes map item identified by its value's rank.

**__init__**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove key with smallest value in map bin "b".
expr = exp.MapRemoveByRank(None, 0, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByRankRange**(*ctx: TypeCTX*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that removes "count" map items starting at specified rank.

**__init__**(*ctx: TypeCTX*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove 3 keys with the smallest values from map bin "b".
expr = exp.MapRemoveByRankRange(None, 0, 3, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByRankRangeToEnd**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that removes map items starting at specified rank to the last ranked item.

**__init__**(*ctx: TypeCTX*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **rank** (*TypeRank*) – Rank integer or integer expression of element to start removing at.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove keys with 2 largest values from map bin "b".
expr = exp.MapRemoveByRankRangeToEnd(None, -2, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByValue**(*ctx: TypeCTX*, *value: TypeValue*, *bin: TypeBinName*)

Create an expression that removes map items identified by value.

**__init__**(*ctx: TypeCTX*, *value: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **value** (*TypeValue*) – Value or value expression to remove.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove {"key1": 1} from map bin "b".
expr = exp.MapRemoveByValue(None, 1, exp.MapBin("b")).compile()
```

**class** aerospike_helpers.expressions.map.**MapRemoveByValueList**(*ctx: TypeCTX*, *values: TypeListValue*, *bin: TypeBinName*)

Create an expression that removes map items identified by values.

**__init__**(*ctx: TypeCTX*, *values: TypeListValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **values** (*TypeListValue*) – List of values or list expression.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove elements with values 1, 2, 3 from map bin "b".
expr = exp.MapRemoveByValueList(None, [1, 2, 3], exp.MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValueRange**(*ctx: TypeCTX*, *begin: TypeValue*, *end: TypeValue*, *bin: TypeBinName*)

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: TypeCTX*, *begin: TypeValue*, *end: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **begin** (*TypeValue*) – Begin value or value expression for range.
- **end** (*TypeValue*) – End value or value expression for range.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove list of items with values >= 3 and < 7 from map bin "b".
expr = exp.MapRemoveByValueRange(None, 3, 7, exp.MapBin("b")).compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValueRelRankRange**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

Create an expression that removes map items nearest to value and greater by relative rank with a count limit.

**__init__**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *count: TypeCount*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **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 expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove the key with a value just lower than 17
expr = exp.MapRemoveByValueRelRankRange(None, 17, -1, 1, exp.MapBin("b")).
↪compile()
```

class aerospike_helpers.expressions.map.**MapRemoveByValueRelRankRangeToEnd**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

Create an expression that removes map items nearest to value and greater by relative rank.

**__init__**(*ctx: TypeCTX*, *value: TypeValue*, *rank: TypeRank*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **value** (*TypeValue*) – Value or value expression to start removing from.
- **rank** (*TypeRank*) – Integer or integer expression of rank.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Map expression.

Example:

```
# Remove all elements with values larger than 3 from map bin "b".
expr = exp.MapRemoveByValueRelRankRangeToEnd(None, 3, 1, exp.MapBin("b")).
↪compile()
```

**class** aerospike_helpers.expressions.map.**MapSize**(*ctx: TypeCTX*, *bin: TypeBinName*)

Create an expression that returns map size.

**__init__**(*ctx: TypeCTX*, *bin: TypeBinName*)

**Parameters**

- **ctx** (*TypeCTX*) – An optional list of nested CDT *cdt_ctx* context operation objects.
- **bin** (*TypeBinName*) – bin expression, such as *MapBin* or *ListBin*.

**Returns**
Integer expression.

Example:

```
#Take the size of map bin "b".
expr = exp.MapSize(None, exp.MapBin("b")).compile()
```

### aerospike_helpers.expressions.bit module

Bitwise expressions contain expressions for performing bitwise operations. Most of these operations are equivalent to the *Bitwise Operations API* for binary data.

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: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: int*, *action: int*, *bin: TypeBinName*)

Create an expression that performs a bit_add operation. Note: integers are stored big-endian.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: int*, *action: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* expression.

**Returns**
resulting blob with the bits operated on.

Example:

```
# Assume we have a blob bin of five bytes: bytearray([1, 1, 1, 1, 1])
expr = exp.BitAdd(None, 8, 8, 1, aerospike.BIT_OVERFLOW_FAIL, exp.BlobBin("b")).
↪compile()
# Treat the selected bits as a number and add the value to it
#   00000001 00000001 00000001 00000001 00000001
#   01234567 89012345 (offset)
#            ^(+8)
#            xxxxxxxx (bits selected)
# + 00000000 00000001 (value to add)
#   ========================================
#   00000001 00000010 00000001 00000001 00000001
```

**class** aerospike_helpers.expressions.bitwise.**BitAnd**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

Create an expression that performs a bit_and operation.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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, 1, 1, 1, 1])
expr = exp.BitAnd(None, 0, 8, bytearray([0]), exp.BlobBin("c")).compile()
#   00000001 00000001 00000001 00000001 00000001
#   01234567 (offset)
#   ^(+8)
#   xxxxxxxx (bits selected)
```

(continues on next page)

(continued from previous page)

```
#   00000000 (& operation)
#   ========================================
#   00000000 00000001 00000001 00000001 00000001
```

**class** aerospike_helpers.expressions.bitwise.**BitCount**(*bit_offset: int*, *bit_size: int*, *bin: TypeBinName*)

Create an expression that performs a bit_count operation.

**__init__**(*bit_offset: int*, *bit_size: int*, *bin: TypeBinName*)

**Parameters**

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to count.
- **bin** (*TypeBinName*) – A *BlobBin* 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 = exp.BitCount(16, 8 * 3, exp.BlobBin("c")).compile()
# 00000011 00000011 00000011 00000011 00000011
# 01234567 89012345 6 (offset = 16)
#                   12345678 90123456 78901234 (bit count = 24)
# Number of 1's = 6
```

**class** aerospike_helpers.expressions.bitwise.**BitGet**(*bit_offset: int*, *bit_size: int*, *bin: TypeBinName*)

Create an expression that performs a bit_get operation.

**__init__**(*bit_offset: int*, *bit_size: int*, *bin: TypeBinName*)

**Parameters**

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to get.
- **bin** (*TypeBinName*) – A *BlobBin* 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 = exp.BitGet(8, 8, exp.BlobBin("c")).compile()
```

**class** aerospike_helpers.expressions.bitwise.**BitGetInt**(*bit_offset: int*, *bit_size: int*, *sign: bool*, *bin: TypeBinName*)

Create an expression that performs a bit_get_int operation.

**__init__**(*bit_offset: int*, *bit_size: int*, *sign: bool*, *bin: TypeBinName*)

**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*) – A *BlobBin* expression.

**Returns**
Integer expression.

Example:

```
# Let blob bin "c" == bytearray([1, 2, 3, 4, 5).
# Get 2 as an integer from bin "c".
expr = exp.BitGetInt(8, 8, True, exp.BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitInsert**(*policy: TypePolicy*, *byte_offset: int*, *value: TypeBitValue*, *bin: TypeBinName*)

Create an expression that performs a bit_insert operation.

**__init__**(*policy: TypePolicy*, *byte_offset: int*, *value: TypeBitValue*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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 = exp.BitInsert(None, 1, bytearray([3]), exp.BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitLeftScan**(*bit_offset: int*, *bit_size: int*, *value: bool*, *bin: TypeBinName*)

Create an expression that performs a bit_lscan operation.

**__init__**(*bit_offset: int*, *bit_size: int*, *value: bool*, *bin: TypeBinName*)

**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*) – A *BlobBin* 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 = exp.BitLeftScan(0, 8, True, exp.BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitLeftShift**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *shift: int*, *bin: TypeBinName*)

Create an expression that performs a bit_lshift operation.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *shift: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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 = exp.BitLeftShift(None, 0, 8, 3, exp.BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitNot**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *bin: TypeBinName*)

Create an expression that performs a bit_not operation.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **bit_offset** (*int*) – Bit index of where to start operation.
- **bit_size** (*int*) – Number of bits to be operated on.
- **bin** (*TypeBinName*) – A *BlobBin* 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 = exp.BitNot(None, 0, 40, exp.BlobBin("c")).compile()
```

class aerospike_helpers.expressions.bitwise.**BitOr**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

Create an expression that performs a bit_or operation.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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 = exp.BitOr(None, 0, 8, bytearray([8]), exp.BlobBin("c")).compile()
#   00000001 00000001 00000001 00000001 00000001
#   0 (index)
#   xxxxxxxx (bits applied)
#   00001000 (OR operation)
#   ========================================
#   00001001 00000001 00000001 00000001 00000001
```

**class** aerospike_helpers.expressions.bitwise.**BitRemove**(*policy: TypePolicy*, *byte_offset: int*, *byte_size: int*, *bin: TypeBinName*)

Create an expression that performs a bit_remove operation.

**__init__**(*policy: TypePolicy*, *byte_offset: int*, *byte_size: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **byte_offset** (*int*) – Byte index of where to start removing from.
- **byte_size** (*int*) – Number of bytes to remove.
- **bin** (*TypeBinName*) – A *BlobBin* expression.

**Returns**

Resulting blob containing the remaining bytes.

Example:

```
# b = bytearray([1, 2, 3, 4, 5])
expr = exp.BitRemove(None, 1, 1, exp.BlobBin("b")).compile()
#   00000001 00000010 00000011 00000100 00000101
#   0        1 (selected byte)
#            xxxxxxxx (byte(s) to remove)
#   ========================================
#   00001001 <-- 00000001 00000001 00000001
```

**class** aerospike_helpers.expressions.bitwise.**BitResize**(*policy: TypePolicy*, *byte_size: int*, *flags: int*, *bin: TypeBinName*)

Create an expression that performs a bit_resize operation.

**__init__**(*policy: TypePolicy*, *byte_size: int*, *flags: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **byte_size** (*int*) – Number of bytes the resulting blob should occupy.
- **flags** (*int*) – One or a combination of bit resize flags.
- **bin** (*TypeBinName*) – A *BlobBin* 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 = exp.BitResize(None, 10, aerospike.BIT_RESIZE_FROM_FRONT, exp.BlobBin("c
↪")).compile()
```

**class** aerospike_helpers.expressions.bitwise.**BitRightScan**(*bit_offset: int*, *bit_size: int*, *value: bool*, *bin: TypeBinName*)

Create an expression that performs a bit_rscan operation.

**__init__**(*bit_offset: int*, *bit_size: int*, *value: bool*, *bin: TypeBinName*)

**Parameters**

- **bit_offset** (*int*) – Bit index of where to start reading.
- **bit_size** (*int*) – Number of bits to read.
- **value** (*bool*) – Bit value to check for.
- **bin** (*TypeBinName*) – A *BlobBin* expression.

**Returns**

Index of the right most bit starting from bit_offset set to value. Returns -1 if not found.

Example:

```
# b = bytearray([1, 0, 0, 0, 128])
expr = exp.BitRightScan(32, 8, True, exp.BlobBin("b")).compile()
# 00000001 00000000 00000000 00000000 10000000
# 01234567 89012345 67890123 45678901 2 (offset=32)
#                                     xxxxxxxx (selected bits)
#                                     01234567 (local offset)
# 1 found at local offset 0
```

**class** aerospike_helpers.expressions.bitwise.**BitRightShift**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *shift: int*, *bin: TypeBinName*)

Create an expression that performs a bit_rshift operation.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *shift: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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 = exp.BitRightShift(None, 0, 8, 1, exp.BlobBin("c")).compile()
# 00001000 00001000 00001000 00001000 00001000
# 0 (offset=0)
# xxxxxxxx (selected bits -> by 1 only in that region)
# The rest of the bits are unaffected
# 00000100 00001000 00001000 00001000 00001000
```

**class** aerospike_helpers.expressions.bitwise.**BitSet**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

Create an expression that performs a bit_set operation.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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 = exp.BitSet(None, 7, 1, bytearray([255]), exp.BlobBin("c")).compile()
```

**class** aerospike_helpers.expressions.bitwise.**BitSetInt**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: int*, *bin: TypeBinName*)

Create an expression that performs a bit_set_int operation. Note: integers are stored big-endian.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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 = exp.BitSetInt(None, 7, 1, 1, exp.BlobBin("c")).compile()
```

**class** aerospike_helpers.expressions.bitwise.**BitSubtract**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: int*, *action: int*, *bin: TypeBinName*)

Create an expression that performs a bit_subtract operation. Note: integers are stored big-endian.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: int*, *action: int*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.
- **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*) – A *BlobBin* 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 = exp.BitSubtract(None, 8, 8, 1, aerospike.BIT_OVERFLOW_FAIL).compile()
```

**class** aerospike_helpers.expressions.bitwise.**BitXor**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

Create an expression that performs a bit_xor operation.

**__init__**(*policy: TypePolicy*, *bit_offset: int*, *bit_size: int*, *value: TypeBitValue*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – Optional dictionary of *Bit policies*.

- **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*) – A *BlobBin* 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 = exp.BitXor(None, 0, 8, bytearray([1]), exp.BlobBin("c")).compile()
```

### aerospike_helpers.expressions.hll module

HyperLogLog expressions contain expressions for performing HLL operations. Most of these operations are equivalent to the *HyperLogLog API*.

**class** aerospike_helpers.expressions.hll.**HLLAdd**(*policy: TypePolicy*, *list: TypeListValue*, *index_bit_count: Optional[int]*, *mh_bit_count: Optional[int]*, *bin: TypeBinName*)

Create an expression that performs an hll_add.

**__init__**(*policy: TypePolicy*, *list: TypeListValue*, *index_bit_count: Optional[int]*, *mh_bit_count: Optional[int]*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – An optional dictionary of *HyperLogLog policies*.
- **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*) – An *HLLBin* expression.

**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 = exp.HLLAdd(None, ['key4', 'key5', 'key6'], 8, 8, exp.HLLBin("d")).
↪compile()
```

**class** aerospike_helpers.expressions.hll.**HLLDescribe**(*bin: TypeBinName*)

Create an expression that performs an as_operations_hll_describe.

**__init__**(*bin: TypeBinName*)

**Parameters**
**bin** (*TypeBinName*) – An *HLLBin* expression.

**Returns**
List bin, a list containing the index_bit_count and minhash_bit_count.

Example:

```
# Get description of HLL bin "d".
expr = exp.HLLDescribe(exp.HLLBin("d")).compile()
```

**class** aerospike_helpers.expressions.hll.**HLLGetCount**(*bin: TypeBinName*)

Create an expression that performs an as_operations_hll_get_count.

**__init__**(*bin: TypeBinName*)

**Parameters**
**bin** (*TypeBinName*) – An *HLLBin* expression.

**Returns**
Integer bin, the estimated number of unique elements in an HLL.

Example:

```
# Get count from HLL bin "d".
expr = exp.HLLGetCount(exp.HLLBin("d")).compile()
```

**class** aerospike_helpers.expressions.hll.**HLLGetIntersectCount**(*values: TypeValue*, *bin: TypeBinName*)

Create an expression that performs an as_operations_hll_get_intersect_count.

**__init__**(*values: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to intersect with bin.
- **bin** (*TypeBinName*) – An *HLLBin* expression.

**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 = exp.HLLGetIntersectCount(values, exp.HLLBin("d")).compile()
```

**class** aerospike_helpers.expressions.hll.**HLLGetSimilarity**(*values: TypeValue*, *bin: TypeBinName*)

Create an expression that performs an as_operations_hll_get_similarity.

**__init__**(*values: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to calculate similarity with.
- **bin** (*TypeBinName*) – An *HLLBin* expression.

**Returns**
Float bin, estimated 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 = exp.HLLGetSimilarity(values, exp.HLLBin("d")).compile()
```

**class** aerospike_helpers.expressions.hll.**HLLGetUnion**(*values: TypeValue*, *bin: TypeBinName*)

Create an expression that performs an hll_get_union.

**__init__**(*values: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to union with bin.
- **bin** (*TypeBinName*) – An *HLLBin* expression.

**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 = exp.HLLGetUnion(values, exp.HLLBin("d")).compile()
```

**class** aerospike_helpers.expressions.hll.**HLLGetUnionCount**(*values: TypeValue*, *bin: TypeBinName*)

Create an expression that performs an as_operations_hll_get_union_count.

**__init__**(*values: TypeValue*, *bin: TypeBinName*)

**Parameters**

- **values** (*TypeValue*) – A single HLL or list of HLLs, values or expressions, to union with bin.
- **bin** (*TypeBinName*) – An *HLLBin* expression.

**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)].
```

(continues on next page)

(continued from previous page)

```
# Find the count of keys in the union of HLL bin "d" and all HLLs in values.␣
↪(Should be around 15000)
expr = exp.HLLGetUnionCount(values, exp.HLLBin("d")).compile()
```

**class** aerospike_helpers.expressions.hll.**HLLInit**(*policy: TypePolicy*, *index_bit_count: Optional[int]*, *mh_bit_count: Optional[int]*, *bin: TypeBinName*)

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.

**__init__**(*policy: TypePolicy*, *index_bit_count: Optional[int]*, *mh_bit_count: Optional[int]*, *bin: TypeBinName*)

**Parameters**

- **policy** (*TypePolicy*) – An optional dictionary of *HyperLogLog policies*.
- **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*) – An *HLLBin* expression.

**Returns**

Returns the resulting hll.

Example:

```
# Create an HLL with 12 index bits and 24 min hash bits.
expr = exp.HLLInit(None, 12, 24, exp.HLLBin("my_hll"))
```

**class** aerospike_helpers.expressions.hll.**HLLMayContain**(*list: TypeListValue*, *bin: TypeBinName*)

Create an expression that checks if the HLL bin contains any keys in list.

**__init__**(*list: TypeListValue*, *bin: TypeBinName*)

**Parameters**

- **list** (*TypeListValue*) – A list expression of keys to check if the HLL may contain them.
- **bin** (*TypeBinName*) – An *HLLBin* expression.

**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 = exp.HLLMayContain(["key1", "key2", "key3"], exp.HLLBin("d")).compile()
```

### aerospike_helpers.expressions.arithmetic module

Arithmetic expressions provide arithmetic operator support for Aerospike expressions.

**class** aerospike_helpers.expressions.arithmetic.**Abs**(*value: TypeNumber*)

Create operator that returns absolute value of a number. All arguments must resolve to integer or float.

Abs is also available via operator overloading using the built-in abs() function and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(*value: TypeNumber*)

**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 = exp.Eq(exp.Abs(exp.IntBin("a")), 1).compile()

# Using operator overloading
expr = exp.Eq(abs(exp.IntBin("a")), 1).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Add**(**args: TypeNumber*)

Create an add, (+) expression. All arguments must be the same type (integer or float).

Add is also available via operator overloading using + and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(**args: TypeNumber*)

**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 = exp.Eq(exp.Add(exp.IntBin("a"), exp.IntBin("b")), 11).compile()

# Using operator overloading.
expr = exp.Eq(exp.IntBin("a") + exp.IntBin("b"), 11).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Ceil**(*value: TypeFloat*)

Create ceil expression that rounds a floating point number up to the closest integer value.

Ceil is also available via operator overloading using the math.ceil() function and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(*value: TypeFloat*)

**Parameters**
**value** (*TypeFloat*) – Float expression or value to take ceiling of.

**Returns**
(float value)

Example:

```
# Ceil(2.25) == 3.0
expr = exp.Eq(exp.Ceil(2.25), 3.0).compile()

# Using operator overloading
import math
expr = exp.Eq(math.ceil(2.25), 3.0).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Div**(**args: TypeNumber*)

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).

Div is also available via operator overloading using / and any subclass of _BaseExpr. See the second example.

Floor div is also available via // but must be used with floats.

Requires server version 5.6.0+.

**__init__**(**args: TypeNumber*)

**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 = exp.GE(exp.Div(exp.IntBin("a"), exp.IntBin("b"), exp.IntBin("c")), 11).
↪compile()

# Using operator overloading.
expr = exp.GE(exp.IntBin("a") / exp.IntBin("b") / exp.IntBin("c"), 11).compile()

# Float bin "a" // "b" // "c" >= 11.0
expr = exp.GE(exp.FloatBin("a") // exp.FloatBin("b") // exp.FloatBin("c"), 11.
↪0).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Floor**(*value: TypeFloat*)

Create floor expression that rounds a floating point number down to the closest integer value.

Floor is also available via operator overloading using the math.floor() function and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(*value: TypeFloat*)

**Parameters**
**value** (*TypeFloat*) – Float expression or value to take floor of.

**Returns**
(float value)

Example:

```
# Floor(2.25) == 2.0
expr = exp.Eq(exp.Floor(2.25), 2.0).compile()

# Using operator overloading
import math
expr = exp.Eq(math.floor(2.25), 2.0).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Log**(*num: TypeFloat*, *base: TypeFloat*)

Create "log" operator for logarithm of "num" with base "base". All arguments must resolve to floats.

Requires server version 5.6.0+.

**__init__**(*num: TypeFloat*, *base: TypeFloat*)

**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 = exp.Eq(exp.Log(exp.FloatBin("a"), 2.0), 16.0).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Max**(**args: TypeNumber*)

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+.

**__init__**(**args: TypeNumber*)

**Parameters**

- ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values from which to find the
- **value.** (*maximum*) –

**Returns**
(integer or float value).

Example:

```
# for integer bins a, b, c, max(a, b, c) > 100
expr = exp.GT(exp.Max(exp.IntBin("a"), exp.IntBin("b"), exp.IntBin("c")), 100).
↪compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Min**(**args: TypeNumber*)

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+.

**__init__**(**args: TypeNumber*)

**Parameters**

- ***args** (*TypeNumber*) – Variable amount of float or integer expressions or values from which to find the
- **value.** (*minimum*) –

**Returns**
(integer or float value).

Example:

```
# for integer bins a, b, c, min(a, b, c) > 0
expr = exp.GT(exp.Min(exp.IntBin("a"), exp.IntBin("b"), exp.IntBin("c")), 0).
↪compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Mod**(*numerator: TypeInteger*, *denominator: TypeInteger*)

Create "modulo" (%) operator that determines the remainder of "numerator" divided by "denominator". All arguments must resolve to integers.

Mod is also available via operator overloading using % and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(*numerator: TypeInteger*, *denominator: TypeInteger*)

**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 = exp.Eq(exp.Mod(exp.IntBin("a"), 10), 0).compile()

# Using operator overloading.
expr = exp.Eq(exp.IntBin("a") % 10, 0).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Mul**(**args: TypeNumber*)

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).

Mul is also available via operator overloading using * and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(**args: TypeNumber*)

**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 = exp.GE(exp.Mul(exp.IntBin("a"), exp.IntBin("b")), 11).compile()

# Using operator overloading.
expr = exp.GE(exp.IntBin("a") * exp.IntBin("b"), 11).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Pow**(*base: TypeFloat*, *exponent: TypeFloat*)

Create "pow" operator that raises a "base" to the "exponent" power. All arguments must resolve to floats.

Pow is also available via operator overloading using ** and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(*base: TypeFloat*, *exponent: TypeFloat*)

**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 = exp.Eq(exp.Pow(exp.FloatBin("a"), 2.0), 16.0).compile()

# Using operator overloading.
expr = exp.Eq(exp.FloatBin("a") ** 2.0, 16.0).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**Sub**(**args: TypeNumber*)

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).

Sub is also available via operator overloading using - and any subclass of _BaseExpr. See the second example.

Requires server version 5.6.0+.

**__init__**(**args: TypeNumber*)

**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 = exp.Eq(exp.Sub(exp.IntBin("a"), exp.IntBin("b")), 11).compile()

# Using operator overloading.
expr = exp.Eq(exp.IntBin("a") - exp.IntBin("b"), 11).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**ToFloat**(*value: TypeInteger*)

Create expression that converts an integer to a float.

Requires server version 5.6.0+.

**__init__**(*value: TypeInteger*)

**Parameters**
**value** (*TypeInteger*) – Integer expression or value to convert to float.

**Returns**
(float value)

Example:

```
#For int bin "a", float(exp.IntBin("a")) == 2
expr = exp.Eq(exp.ToFloat(exp.IntBin("a")), 2).compile()
```

**class** aerospike_helpers.expressions.arithmetic.**ToInt**(*value: TypeFloat*)

Create expression that converts a float to an integer.

Requires server version 5.6.0+.

**__init__**(*value: TypeFloat*)

**Parameters**
**value** (*TypeFloat*) – Float expression or value to convert to int.

**Returns**
(integer value)

Example:

```
#For float bin "a", int(exp.FloatBin("a")) == 2
expr = exp.Eq(exp.ToInt(exp.FloatBin("a")), 2).compile()
```

### aerospike_helpers.expressions.bitwise_operators module

Bitwise operator expressions provide support for bitwise operators like & and >> in Aerospike expressions.

**class** aerospike_helpers.expressions.bitwise_operators.**IntAnd**(**exprs: TypeInteger*)

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+.

**__init__**(**exprs: TypeInteger*)

**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 = exp.Eq(exp.IntAnd(exp.IntBin("a"), 0xff), 0x11).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntArithmeticRightShift**(*value: TypeInteger*, *shift: TypeInteger*)

Create integer "arithmetic right shift" (>>) operator.

Requires server version 5.6.0+.

**__init__**(*value: TypeInteger*, *shift: TypeInteger*)

**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 = exp.GT(exp.IntArithmeticRightShift(exp.IntBin("a"), 8), 0xff).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntCount**(*value: TypeInteger*)

Create expression that returns count of integer bits that are set to 1.

Requires server version 5.6.0+.

**__init__**(*value: TypeInteger*)

**Parameters**
**value** (*TypeInteger*) – An integer value or expression to have bits counted.

**Returns**
(integer value)

Example:

```
# for int bin "a", count(a) == 4
expr = exp.Eq(exp.IntCount(exp.IntBin("a")), 4).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntLeftScan**(*value: TypeInteger*, *search: TypeBool*)

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+.

**__init__**(*value: TypeInteger*, *search: TypeBool*)

**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 = exp.GT(exp.IntLeftScan(exp.IntBin("a"), True), 4).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntLeftShift**(*value: TypeInteger*, *shift: TypeInteger*)

Create integer "left shift" (<<) operator.

Requires server version 5.6.0+.

**__init__**(*value: TypeInteger*, *shift: TypeInteger*)

**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 = exp.GT(exp.IntLeftShift(exp.IntBin("a"), 8), 0xff).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntNot**(*expr: TypeInteger*)

Create integer "not" (~) operator.

Requires server version 5.6.0+.

**__init__**(*expr: TypeInteger*)

**Parameters**
**expr** (*TypeInteger*) – An integer value or expression to be bitwise negated.

**Returns**
(integer value)

Example:

```
# for int bin "a", ~ a == 7
expr = exp.Eq(exp.IntNot(exp.IntBin("a")), 7).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntOr**(**exprs: TypeInteger*)

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+.

**__init__**(**exprs: TypeInteger*)

**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 = exp.NE(exp.IntOr(IntBin("a"), 0x10), 0).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntRightScan**(*value: TypeInteger*, *search: TypeBool*)

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+.

**__init__**(*value: TypeInteger*, *search: TypeBool*)

**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 = exp.GT(exp.IntRightScan(exp.IntBin("a"), True), 4).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntRightShift**(*value: TypeInteger*, *shift: TypeInteger*)

Create integer "logical right shift" (>>>) operator.

Requires server version 5.6.0+.

**__init__**(*value: TypeInteger*, *shift: TypeInteger*)

**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 = exp.GT(exp.IntRightShift(exp.IntBin("a"), 8), 0xff).compile()
```

**class** aerospike_helpers.expressions.bitwise_operators.**IntXOr**(**exprs: TypeInteger*)

Create integer "xor" (^) operator that is applied to two or more integers. All arguments must resolve to integers.

Requires server version 5.6.0+.

**__init__**(**exprs: TypeInteger*)

**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 = exp.Eq(exp.IntXOr(exp.IntBin("a"), exp.IntBin("b")), 16).compile()
```

### aerospike_helpers.expressions.resources module

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**

### 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:

```
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)]}
client = aerospike.client(config).connect()

key = ("test", "demo", "foo")
listWithMaps = [
    {"name": "John", "id": 100},
    {"name": "Bill", "id": 200}
]
binName = "users"

# Write the record
client.put(key, {binName: listWithMaps})

# Example 1: read the id of the second person on the list
# Get context of the second person
ctx = [cdt_ctx.cdt_ctx_list_index(1)]
ops = [
    map_operations.map_get_by_key(
        binName, "id", aerospike.MAP_RETURN_VALUE, ctx
    )
]

_, _, result = client.operate(key, ops)
print(result)
# {'users': 200}

# Example 2: add a new person and get their rating of Facebook
cindy = {
    "name": "Cindy",
    "id": 300,
    "ratings": {
        "Facebook": 4,
        "Snapchat": 5
    }
}

# Context list used for read operation after adding Cindy
# Cindy will be the third person (index 2)
# Then go to their ratings
ctx = [cdt_ctx.cdt_ctx_list_index(2), cdt_ctx.cdt_ctx_map_key("ratings")]
ops = [
    list_operations.list_append(binName, cindy),
    map_operations.map_get_by_key(
        binName, "Facebook", aerospike.MAP_RETURN_VALUE, ctx
    )
```

(continues on next page)

(continued from previous page)

```
]

_, _, result = client.operate(key, ops)
print(result)
# {'users': 4}

# Example 3: create a CDT secondary index from a base64 encoded _cdt_ctx with info␣
↪command
policy = {}

bs_b4_cdt = client.get_cdtctx_base64(ctx_list_index)

r = []
r.append("sindex-create:ns=test;set=demo;indexname=test_string_list_cdt_index")
# use index_type_string to convert enum value to string
r.append(";indextype=%s" % (cdt_ctx.index_type_string(aerospike.INDEX_TYPE_LIST)))
# use index_datatype_string to convert enum value to string
r.append(";indexdata=string_list,%s" % (cdt_ctx.index_datatype_string(aerospike.INDEX_
↪STRING)))
r.append(";context=%s" % (bs_b4_cdt))
req = ''.join(r)

# print("req is ============{}", req)
retobj = client.info_all(req, policy=None)
# print("res is ============{}", res)
client.index_remove('test', 'test_string_list_cdt_index', policy)

# Cleanup
client.remove(key)
client.close()
```

aerospike_helpers.cdt_ctx.**cdt_ctx_list_index**(*index*)

Creates a nested cdt_ctx object 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**
_cdt_ctx

aerospike_helpers.cdt_ctx.**cdt_ctx_list_index_create**(*index: int*, *order: int = 0*, *pad: bool = False*) → _cdt_ctx

Creates a nested cdt_ctx object to create an list and insert at a given index.

If a list already exists at the index, a new list will not be created. Any operations using this cdt_ctx object will be applied to the existing list.

If a non-list element exists at the index, an *InvalidRequest* will be thrown.

**Parameters**

- **key** (*object*) – The index to create the list at.

- **order** (*int*) – The *sort order* to create the List with. (default: `aerospike.LIST_UNORDERED`)
- **pad** (*bool*) – If index is out of bounds and `pad` is `True`, then the list will be created at the index with `None` elements inserted behind it. `pad` is only compatible with unordered lists.

**Returns**
`_cdt_ctx`

`aerospike_helpers.cdt_ctx.`**cdt_ctx_list_rank**(*rank*)

Creates a nested cdt_ctx object 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**
`_cdt_ctx`

`aerospike_helpers.cdt_ctx.`**cdt_ctx_list_value**(*value*)

Creates a nested cdt_ctx object to lookup an object in a list by value.

**Parameters**
**value** (*object*) – The value to look for in the list.

**Returns**
`_cdt_ctx`

`aerospike_helpers.cdt_ctx.`**cdt_ctx_map_index**(*index*)

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**
`_cdt_ctx`

`aerospike_helpers.cdt_ctx.`**cdt_ctx_map_key**(*key*)

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**
`_cdt_ctx`

`aerospike_helpers.cdt_ctx.`**cdt_ctx_map_key_create**(*key: any*, *order: int = 0*) → _cdt_ctx

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 List with. (default: `aerospike.MAP_UNORDERED`)

**Returns**
`_cdt_ctx`

aerospike_helpers.cdt_ctx.**cdt_ctx_map_rank**(*rank*)

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**
_cdt_ctx

aerospike_helpers.cdt_ctx.**cdt_ctx_map_value**(*value*)

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**
_cdt_ctx

aerospike_helpers.cdt_ctx.**index_datatype_string**(*index_datatype*)

Converts index_datatype enum value to string.

**Parameters**
**index_datatype** (*int*) – The index_datatype to convert into equivalent string value.

**Returns**
(string) - must be one of must be one of 'numeric', 'string', 'geo2dsphere'

aerospike_helpers.cdt_ctx.**index_type_string**(*index_type*)

Converts index_type enum value to string.

**Parameters**
**index_type** (*int*) – The index_type to convert into equivalent string value.

**Returns**
(string) - must be one of 'default', 'list', 'mapkeys', 'mapvalues'

## aerospike_helpers.batch package

## aerospike_helpers.batch.records module

Classes for the use with client batch APIs `batch_write()`, `batch_operate()`, `batch_apply()`, `batch_remove()`.

**class** aerospike_helpers.batch.records.**Apply**(*key: tuple*, *module: str*, *function: str*, *args: List[Any]*, *policy: Optional[Dict] = None*)

Bases: *BatchRecord*

BatchApply is used for executing Batch UDF (user defined function) apply operations with batch_write and retrieving results.

**key**

The aerospike key to operate on.

**Type**
tuple

**module**

Name of the lua module previously registered with the server.

**Type**
str

**function**

Name of the UDF to invoke.

**Type**
str

**args**

List of arguments to pass to the UDF.

**Type**
list

**record**

The record corresponding to the requested key.

**Type**
*Record Tuple*

**result**

The status code of the operation.

**Type**
int

**in_doubt**

Is it possible that the write transaction completed even though an error was generated. This may be the case when a client error occurs (like timeout) after the command was sent to the server.

**Type**
bool

**ops**

A list of aerospike operation dictionaries to perform on the record at key.

**Type**
*aerospike_helpers.operations package*

**policy**

An optional dictionary of batch apply policy flags.

**Type**
*Batch Apply Policies*, optional

**__init__**(*key: tuple*, *module: str*, *function: str*, *args: List[Any]*, *policy: Optional[Dict] = None*) → None

Example:

```
# Create a batch Apply to apply UDF "test_func" to bin "a" from the record.
# Assume that "test_func" takes a bin name string as an argument.
# Assume the appropriate UDF module has already been registered.
import aerospike_helpers.operations as op


module = "my_lua"
```

(continues on next page)

(continued from previous page)

```
function = "test_func"

bin_name = "a"
args = [
    bin_name
]

namespace = "test"
set = "demo"
user_key = 1
key = (namespace, set, user_key)

ba = Apply(key, module, function, args)
```

**class** aerospike_helpers.batch.records.**BatchRecord**(*key: tuple*)

Bases: object

BatchRecord provides the base fields for BatchRecord objects.

BatchRecord should usually be read from as a result and not created by the user. Its subclasses can be used as input to batch_write. Client methods `batch_apply()`, `batch_operate()`, `batch_remove()` with batch_records field as a list of these BatchRecord objects containing the batch request results.

**key**

The aerospike key to operate on.

**Type**
tuple

**record**

The record corresponding to the requested key.

**Type**
*Record Tuple*

**result**

The status code of the operation.

**Type**
int

**in_doubt**

Is it possible that the write transaction completed even though an error was generated. This may be the case when a client error occurs (like timeout) after the command was sent to the server.

**Type**
bool

**__init__**(*key: tuple*) → None

**__weakref__**

list of weak references to the object (if defined)

**class** aerospike_helpers.batch.records.**BatchRecords**(*batch_records: Optional[List[*BatchRecord*]] = None*)

Bases: object

BatchRecords is used as input and output for multiple batch APIs.

**batch_records**

A list of BatchRecord subtype objects used to define batch operations and hold results. BatchRecord Types can be Remove, Write, Read, and Apply.

**Type**
list

**result**

The status code of the last batch call that used this BatchRecords.

**Type**
int

**0 if all batch subtransactions succeeded**

**Type**
or if the only failures were FILTERED_OUT or RECORD_NOT_FOUND

**non 0 if an error occured. The most common error being -16**

**Type**
One or more batch sub transactions failed

**__init__**(*batch_records: Optional[List[*BatchRecord*]] = None*) → None

Example:

```
# Create a BatchRecords to remove a record, write a bin, and read a bin.
# Assume client is an instantiated and connected aerospike cleint.
import aerospike_helpers.operations as op


namespace = "test"
set = "demo"
bin_name = "id"
keys = [
    (namespace, set, 1),
    (namespace, set, 2),
    (namespace, set, 3)
]

brs = BatchRecords(
    [
        Remove(
            key=(namespace, set, 1),
        ),
        Write(
            key=(namespace, set, 100),
            ops=[
                op.write(bin_name, 100),
                op.read(bin_name),
            ]
        ),
        BatchRead(
            key=(namespace, set, 333),
            ops=[
                op.read(bin_name)
```

(continues on next page)

(continued from previous page)

```
            ]
        )
    ]
)

# Note this call will mutate brs and set results in it.
client.batch_write(brs)
```

**__weakref__**

list of weak references to the object (if defined)

**class** aerospike_helpers.batch.records.**Read**(*key: tuple*, *ops: Optional[List[Dict]]*, *read_all_bins: bool = False*, *meta: Optional[dict] = None*, *policy: Optional[Dict] = None*)

Bases: *BatchRecord*

Read is used for executing Batch read operations with batch_write and retrieving results.

**key**

The aerospike key to operate on.

**Type**
tuple

**record**

The record corresponding to the requested key.

**Type**
tuple

**result**

The status code of the operation.

**Type**
int

**in_doubt**

Is it possible that the write transaction completed even though an error was generated. This may be the case when a client error occurs (like timeout) after the command was sent to the server.

**Type**
bool

**ops**

list of aerospike operation dictionaries to perform on the record at key.

**Type**
*aerospike_helpers.operations package*

**meta**

the metadata to set for the operations in this BatchRecord

**Type**
dict

**read_all_bins**

An optional bool, if True, read all bins in the record.

**Type**
bool, optional

**policy**

An optional dictionary of batch read policy flags.

**Type**
*Batch Read Policies*, optional

**__init__**(*key: tuple*, *ops: Optional[List[Dict]]*, *read_all_bins: bool = False*, *meta: Optional[dict] = None*, *policy: Optional[Dict] = None*) → None

Example:

```
# Create a batch Read to read bin "a" from the record.
import aerospike
import aerospike_helpers.operations as op
from aerospike_helpers.batch.records import Read

bin_name = "a"

namespace = "test"
set = "demo"
user_key = 1
key = (namespace, set, user_key)

ops = [
    op.read(bin_name)
]

meta={"gen": 1, "ttl": aerospike.TTL_NEVER_EXPIRE}
br = Read(key, ops, meta=meta)
```

**class** aerospike_helpers.batch.records.**Remove**(*key: tuple*, *policy: Optional[Dict] = None*)

Bases: *BatchRecord*

Remove is used for executing Batch remove operations with batch_write and retrieving results.

**key**

The aerospike key to operate on.

**Type**
tuple

**record**

The record corresponding to the requested key.

**Type**
*Record Tuple*

**result**

The status code of the operation.

**Type**
int

**in_doubt**

Is it possible that the write transaction completed even though an error was generated. This may be the case when a client error occurs (like timeout) after the command was sent to the server.

**Type**
bool

**ops**

A list of aerospike operation dictionaries to perform on the record at key.

**Type**
*aerospike_helpers.operations package*

**policy**

An optional dictionary of batch remove policy flags.

**Type**
*Batch Remove Policies*, optional

**__init__**(*key: tuple*, *policy: Optional[Dict] = None*) → None

Example:

```
# Create a batch Remove to remove the record.
import aerospike_helpers.operations as op


namespace = "test"
set = "demo"
user_key = 1
key = (namespace, set, user_key)

br = Remove(key, ops)
```

**class** aerospike_helpers.batch.records.**Write**(*key: tuple*, *ops: List[Dict]*, *meta: Optional[dict] = None*, *policy: Optional[Dict] = None*)

Bases: *BatchRecord*

Write is used for executing Batch write operations with batch_write and retrieving batch write results.

**key**

The aerospike key to operate on.

**Type**
tuple

**record**

The record corresponding to the requested key.

**Type**
tuple

**result**

The status code of the operation.

**Type**
int

**in_doubt**

Is it possible that the write transaction completed even though an error was generated. This may be the case when a client error occurs (like timeout) after the command was sent to the server.

**Type**
bool

**ops**

A list of aerospike operation dictionaries to perform on the record at key.

**Type**
*aerospike_helpers.operations package*

**meta**

the metadata to set for the operations in this BatchRecord

**Type**
dict

**policy**

An optional dictionary of batch write policy flags.

**Type**
*Batch Write Policies*, optional

**__init__**(*key: tuple*, *ops: List[Dict]*, *meta: Optional[dict] = None*, *policy: Optional[Dict] = None*) → None

Example:

```
# Create a batch Write to increment bin "a" by 10 and read the result from the␣
↪record.
import aerospike
import aerospike_helpers.operations as op
from aerospike_helpers.batch.records import Write

bin_name = "a"

namespace = "test"
set = "demo"
user_key = 1
key = (namespace, set, user_key)

ops = [
    op.increment(bin_name, 10),
    op.read(bin_name)
]

meta={"gen": 1, "ttl": aerospike.TTL_NEVER_EXPIRE}
bw = Write(key, ops, meta=meta)
```

### 1.1.10 Python Data Mappings

#### How Python types map to server types

By default, the *Client* maps the supported Python types to Aerospike server types. When an unsupported type is encountered by the module, it uses cPickle to serialize and deserialize the data, storing it in the server as a blob with 'Python encoding' (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 in the server with generic encoding (AS_BYTES_BLOB). This type allows the storage of binary data readable by Aerospike Clients in other languages. The *serialization* config parameter 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`. It 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. For example, if there is a client that reads and writes booleans as integers, then another Python client working with the same data should do the same thing.

`send_bool_as` can be set so the client writes Python booleans as `AS_BYTES_PYTHON`, integers, 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.

| 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 |
| `bytes` | blob |
| *`aerospike.GeoJSON`* | GeoJSON |

**Note:** *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*).

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.

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.

## 1.2 Indices and tables

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## a

# INDEX

## Symbols

## A

## B

## C



## D

## I

## J



## K



## L

## M

## N



## O



## P

## Q



## R



## S

## T



## U



## V



## W