etcd3-py Documentation

Release 0.1.6

Renjie Cai

Apr 01, 2020

Contents

Python client for etcd v3 (Using gRPC-JSON-Gateway)

- Free software: Apache Software License 2.0
- Source Code: https://github.com/Revolution1/etcd3-py
- Documentation: https://etcd3-py.readthedocs.io.
- etcd version required: v3.2.2+

Notice: The authentication header through gRPC-JSON-Gateway only supported in etcd v3.3.0+

CHAPTER 1

Features

- [x] Support python2.7 and python3.5+ (aiohttp requires python3.5.2+)
- [x] Sync client based on requests
- [x] Async client based on aiohttp
- [x] TLS Connection
- [x] support APIs
  - [x] Auth
  - [x] KV
  - [x] Watch
  - [x] Cluster
  - [x] Lease
  - [x] Lock
  - [x] Maintenance
  - [x] Extra APIs
- [x] stateful utilities
  - [x] Watch
  - [x] Lease
  - [x] Transaction
  - [x] Lock

CHAPTER 2

Quick Start

Install

```
$ pip install --upgrade etcd3-py
```

Sync Client

```
>>> from etcd3 import Client
>>> client = Client('127.0.0.1', 2379, cert=(CERT_PATH, KEY_PATH), verify=CA_PATH)
>>> client.version()
EtcdVersion(etcdserver='3.3.0-rc.4', etcdcluster='3.3.0')
>>> client.put('foo', 'bar')
etcdserverpbPutResponse(header=etcdserverpbResponseHeader(cluster_
↪id=11588568905070377092, member_id=128088275939295631, revision=15433, raft_term=4))
>>> client.range('foo').kvs
[mvccpbKeyValue(key=b'foo', create_revision=15429, mod_revision=15433, version=5,␣
↪value=b'bar')]
```

Async Client (Python3.5+)

```
>>> import asyncio
>>> from etcd3 import AioClient
>>> client = AioClient('127.0.0.1', 2379)
>>> async def getFoo():
...     await client.put('foo', 'bar')
...     r = await client.range('foo')
...     print('key:', r.kvs[0].key, 'value:', r.kvs[0].value)
>>> loop = asyncio.get_event_loop()
>>> loop.run_until_complete(getFoo())
key: b'foo' value: b'bar'
```

Transaction Util

```
>>> from etcd3 import Client
>>> txn = Client().Txn()
```

(continues on next page)

(continued from previous page)

```
>>> txn.compare(txn.key('foo').value == 'bar')
>>> txn.success(txn.put('foo', 'bra'))
>>> txn.commit()
etcdserverpbTxnResponse(header=etcdserverpbResponseHeader(cluster_
↪id=11588568905070377092, member_id=128088275939295631, revision=15656, raft_term=4),
↪ succeeded=True, responses=[etcdserverpbResponseOp(response_
↪put=etcdserverpbPutResponse(header=etcdserverpbResponseHeader(revision=15656)))])
```

Lease Util

```
>>> from etcd3 import Client
>>> client = Client()
>>> with client.Lease(ttl=5) as lease:
...     client.put('foo', 'bar', lease=lease.ID)
...     client.put('fizz', 'buzz', lease=lease.ID)
...     r = lease.time_to_live(keys=True)
...     assert set(r.keys) == {b'foo', b'fizz'}
...     assert lease.alive()
```

Watch Util

```
>>> from etcd3 import Client
>>> client = Client()
>>> watcher = c.Watcher(all=True, progress_notify=True, prev_kv=True)
>>> w.onEvent('f.*', lambda e: print(e.key, e.value))
>>> w.runDaemon()
>>> # etcdctl put foo bar
>>> # etcdctl put foz bar
b'foo' b'bar'
b'foz' b'bar'
>>> w.stop()
```

Lock Util

```
>>> import time
>>> from threading import Thread
>>> from etcd3 import Client
>>> client = Client()
>>> name = 'lock_name'
>>> def user1():
...     with client.Lock(name, lock_ttl=5):
...         print('user1 got the lock')
...         time.sleep(5)
...         print('user1 releasing the lock')
>>> def user2():
...     with client.Lock(name, lock_ttl=5):
...         print('user2 got the lock')
...         time.sleep(5)
...         print('user2 releasing the lock')
>>> t1 = Thread(target=user1, daemon=True)
>>> t2 = Thread(target=user2, daemon=True)
>>> t1.start()
>>> t2.start()
>>> t1.join()
>>> t2.join()
user1 got the lock
user1 releasing the lock
```

(continues on next page)

(continued from previous page)

```
user2 got the lock
user2 releasing the lock
```

Start a single-node etcd using docker

```
export NODE1=0.0.0.0
export ETCD_VER=v3.3
docker run -d \
-p 2379:2379 \
-p 2380:2380 \
--volume=/tmp/etcd3-data:/etcd-data \
--name etcd3 quay.io/coreos/etcd:$ETCD_VER \
/usr/local/bin/etcd \
--data-dir=/etcd-data --name node1 \
--initial-advertise-peer-urls http://${NODE1}:2380 --listen-peer-urls http://${NODE1}
↪:2380 \
--advertise-client-urls http://${NODE1}:2379 --listen-client-urls http://${NODE1}
↪:2379 \
--initial-cluster node1=http://${NODE1}:2380
```

CHAPTER 3

FAQ

Q: authentication seems not working? Try calling api of a auth-enabled etcd server returned error "ErrUserEmpty error:'etcdserver: user name is empty'"

A: Take a look at #41, currently etcd3-py dose not authenticate automatically, you need to call client.auth() by yourself.

CHAPTER 4

TODO

- [ ] human friendly middle level apis
- [ ] able to expose json or raw response to user
- [ ] add election api
- [ ] benchmark
- [ ] python-etcd(etcd v2) compatible client
- [ ] etcd browser
- [ ] support etcd v3.4.x

CHAPTER 5

Instructions

5.1 Installation

5.1.1 Stable release

To install etcd3-py, run this command in your terminal:

```
$ pip install etcd3
```

This is the preferred method to install etcd3-py, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

5.1.2 From sources

The sources for etcd3-py can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/revolution1/etcd3
```

Or download the tarball:

```
$ curl  -OL https://github.com/revolution1/etcd3/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

5.2 Usage

To use etcd3-py in a project:

```
import etcd3
```

5.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.3.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/revolution1/etcd3/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

etcd3-py could always use more documentation, whether as part of the official etcd3-py docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/revolution1/etcd3/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.3.2 Get Started!

Ready to contribute? Here's how to set up *etcd3* for local development.

1. Fork the *etcd3* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/etcd3.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv etcd3
$ cd etcd3/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 etcd3 tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/revolution1/etcd3/pull_requests and make sure that the tests pass for all supported Python versions.

5.3.4 Tips

To run a subset of tests:

```
$ py.test tests.test_etcd3
```

5.4 Credits

5.4.1 Development Lead

- Renjie Cai <revol.cai@gmail.com>

5.4.2 Contributors

None yet. Why not be the first?

5.5 History

5.5.1 0.1.6 (2019-05-9)

- merge pull request #90 Fix lease util keeping problems
- merge pull request #89 Add range end and lease to txn
- merge pull request #87 Add handel null value as gogoproto does while modelizing response data
- merge pull request #82 Fix watch util issue #18 and #78
- merge pull request #79 Improve etcd comapabitity of multiple versions
- merge pull request #51 Add a base EtcdModel to all dynamic created model
- merge pull request #50 Add support for lock service API
- merge pull request #42 Improve etcd comapabitity of multiple versions

5.5.2 0.1.5 (2018-07-4)

- merge pull request #34 enum34 only where it's needed

5.5.3 0.1.4 (2018-03-30)

- better code quality
- support etcd v3.2.2+

5.5.4 0.1.3 (2018-03-21)

- finished lock util

5.5.5 0.1.2 (2018-03-20)

- Add more test
- Add watcher, transaction and lease util

You can try it at dev environment

5.5.6 0.1.0 (2018-03-19)

- Implemented all APIs of etcd3's gRPC-JSON-Gateway
- Stateful utils (Watcher Lease Lock Transaction) are in progress

CHAPTER 6

API Reference

6.1 API Reference

6.1.1 etcd3.baseclient

synchronous client

class etcd3.baseclient.**BaseModelizedStreamResponse**

Bases: object

Model of a stream response

close()

close the stream

class etcd3.baseclient.**BaseClient**(*host='127.0.0.1'*, *port=2379*, *protocol='http'*, *cert=()*, *verify=None*, *timeout=None*, *headers=None*, *user_agent=None*, *pool_size=30*, *username=None*, *password=None*, *token=None*, *server_version='3.3.0'*, *cluster_version='3.3.0'*)

Bases: *etcd3.apis.auth.AuthAPI*, *etcd3.apis.cluster.ClusterAPI*, *etcd3.apis.kv.KVAPI*, *etcd3.apis.lease.LeaseAPI*, *etcd3.apis.maintenance.MaintenanceAPI*, *etcd3.apis.watch.WatchAPI*, *etcd3.apis.extra.ExtraAPI*, *etcd3.apis.lock.LockAPI*

__init__(*host='127.0.0.1'*, *port=2379*, *protocol='http'*, *cert=()*, *verify=None*, *timeout=None*, *headers=None*, *user_agent=None*, *pool_size=30*, *username=None*, *password=None*, *token=None*, *server_version='3.3.0'*, *cluster_version='3.3.0'*)

Initialize self. See help(type(self)) for accurate signature.

baseurl

Returns baseurl from protocol, host, self

__enter__()

__exit__(**args*)

close()
close all connections in connection pool

call_rpc(*method*, *data=None*, *stream=False*, *encode=True*, *raw=False*, ***kwargs*)
call ETCDv3 RPC and return response object

Parameters

- **method** (*str*) – the rpc method, which is a path of RESTful API
- **data** (*dict*) – request payload to be post to ETCD's gRPC-JSON-Gateway default: {}
- **stream** (*bool*) – whether return a stream response object, default: False
- **encode** (*bool*) – whether encode the data before post, default: True
- **kwargs** – additional params to pass to the http request, like headers, timeout etc.

Returns Etcd3RPCResponseModel or Etcd3StreamingResponse

auth(*username=None*, *password=None*)
call auth.authenticate and save the token

Parameters

- **username** (*str*) – username
- **password** (*str*) – password

Txn()
Initialize a Transaction

Lease(*ttl*, *ID=0*, *new=True*)
Initialize a Lease

Parameters

- **ID** (*int*) – ID is the requested ID for the lease. If ID is set to 0, the lessor chooses an ID.
- **new** (*bool*) – whether grant a new lease or maintain a exist lease by its id [default: True]

Watcher(*key=None*, *range_end=None*, *max_retries=-1*, *start_revision=None*, *progress_notify=None*, *prev_kv=None*, *prefix=None*, *all=None*, *no_put=False*, *no_delete=False*)
Initialize a Watcher

Parameters

- **key** (*str or bytes*) – key is the key to register for watching.
- **range_end** (*str or bytes*) – range_end is the end of the range [key, range_end) to watch. If range_end is not given, only the key argument is watched. If range_end is equal to '', all keys greater than or equal to the key argument are watched. If the range_end is one bit larger than the given key, then all keys with the prefix (the given key) will be watched.
- **max_retries** (*int*) – max retries when watch failed due to network problem, -1 means no limit [default: -1]
- **start_revision** (*int*) – start_revision is an optional revision to watch from (inclusive). No start_revision is "now".
- **progress_notify** (*bool*) – progress_notify is set so that the etcd server will periodically send a WatchResponse with no events to the new watcher if there are no recent events. It is useful when clients wish to recover a disconnected watcher starting from a recent known revision. The etcd server may decide how often it will send notifications based on current load.

- **prev_kv** (*bool*) – If prev_kv is set, created watcher gets the previous KV before the event happens. If the previous KV is already compacted, nothing will be returned.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]
- **no_put** (*bool*) – filter out the put events at server side before it sends back to the watcher. [default: False]
- **no_delete** (*bool*) – filter out the delete events at server side before it sends back to the watcher. [default: False]

Returns Watcher

Lock(*lock_name*, *lock_ttl=60*, *reentrant=None*, *lock_prefix='_locks'*)

6.1.2 etcd3.client

synchronous client

class etcd3.client.**ModelizedStreamResponse**(*client*, *method*, *resp*, *decode=True*)

Bases: *etcd3.baseclient.BaseModelizedStreamResponse*

Model of a stream response

__init__(*client*, *method*, *resp*, *decode=True*)

Parameters **resp** – Response

raw

close()

close the stream

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

etcd3.client.**iter_response**(*resp*)

yield response content by every json object we don't yield by line, because the content of etcd's gRPC-JSON-Gateway stream response does not have a delimiter between each object by default. (only one line)

https://github.com/grpc-ecosystem/grpc-gateway/pull/497/files

Parameters **resp** – Response

Returns dict

class etcd3.client.**Client**(*host='127.0.0.1'*, *port=2379*, *protocol='http'*, *cert=()*, *verify=None*, *timeout=None*, *headers=None*, *user_agent=None*, *pool_size=30*, *username=None*, *password=None*, *token=None*, *max_retries=0*, *server_version='3.3.0'*, *cluster_version='3.3.0'*)

Bases: *etcd3.baseclient.BaseClient*

__init__(*host='127.0.0.1'*, *port=2379*, *protocol='http'*, *cert=()*, *verify=None*, *timeout=None*, *headers=None*, *user_agent=None*, *pool_size=30*, *username=None*, *password=None*, *token=None*, *max_retries=0*, *server_version='3.3.0'*, *cluster_version='3.3.0'*)

Parameters **max_retries** – The maximum number of retries each connection should attempt. Note, this applies only to failed DNS lookups, socket connections and connection timeouts, never to requests where data has made it to the server. By default, Requests does not retry failed connections. If you need granular control over the conditions under which we retry a request, import urllib3's `Retry` class and pass that instead.

close()
close all connections in connection pool

call_rpc(*method*, *data=None*, *stream=False*, *encode=True*, *raw=False*, ***kwargs*)
call ETCDv3 RPC and return response object

Parameters

- **method** (*str*) – the rpc method, which is a path of RESTful API
- **data** (*dict*) – request payload to be post to ETCD's gRPC-JSON-Gateway default: {}
- **stream** (*bool*) – whether return a stream response object, default: False
- **encode** (*bool*) – whether encode the data before post, default: True
- **kwargs** – additional params to pass to the http request, like headers, timeout etc.

Returns Etcd3RPCResponseModel or Etcd3StreamingResponse

auth(*username=None*, *password=None*)
call auth.authenticate and save the token

Parameters

- **username** (*str*) – username
- **password** (*str*) – password

6.1.3 etcd3.aio_client

asynchronous client

class etcd3.aio_client.**ModelizedResponse**(*client*, *method*, *resp*, *decode=True*)
Bases: object

__init__(*client*, *method*, *resp*, *decode=True*)
Initialize self. See help(type(self)) for accurate signature.

class etcd3.aio_client.**ModelizedStreamResponse**(*client*, *method*, *resp*, *decode=True*)
Bases: *etcd3.baseclient.BaseModelizedStreamResponse*

Model of a stream response

__init__(*client*, *method*, *resp*, *decode=True*)

Parameters **resp** – aiohttp.ClientResponse

connection

resp_iter

close()
close the stream

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

__aenter__()

__aexit__(*exc_type*, *exc_val*, *exc_tb*)

class etcd3.aio_client.**ResponseIter**(*resp*)
Bases: object

yield response content by every json object we don't yield by line, because the content of etcd's gRPC-JSON-Gateway stream response does not have a delimiter between each object by default. (only one line)

https://github.com/grpc-ecosystem/grpc-gateway/pull/497/files

Parameters **resp** – aiohttp.ClientResponse

Returns dict

__init__(*resp*)
Initialize self. See help(type(self)) for accurate signature.

next()

class etcd3.aio_client.**AioClient**(*host='127.0.0.1'*, *port=2379*, *protocol='http'*, *cert=()*, *verify=None*, *timeout=None*, *headers=None*, *user_agent=None*, *pool_size=30*, *username=None*, *password=None*, *token=None*, *server_version='3.3.0'*, *cluster_version='3.3.0'*)
Bases: *etcd3.baseclient.BaseClient*

__init__(*host='127.0.0.1'*, *port=2379*, *protocol='http'*, *cert=()*, *verify=None*, *timeout=None*, *headers=None*, *user_agent=None*, *pool_size=30*, *username=None*, *password=None*, *token=None*, *server_version='3.3.0'*, *cluster_version='3.3.0'*)
Initialize self. See help(type(self)) for accurate signature.

session

call_rpc(*method*, *data=None*, *stream=False*, *encode=True*, *raw=False*, ***kwargs*)
call ETCDv3 RPC and return response object

Parameters

- **method** (*str*) – the rpc method, which is a path of RESTful API
- **data** (*dict or str*) – request payload to be post to ETCD's gRPC-JSON-Gateway default: {}
- **stream** (*bool*) – whether return a stream response object, default: False
- **encode** (*bool*) – whether encode the data before post, default: True
- **kwargs** – additional params to pass to the http request, like headers, timeout etc.

Returns Etcd3RPCResponseModel or Etcd3StreamingResponse

__aenter__()

__aexit__(*exc_type*, *exc_val*, *exc_tb*)

auth(*username=None*, *password=None*)
call auth.authenticate and save the token

Parameters

- **username** (*str*) – username
- **password** (*str*) – password

close()
close all connections in connection pool

6.1.4 etcd3.stateful

etcd3.stateful.lease

Sync lease util

class etcd3.stateful.lease.**Lease**(*client*, *ttl*, *ID=0*, *new=True*)
Bases: object

__init__(*client*, *ttl*, *ID=0*, *new=True*)

Parameters

- **client** (BaseClient) – client instance of etcd3
- **ID** (*int*) – ID is the requested ID for the lease. If ID is set to 0, the lessor chooses an ID.
- **new** (*bool*) – whether grant a new lease or maintain a exist lease by its id [default: True]

ID
Property: the id of the granted lease

Returns int

grant()
Grant the lease if new is set to False or it just inherit the lease of the specified id

When granting new lease if ID is set to 0, the lessor will chooses an ID.

time_to_live(*keys=False*)
Retrieves lease information.

Parameters **keys** (*bool*) – whether return the keys that attached to the lease

ttl()
Get the ttl that lease has left

Returns int

alive()
Tell if the lease is still alive

Returns bool

keepalive_once()
Call keepalive for once to refresh the ttl of the lease

refresh()
Call keepalive for once to refresh the ttl of the lease

keepalive(*keep_cb=None*, *cancel_cb=None*)
Start a daemon thread to constantly keep the lease alive

Parameters

- **keep_cb** (*callable*) – callback function that will be called after every refresh
- **cancel_cb** (*callable*) – callback function that will be called after cancel keepalive

cancel_keepalive(*join=True*)
stop keeping-alive

Parameters **join** (*bool*) – whether to wait the keepalive thread to exit

jammed()
if is failed to keepalive at the last loop

revoke()
revoke the lease

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

etcd3.stateful.transaction

class etcd3.stateful.transaction.**Txn**(*client*, *compare=None*, *success=None*, *failure=None*)
Bases: object

Txn (transaction) util provides a human friendly way to build kv.txn request Usage:

```
>>> from etcd3 import Client, Txn
>>> txn = Txn(Client())
>>> txn.compare(txn.key('foo').value == 'bar')
>>> txn.success(txn.put('foo', 'bra'))
>>> txn.commit()
etcdserverpbTxnResponse(header=etcdserverpbResponseHeader(cluster_
↪id=11588568905070377092, member_id=128088275939295631, revision=15656, raft_
↪term=4), succeeded=True, responses=[etcdserverpbResponseOp(response_
↪put=etcdserverpbPutResponse(header=etcdserverpbResponseHeader(revision=15656)))])
```

From google paxosdb paper:

Our implementation hinges around a powerful primitive which we call MultiOp. All other database operations except for iteration are implemented as a single call to MultiOp. A MultiOp is applied atomically and consists of three components:

1. A list of tests called guard. Each test in guard checks a single entry in the database. It may check for the absence or presence of a value, or compare with a given value. Two different tests in the guard may apply to the same or different entries in the database. All tests in the guard are applied and MultiOp returns the results. If all tests are true, MultiOp executes t op (see item 2 below), otherwise it executes f op (see item 3 below).
2. A list of database operations called t op. Each operation in the list is either an insert, delete, or lookup operation, and applies to a single database entry. Two different operations in the list may apply to the same or different entries in the database. These operations are executed if guard evaluates to true.
3. A list of database operations called f op. Like t op, but executed if guard evaluates to false.

__init__(*client*, *compare=None*, *success=None*, *failure=None*)

Parameters

- **client** (BaseClient) – the instance of etcd3's client
- **compare** – guard components of the transaction, default to []
- **success** – success components of the transaction, default to []
- **failure** – failure components of the transaction, default to []

clear()
clear all ops

compare(*compareOp*)
Add a test to the transaction guard component

Parameters **compareOp** – TxnCompareOp

Returns self

If(*compareOp*)
Add a test to the transaction guard component

Parameters **compareOp** – TxnCompareOp

Returns self

success(*successOp*)
Add a database operation to the transaction success component

Then(*successOp*)
Add a database operation to the transaction success component

failure(*failureOp*)
Add a database operation to the transaction failure component

Else(*failureOp*)
Add a database operation to the transaction failure component

commit()

static key(*key=None*, *range_end=None*, *prefix=None*, *all=None*)
Get the TxnCompareOp of a key

Parameters

- **key** (*str or bytes*) – key is the subject key for the comparison operation.
- **range_end** (*str or bytes*) – range_end is the upper bound on the requested range [key, range_end). If range_end is ‘’, the range is all keys >= key. If range_end is key plus one (e.g., “aa”+1 == “ab”, “aÿ”+1 == “b”), then the range request gets all keys prefixed with key. If both key and range_end are ‘’, then the range request returns all keys.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]

Returns TxnCompareOp

static range(*key=None*, *range_end=None*, *limit=0*, *revision=None*, *serializable=False*, *keys_only=False*, *count_only=False*, *min_mod_revision=None*, *max_mod_revision=None*, *min_create_revision=None*, *max_create_revision=None*, *sort_order=<RangeRequestSortOrder.NONE: 'NONE'>*, *sort_target=<RangeRequestSortTarget.KEY: 'KEY'>*, *prefix=None*, *all=None*)
Operation of keys in the range from the key-value store.

Parameters

- **key** (*str or bytes*) – key is the first key for the range. If range_end is not given, the request only looks up key.
- **range_end** (*str or bytes*) – range_end is the upper bound on the requested range [key, range_end). If range_end is ‘’, the range is all keys >= key. If range_end is key plus one (e.g., “aa”+1 == “ab”, “aÿ”+1 == “b”), then the range request gets all keys prefixed with key. If both key and range_end are ‘’, then the range request returns all keys.
- **limit** (*int*) – limit is a limit on the number of keys returned for the request. When limit is set to 0, it is treated as no limit.
- **revision** (*int*) – revision is the point-in-time of the key-value store to use for the range. If revision is less or equal to zero, the range is over the newest key-value store. If the revision has been compacted, ErrCompacted is returned as a response.
- **sort_order** (RangeRequestSortOrder) – sort_order is the order for returned sorted results.

- **sort_target** (RangeRequestSortTarget) – sort_target is the key-value field to use for sorting.
- **serializable** (*bool*) – serializable sets the range request to use serializable member-local reads. Range requests are linearizable by default; linearizable requests have higher latency and lower throughput than serializable requests but reflect the current consensus of the cluster. For better performance, in exchange for possible stale reads, a serializable range request is served locally without needing to reach consensus with other nodes in the cluster.
- **keys_only** (*bool*) – keys_only when set returns only the keys and not the values.
- **count_only** (*bool*) – count_only when set returns only the count of the keys in the range.
- **min_mod_revision** (*int*) – min_mod_revision is the lower bound for returned key mod revisions; all keys with lesser mod revisions will be filtered away.
- **max_mod_revision** (*int*) – max_mod_revision is the upper bound for returned key mod revisions; all keys with greater mod revisions will be filtered away.
- **min_create_revision** (*int*) – min_create_revision is the lower bound for returned key create revisions; all keys with lesser create revisions will be filtered away.
- **max_create_revision** (*int*) – max_create_revision is the upper bound for returned key create revisions; all keys with greater create revisions will be filtered away.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]

static put(*key*, *value*, *lease=0*, *prev_kv=False*, *ignore_value=False*, *ignore_lease=False*)

Operation of puts the given key into the key-value store. A put request increments the revision of the key-value store and generates one event in the event history.

Parameters

- **key** (*str or bytes*) – key is the key, in bytes, to put into the key-value store.
- **value** (*str*) – value is the value, in bytes, to associate with the key in the key-value store.
- **lease** (*int*) – lease is the lease ID to associate with the key in the key-value store. A lease value of 0 indicates no lease.
- **prev_kv** (*bool*) – If prev_kv is set, etcd gets the previous key-value pair before changing it. The previous key-value pair will be returned in the put response.
- **ignore_value** (*bool*) – If ignore_value is set, etcd updates the key using its current value. Returns an error if the key does not exist.
- **ignore_lease** (*bool*) – If ignore_lease is set, etcd updates the key using its current lease. Returns an error if the key does not exist.

static delete(*key=None*, *range_end=None*, *prev_kv=False*, *prefix=None*, *all=None*)

Operation of deletes the given range from the key-value store. A delete request increments the revision of the key-value store and generates a delete event in the event history for every deleted key.

Parameters

- **key** (*str or bytes*) – key is the first key to delete in the range.
- **range_end** (*str or bytes*) – range_end is the key following the last key to delete for the range [key, range_end). If range_end is not given, the range is defined to contain

only the key argument. If range_end is one bit larger than the given key, then the range is all the keys with the prefix (the given key). If range_end is ‘’, the range is all keys greater than or equal to the key argument.

- **prev_kv** (*bool*) – If prev_kv is set, etcd gets the previous key-value pairs before deleting it. The previous key-value pairs will be returned in the delete response.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]

clone()

Returns Txn

class etcd3.stateful.transaction.**TxnCompareOp**(*key*, *range_end=None*)
Bases: object

The operator of transaction's compare part

__init__(*key*, *range_end=None*)

Parameters **key** – the key to compare

value
represents the value of the key

mod
represents the mod_revision of the key

version
represents the version of the key

create
represents the create_revision of the key

lease
represents the lease_id of the key

ref: https://github.com/etcd-io/etcd/blob/v3.3.12/clientv3/compare.go#L87-L91 LeaseValue compares a key's LeaseID to a value of your choosing. The empty LeaseID is 0, otherwise known as *NoLease*.

to_compare()
return the compare payload that the rpc accepts

etcd3.stateful.watch

exception etcd3.stateful.watch.**OnceTimeout**
Bases: OSError

Timeout caused by watch once

class etcd3.stateful.watch.**KeyValue**(*data*)
Bases: object

Model of the key-value of the event

__init__(*data*)
Initialize self. See help(type(self)) for accurate signature.

get(*key*, *default=None*)

class etcd3.stateful.watch.**Event**(*data*, *header=None*)
Bases: *etcd3.stateful.watch.KeyValue*

Watch event

__init__(*data*, *header=None*)

Parameters

- **data** – dict data of a etcdserverpbWatchResponse.events[<mvccpbEvent>]
- **header** – the header of etcdserverpbWatchResponse

class etcd3.stateful.watch.**Watcher**(*client*, *max_retries=-1*, *key=None*, *range_end=None*, *start_revision=None*, *progress_notify=None*, *prev_kv=None*, *prefix=None*, *all=None*, *no_put=False*, *no_delete=False*)
Bases: object

__init__(*client*, *max_retries=-1*, *key=None*, *range_end=None*, *start_revision=None*, *progress_notify=None*, *prev_kv=None*, *prefix=None*, *all=None*, *no_put=False*, *no_delete=False*)
Initialize a watcher

Parameters

- **client** (BaseClient) – client instance of etcd3
- **max_retries** (*int*) – max retries when watch failed due to network problem, -1 means no limit [default: -1]
- **key** (*str or bytes*) – key is the key to register for watching.
- **range_end** (*str or bytes*) – range_end is the end of the range [key, range_end) to watch. If range_end is not given, only the key argument is watched. If range_end is equal to ‘’, all keys greater than or equal to the key argument are watched. If the range_end is one bit larger than the given key, then all keys with the prefix (the given key) will be watched.
- **start_revision** (*int*) – start_revision is an optional revision to watch from (inclusive). No start_revision is “now”.
- **progress_notify** (*bool*) – progress_notify is set so that the etcd server will periodically send a WatchResponse with no events to the new watcher if there are no recent events. It is useful when clients wish to recover a disconnected watcher starting from a recent known revision. The etcd server may decide how often it will send notifications based on current load.
- **prev_kv** (*bool*) – If prev_kv is set, created watcher gets the previous KV before the event happens. If the previous KV is already compacted, nothing will be returned.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]
- **no_put** (*bool*) – filter out the put events at server side before it sends back to the watcher. [default: False]
- **no_delete** (*bool*) – filter out the delete events at server side before it sends back to the watcher. [default: False]

set_default_timeout(*timeout*)
Set the default timeout of watch request

Parameters **timeout** (*int*) – timeout in seconds

clear_revision()
Clear the start_revision that stored in watcher

clear_callbacks()
Remove all callbacks

request_create()
Start a watch request

request_cancel()
Cancel the watcher [Not Implemented because of etcd3 returns no watch_id]

static get_filter(*filter*)
Get the event filter function

Parameters **filter** (*callable or regex string or EventType or None*) – will generate a filter function from this param

Returns callable

onEvent(*filter_or_cb*, *cb=None*)
Add a callback to a event that matches the filter

If only one param is given, which is filter_or_cb, it will be treated as the callback. If any event comes, it will be called.

Parameters

- **filter_or_cb** (*callable or regex string or EventType*) – filter or callback function
- **cb** – the callback function

unEvent(*filter=None*, *cb=None*)
remove a callback or filter event that's been previously added via onEvent() If both parameters are given they are ANDd together; to OR the, make two calls.

Parameters

- **filter** (*callable or regex string or EventType*) – the callable filter or regex string or EventType the event to be removed was registerd with
- **cb** – the callback funtion the event to be removed was registerd with

dispatch_event(*event*)
Find the callbacks, if callback's filter fits this event, call the callback

Parameters **event** – Event

run()
Run the watcher and handel events by callbacks

stop()
Stop watching, close the watch stream and exit the daemon thread

cancel()
Stop watching, close the watch stream and exit the daemon thread

runDaemon()
Run Watcher in a daemon thread

watch_once(*filter=None*, *timeout=None*)
watch the filtered event, once have event, return it if timed out, return None

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

etcd3.stateful.lock

exception etcd3.stateful.lock.**EtcdLockError**
Bases: Exception

exception etcd3.stateful.lock.**EtcdLockAcquireTimeout**
Bases: Exception

class etcd3.stateful.lock.**Lock**(*client*, *lock_name*, *lock_ttl=60*, *reentrant=None*, *lock_prefix='_locks'*)
Bases: object

Locking recipe for etcd, inspired by the kazoo recipe for zookeeper

DEFAULT_LOCK_TTL = 60

HOST = 'host'

PROCESS = 'process'

THREAD = 'thread'

__init__(*client*, *lock_name*, *lock_ttl=60*, *reentrant=None*, *lock_prefix='_locks'*)

Parameters

- **client** (BaseClient) – instance of etcd.Client
- **lock_name** (*str*) – the name of the lock
- **lock_ttl** (*int*) – ttl of the lock, default is 60s
- **reentrant** (*str*) – the reentrant type of the lock can set to Lock.HOST, Lock.PROCESS, Lock.THREAD
- **lock_prefix** (*str*) – the prefix of the lock key

holders()
tell how many holders are holding the lock

Returns int

incr_holder()
Atomic increase the holder count by 1

decr_holder()
Atomic decrease the holder count by 1

is_acquired
if the lock is acquired

acquired
if the lock is acquired

acquire(*block=True*, *lock_ttl=None*, *timeout=None*, *delete_key=True*)
Acquire the lock.

Parameters

- **block** (*bool*) – Block until the lock is obtained, or timeout is reached [default: True]
- **lock_ttl** (*int*) – The duration of the lock we acquired, set to None for eternal locks
- **timeout** (*int*) – The time to wait before giving up on getting a lock

- **delete_key** (*bool*) – whether delete the key if it has not attached to any lease [default: True]

wait (*locker=None*, *timeout=None*)
Wait until the lock is lock is able to acquire

Parameters

- **locker** – kv of the lock
- **timeout** – wait timeout

release ()
Release the lock

__enter__ ()
You can use the lock as a contextmanager

__exit__ (*type*, *value*, *traceback*)

6.1.5 etcd3.apis

etcd3.apis.auth

class etcd3.apis.auth.**AuthAPI**
Bases: *etcd3.apis.base.BaseAPI*

authenticate (*name*, *password*)
Authenticate processes an authenticate request.

Parameters

- **name** (*str*) – name of the user
- **password** (*str*) – password of the user

auth_disable ()
AuthDisable disables authentication.

auth_enable ()
AuthEnable enables authentication.

role_add (*name*)
RoleAdd adds a new role.

Parameters **name** (*str*) – name is the name of the role to add to the authentication system.

role_delete (*role*)
RoleDelete deletes a specified role.

Parameters **role** (*str*) – None

role_get (*role*)
RoleGet gets detailed role information.

Parameters **role** (*str*) – None

role_grant_permission (*name*, *key=None*, *permType=<authpbPermissionType.READ: 'READ'>*, *range_end=None*, *prefix=False*, *all=False*)
RoleGrantPermission grants a permission of a specified key or range to a specified role.

Parameters

- **name** (*str*) – name is the name of the role which will be granted the permission.

- **key** (*str or bytes*) – the key been granted to the role
- **perm** (*dict*) – authpbPermissionType.READ or authpbPermissionType.WRITE or authpbPermissionType.READWRITE
- **range_end** (*str or bytes*) – range_end is the upper bound on the requested range [key, range_end). If range_end is ‘’, the range is all keys >= key. If range_end is key plus one (e.g., “aa”+1 == “ab”, “aÿ”+1 == “b”), then the range request gets all keys prefixed with key. If both key and range_end are ‘’, then the range request returns all keys.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]

role_list()
RoleList gets lists of all roles.

role_revoke_permission(*role*, *key=None*, *range_end=None*, *prefix=False*, *all=False*)
RoleRevokePermission revokes a key or range permission of a specified role.

Parameters

- **role** (*str*) – the name of the role which will get permission revoked.
- **key** (*str or bytes*) – the key been revoked from the role
- **range_end** (*str or bytes*) – range_end is the upper bound on the requested range [key, range_end). If range_end is ‘’, the range is all keys >= key. If range_end is key plus one (e.g., “aa”+1 == “ab”, “aÿ”+1 == “b”), then the range request gets all keys prefixed with key. If both key and range_end are ‘’, then the range request returns all keys.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]

user_add(*name*, *password*)
UserAdd adds a new user.

Parameters

- **name** (*str*) – name of the user
- **password** (*str*) – password of the user

user_change_password(*name*, *password*)
UserChangePassword changes the password of a specified user.

Parameters

- **name** (*str*) – name is the name of the user whose password is being changed.
- **password** (*str*) – password is the new password for the user.

user_delete(*name*)
UserDelete deletes a specified user.

Parameters **name** (*str*) – name is the name of the user to delete.

user_get(*name*)
UserGet gets detailed user information.

Parameters **name** (*str*) – name is the name of the user to get.

user_grant_role(*user*, *role*)
UserGrant grants a role to a specified user.

Parameters

- **user** (*str*) – user is the name of the user which should be granted a given role.
- **role** (*str*) – role is the name of the role to grant to the user.

user_list()
UserList gets a list of all users.

user_revoke_role(*name*, *role*)
UserRevokeRole revokes a role of specified user.

Parameters

- **name** (*str*) – username to revoke
- **role** (*str*) – role name

etcd3.apis.base

class etcd3.apis.base.**BaseAPI**
Bases: object

__init__()
Initialize self. See help(type(self)) for accurate signature.

call_rpc(*method*, *data=None*, *stream=False*, *encode=True*, *raw=False*, ***kwargs*)

etcd3.apis.cluster

class etcd3.apis.cluster.**ClusterAPI**
Bases: *etcd3.apis.base.BaseAPI*

member_add(*peerURLs*)
MemberAdd adds a member into the cluster.

Parameters **peerURLs** (*list of str*) – peerURLs is the list of URLs the added member will use to communicate with the cluster.

member_list()
MemberList lists all the members in the cluster.

member_remove(*ID*)
MemberRemove removes an existing member from the cluster.

Parameters **ID** (*int*) – ID is the member ID of the member to remove.

member_update(*ID*, *peerURLs*)
MemberUpdate updates the member configuration.

Parameters

- **ID** (*int*) – ID is the member ID of the member to update.
- **peerURLs** (*list of str*) – peerURLs is the new list of URLs the member will use to communicate with the cluster.

etcd3.apis.kv

class etcd3.apis.kv.**KVAPI**
Bases: *etcd3.apis.base.BaseAPI*

compact(*revision*, *physical=False*)
Compact compacts the event history in the etcd key-value store. The key-value store should be periodically compacted or the event history will continue to grow indefinitely.

Parameters

- **revision** (*int*) – revision is the key-value store revision for the compaction operation.
- **physical** (*bool*) – physical is set so the RPC will wait until the compaction is physically applied to the local database such that compacted entries are totally removed from the backend database. [default: False]

delete_range(*key=None*, *range_end=None*, *prev_kv=False*, *prefix=False*, *all=False*, *txn_obj=False*)
DeleteRange deletes the given range from the key-value store. A delete request increments the revision of the key-value store and generates a delete event in the event history for every deleted key.

Parameters

- **key** (*str or bytes*) – key is the first key to delete in the range.
- **range_end** (*str or bytes*) – range_end is the key following the last key to delete for the range [key, range_end). If range_end is not given, the range is defined to contain only the key argument. If range_end is one bit larger than the given key, then the range is all the keys with the prefix (the given key). If range_end is ‘’, the range is all keys greater than or equal to the key argument.
- **prev_kv** (*bool*) – If prev_kv is set, etcd gets the previous key-value pairs before deleting it. The previous key-value pairs will be returned in the delete response.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]
- **txn_obj** (*bool*) – return dict for the txn instead of call the api

put(*key*, *value*, *lease=0*, *prev_kv=False*, *ignore_value=False*, *ignore_lease=False*, *txn_obj=False*)
Put puts the given key into the key-value store. A put request increments the revision of the key-value store and generates one event in the event history.

Parameters

- **key** (*str or bytes*) – key is the key, in bytes, to put into the key-value store.
- **value** (*str*) – value is the value, in bytes, to associate with the key in the key-value store.
- **lease** (*int*) – lease is the lease ID to associate with the key in the key-value store. A lease value of 0 indicates no lease.
- **prev_kv** (*bool*) – If prev_kv is set, etcd gets the previous key-value pair before changing it. The previous key-value pair will be returned in the put response.
- **ignore_value** (*bool*) – If ignore_value is set, etcd updates the key using its current value. Returns an error if the key does not exist.
- **ignore_lease** (*bool*) – If ignore_lease is set, etcd updates the key using its current lease. Returns an error if the key does not exist.
- **txn_obj** (*bool*) – return dict for the txn instead of call the api

range(*key=None*, *range_end=None*, *limit=0*, *revision=None*, *serializable=False*, *keys_only=False*, *count_only=False*, *min_mod_revision=None*, *max_mod_revision=None*, *min_create_revision=None*, *max_create_revision=None*, *sort_order=<RangeRequestSortOrder.NONE: 'NONE'>*, *sort_target=<RangeRequestSortTarget.KEY: 'KEY'>*, *prefix=False*, *all=False*, *txn_obj=False*)

Range gets the keys in the range from the key-value store.

Parameters

- **key** (*str or bytes*) – key is the first key for the range. If range_end is not given, the request only looks up key.
- **range_end** (*str or bytes*) – range_end is the upper bound on the requested range [key, range_end). If range_end is ‘’, the range is all keys >= key. If range_end is key plus one (e.g., “aa”+1 == “ab”, “aÿ”+1 == “b”), then the range request gets all keys prefixed with key. If both key and range_end are ‘’, then the range request returns all keys.
- **limit** (*int*) – limit is a limit on the number of keys returned for the request. When limit is set to 0, it is treated as no limit.
- **revision** (*int*) – revision is the point-in-time of the key-value store to use for the range. If revision is less or equal to zero, the range is over the newest key-value store. If the revision has been compacted, ErrCompacted is returned as a response.
- **sort_order** (RangeRequestSortOrder) – sort_order is the order for returned sorted results.
- **sort_target** (RangeRequestSortTarget) – sort_target is the key-value field to use for sorting.
- **serializable** (*bool*) – serializable sets the range request to use serializable member-local reads. Range requests are linearizable by default; linearizable requests have higher latency and lower throughput than serializable requests but reflect the current consensus of the cluster. For better performance, in exchange for possible stale reads, a serializable range request is served locally without needing to reach consensus with other nodes in the cluster.
- **keys_only** (*bool*) – keys_only when set returns only the keys and not the values.
- **count_only** (*bool*) – count_only when set returns only the count of the keys in the range.
- **min_mod_revision** (*int*) – min_mod_revision is the lower bound for returned key mod revisions; all keys with lesser mod revisions will be filtered away.
- **max_mod_revision** (*int*) – max_mod_revision is the upper bound for returned key mod revisions; all keys with greater mod revisions will be filtered away.
- **min_create_revision** (*int*) – min_create_revision is the lower bound for returned key create revisions; all keys with lesser create revisions will be filtered away.
- **max_create_revision** (*int*) – max_create_revision is the upper bound for returned key create revisions; all keys with greater create revisions will be filtered away.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]
- **txn_obj** (*bool*) – return dict for the txn instead of call the api

txn(*compare*, *success*, *failure*)

Txn processes multiple requests in a single transaction. A txn request increments the revision of the key-

value store and generates events with the same revision for every completed request. It is not allowed to modify the same key several times within one txn.

Parameters

- **compare** (*list of dict*) – compare is a list of predicates representing a conjunction of terms. If the comparisons succeed, then the success requests will be processed in order, and the response will contain their respective responses in order. If the comparisons fail, then the failure requests will be processed in order, and the response will contain their respective responses in order.
- **success** (*list of dict*) – success is a list of requests which will be applied when compare evaluates to true.
- **failure** (*list of dict*) – failure is a list of requests which will be applied when compare evaluates to false.

etcd3.apis.lease

class etcd3.apis.lease.**LeaseAPI**

Bases: *etcd3.apis.base.BaseAPI*

lease_revoke(*ID*)

LeaseRevoke revokes a lease. All keys attached to the lease will expire and be deleted.

Parameters **ID** (*int*) – ID is the lease ID to revoke. When the ID is revoked, all associated keys will be deleted.

lease_time_to_live(*ID*, *keys=False*)

LeaseTimeToLive retrieves lease information.

Parameters

- **ID** (*int*) – ID is the lease ID for the lease.
- **keys** (*bool*) – keys is true to query all the keys attached to this lease.

lease_grant(*TTL*, *ID=0*)

LeaseGrant creates a lease which expires if the server does not receive a keepAlive within a given time to live period. All keys attached to the lease will be expired and deleted if the lease expires. Each expired key generates a delete event in the event history.

Parameters

- **TTL** (*int*) – TTL is the advisory time-to-live in seconds. the minimum value is 2s
- **ID** (*int*) – ID is the requested ID for the lease. If ID is set to 0, the lessor chooses an ID.

lease_keep_alive(*data*)

PLEASE USE THE TRANSACTION UTIL

LeaseKeepAlive keeps the lease alive by streaming keep alive requests from the client to the server and streaming keep alive responses from the server to the client.

Parameters **data** – ID stream inputs of the lease to keep alive. which not works for now

lease_keep_alive_once(*ID*)

this api only send keep alive once instead of streaming send multiple IDs

LeaseKeepAlive keeps the lease alive by streaming keep alive requests from the client to the server and streaming keep alive responses from the server to the client.

Parameters **ID** (*int*) – ID is the lease ID for the lease to keep alive.

etcd3.apis.lock

class etcd3.apis.lock.**LockAPI**
Bases: *etcd3.apis.base.BaseAPI*

lock(*name*, *lease=0*)
Lock acquires a distributed shared lock on a given named lock. On success, it will return a unique key that exists so long as the lock is held by the caller. This key can be used in conjunction with transactions to safely ensure updates to etcd only occur while holding lock ownership. The lock is held until Unlock is called on the key or the lease associate with the owner expires.

Parameters

- **name** (*str*) – name is the identifier for the distributed shared lock to be acquired.
- **lease** (*int*) – lease is the lease ID to associate with the key in the key-value store. A lease value of 0 indicates no lease.

unlock(*key*)
Unlock takes a key returned by Lock and releases the hold on lock. The next Lock caller waiting for the lock will then be woken up and given ownership of the lock.

Parameters

- **key** (*str or bytes*) – key is the lock ownership key granted by Lock.
- **lease** (*int*) – lease is the lease ID to associate with the key in the key-value store. A lease value of 0 indicates no lease.

etcd3.apis.maintenance

class etcd3.apis.maintenance.**MaintenanceAPI**
Bases: *etcd3.apis.base.BaseAPI*

alarm(*memberID*, *action=<AlarmRequestAlarmAction.GET: 'GET'>*, *alarm=<etcdserverpbAlarmType.NONE: 'NONE'>*)
Alarm activates, deactivates, and queries alarms regarding cluster health.

Parameters

- **action** (AlarmRequestAlarmAction) – action is the kind of alarm request to issue. The action may GET alarm statuses, ACTIVATE an alarm, or DEACTIVATE a raised alarm.
- **memberID** (*int*) – memberID is the ID of the member associated with the alarm. If memberID is 0, the alarm request covers all members.
- **alarm** (etcdserverpbAlarmType) – alarm is the type of alarm to consider for this request.

alarm_get(*memberID*, *alarm*)
Queries alarms regarding cluster health.

Parameters

- **memberID** (*int*) – memberID is the ID of the member associated with the alarm. If memberID is 0, the alarm request covers all members.
- **alarm** (etcdserverpbAlarmType) – alarm is the type of alarm to consider for this request.

alarm_activate(*memberID*, *alarm*)
Activates alarms regarding cluster health.

Parameters

- **memberID** (*int*) – memberID is the ID of the member associated with the alarm. If memberID is 0, the alarm request covers all members.
- **alarm** (etcdserverpbAlarmType) – alarm is the type of alarm to consider for this request.

alarm_deactivate(*memberID*, *alarm*)
Deactivates alarms regarding cluster health.

Parameters

- **memberID** (*int*) – memberID is the ID of the member associated with the alarm. If memberID is 0, the alarm request covers all members.
- **alarm** (etcdserverpbAlarmType) – alarm is the type of alarm to consider for this request.

defragment()
Defragment defragments a member's backend database to recover storage space.

hash()
Hash returns the hash of the local KV state for consistency checking purpose. This is designed for testing; do not use this in production when there are ongoing transactions.

snapshot()
Snapshot sends a snapshot of the entire backend from a member over a stream to a client.

status()
Status gets the status of the member.

etcd3.apis.watch

class etcd3.apis.watch.**WatchAPI**
Bases: *etcd3.apis.base.BaseAPI*

watch(*create_request=None*, *cancel_request=None*, ***kwargs*)
PLEASE USE THE WATCH UTIL

Watch watches for events happening or that have happened. Both input and output are streams; the input stream is for creating and canceling watchers and the output stream sends events. One watch RPC can watch on multiple key ranges, streaming events for several watches at once. The entire event history can be watched starting from the last compaction revision.

Parameters

- **create_request** (*dict*) – None
- **cancel_request** (*dict*) – None

watch_create(*key=None*, *range_end=None*, *start_revision=None*, *progress_notify=None*, *prev_kv=None*, *prefix=False*, *all=False*, *no_put=False*, *no_delete=False*, ***kwargs*)
WatchCreate creates a watch stream on given key or key_range

Parameters

- **key** (*str or bytes*) – key is the key to register for watching.

- **range_end** (*str or bytes*) – range_end is the end of the range [key, range_end) to watch. If range_end is not given, only the key argument is watched. If range_end is equal to '', all keys greater than or equal to the key argument are watched. If the range_end is one bit larger than the given key, then all keys with the prefix (the given key) will be watched.
- **start_revision** (*int*) – start_revision is an optional revision to watch from (inclusive). No start_revision is "now".
- **progress_notify** (*bool*) – progress_notify is set so that the etcd server will periodically send a WatchResponse with no events to the new watcher if there are no recent events. It is useful when clients wish to recover a disconnected watcher starting from a recent known revision. The etcd server may decide how often it will send notifications based on current load.
- **prev_kv** (*bool*) – If prev_kv is set, created watcher gets the previous KV before the event happens. If the previous KV is already compacted, nothing will be returned.
- **prefix** (*bool*) – if the key is a prefix [default: False]
- **all** (*bool*) – all the keys [default: False]
- **no_put** (*bool*) – filter out the put events at server side before it sends back to the watcher. [default: False]
- **no_delete** (*bool*) – filter out the delete events at server side before it sends back to the watcher. [default: False]

watch_cancel(*watch_id*, ***kwargs*)

NOT SUPPORTED UNDER ETCD 3.3-

https://github.com/coreos/etcd/pull/9065

WatchCancel cancels a watch stream

Parameters **watch_id** (*int*) – watch_id is the watcher id to cancel so that no more events are transmitted.

etcd3.apis.extra

class etcd3.apis.extra.**EtcdVersion**(*etcdserver*, *etcdcluster*)

Bases: tuple

etcdcluster

Alias for field number 1

etcdserver

Alias for field number 0

class etcd3.apis.extra.**ExtraAPI**

Bases: *etcd3.apis.base.BaseAPI*

version()

get the version of etcdserver and etcdcluster

Returns EtcdVersion

health()

get the health of etcd-server

Returns EtcdVersion

metrics_raw()
get the raw /metrics text

Returns str

metrics()
get the modelized metrics parsed by prometheus_client

6.1.6 etcd3.errors

etcd3.errors.errors

exception etcd3.errors.errors.**Etcd3StreamError**(*error*, *buf*, *resp*)
Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error*, *buf*, *resp*)
Initialize self. See help(type(self)) for accurate signature.

exception etcd3.errors.errors.**Etcd3WatchCanceled**(*error*, *resp*)
Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error*, *resp*)
Initialize self. See help(type(self)) for accurate signature.

etcd3.errors.errors.**get_client_error**(*error*, *code*, *status*, *response=None*)

exception etcd3.errors.errors.**UnsupportedServerVersion**
Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

etcd3.errors.go_etcd_rpctypes_error

from github.com/coreos/etcd/etcdserver/api/v3rpc/rpctypes/error.go

etcd3.errors.go_etcd_rpctypes_error.**error_desc**(*e*)

exception etcd3.errors.go_etcd_rpctypes_error.**Etcd3Exception**
Bases: Exception

etcd3.errors.go_etcd_rpctypes_error.**Error**(*err*, *name*)

exception etcd3.errors.go_etcd_rpctypes_error.**ErrUnknownError**(*error*, *code*, *status*, *response=None*)
Bases: *etcd3.errors.go_etcd_rpctypes_error.ErrUnknownError*

__init__(*error*, *code*, *status*, *response=None*)
Initialize self. See help(type(self)) for accurate signature.

exception etcd3.errors.go_etcd_rpctypes_error.**ErrEmptyKey**(*error=None*, *code=None*, *status=None*, *response=None*)
Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrKeyNotFound**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrValueProvided**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrLeaseProvided**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrTooManyOps**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrDuplicateKey**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrCompacted**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrFutureRev**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrNoSpace**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrLeaseNotFound**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrLeaseExist**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrLeaseTTLTooLarge**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrMemberExist**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrPeerURLExist**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrMemberNotEnoughStarted**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrMemberBadURLs**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrMemberNotFound**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrRequestTooLarge**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrTooManyRequests**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrRootUserNotExist**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrRootRoleNotExist**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrUserAlreadyExist**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrUserEmpty**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrUserNotFound**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrRoleAlreadyExist**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrRoleNotFound**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrAuthFailed**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrPermissionDenied**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrRoleNotGranted**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrPermissionNotGranted**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrAuthNotEnabled**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrInvalidAuthToken**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrInvalidAuthMgmt**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrNoLeader**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrNotLeader**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrNotCapable**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrStopped**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrTimeout**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrTimeoutDueToLeaderFail**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrTimeoutDueToConnectionLost**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrUnhealthy**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

exception etcd3.errors.go_etcd_rpctypes_error.**ErrCorrupt**(*error=None*, *code=None*, *status=None*, *response=None*)

Bases: *etcd3.errors.go_etcd_rpctypes_error.Etcd3Exception*

__init__(*error=None*, *code=None*, *status=None*, *response=None*)

as_dict()

etcd3.errors.go_grpc_codes

from golang grpc lib: google.golang.org/grpc/codes

class etcd3.errors.go_grpc_codes.**GRPCCode**

Bases: object

OK = 0

Canceled = 1

Unknown = 2

InvalidArgument = 3

DeadlineExceeded = 4

NotFound = 5

AlreadyExists = 6

PermissionDenied = 7

ResourceExhausted = 8

FailedPrecondition = 9

Aborted = 10

OutOfRange = 11

Unimplemented = 12

Internal = 13

Unavailable = 14

DataLoss = 15

Unauthenticated = 16

6.1.7 etcd3.models

class etcd3.models.**EtcdModel**
Bases: object

class etcd3.models.**AlarmRequestAlarmAction**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/AlarmRequestAlarmAction

default: GET

ACTIVATE = 'ACTIVATE'

DEACTIVATE = 'DEACTIVATE'

GET = 'GET'

class etcd3.models.**CompareCompareResult**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/CompareCompareResult

default: EQUAL

EQUAL = 'EQUAL'

GREATER = 'GREATER'

LESS = 'LESS'

NOT_EQUAL = 'NOT_EQUAL'

class etcd3.models.**CompareCompareTarget**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/CompareCompareTarget

default: VERSION

CREATE = 'CREATE'

LEASE = 'LEASE'

MOD = 'MOD'

VALUE = 'VALUE'

VERSION = 'VERSION'

class etcd3.models.**EventEventType**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/EventEventType

default: PUT

DELETE = 'DELETE'

PUT = 'PUT'

class etcd3.models.**RangeRequestSortOrder**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/RangeRequestSortOrder

default: NONE

ASCEND = 'ASCEND'

DESCEND = 'DESCEND'

NONE = 'NONE'

class etcd3.models.**RangeRequestSortTarget**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/RangeRequestSortTarget

default: KEY

CREATE = 'CREATE'

KEY = 'KEY'

MOD = 'MOD'

VALUE = 'VALUE'

VERSION = 'VERSION'

class etcd3.models.**WatchCreateRequestFilterType**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/WatchCreateRequestFilterType

default: NOPUT

NODELETE = 'NODELETE'

NOPUT = 'NOPUT'

class etcd3.models.**authpbPermissionType**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/authpbPermissionType

default: READ

READ = 'READ'

READWRITE = 'READWRITE'

WRITE = 'WRITE'

class etcd3.models.**etcdserverpbAlarmType**
Bases: *etcd3.models.EtcdModel*, enum.Enum

ref: #/definitions/etcdserverpbAlarmType

default: NONE

CORRUPT = 'CORRUPT'

NONE = 'NONE'

NOSPACE = 'NOSPACE'

6.1.8 etcd3.swagger_helper

etcd3.swagger_helper.**swagger_escape**(*s*)
/ and ~ are special characters in JSON Pointers, and need to be escaped when used literally (for example, in path names).

https://swagger.io/docs/specification/using-ref/#escape

class etcd3.swagger_helper.**SwaggerSpec**(*spec*)
Bases: object

Parse the swagger spec of gRPC-JSON-Gateway to object tree

__init__(*spec*)

Parameters **spec** – dict or json string or yaml string

ref(*ref_path*)
get the node object from absolute reference

Parameters **ref_path** – str

Returns SwaggerNode

example:

```
>>> spec.ref('#/definitions/etcdserverpbAlarmResponse')
SwaggerSchema(ref='#/definitions/etcdserverpbAlarmResponse')
```

get(*key*, **args*, ***kwargs*)
equivariant to self.spec.get(key)

getPath(*key*)
get a SwaggerPath instance of the path

Parameters **key** (*SwaggerNode or str*) – receive a SwaggerNode or a $ref string of schema

Return type *SwaggerNode*

getSchema(*key*)
get a SwaggerSchema instance of the schema

Parameters **key** (*SwaggerNode or str*) – receive a SwaggerNode or a $ref string of schema

Return type *SwaggerNode*

getEnum(*key*)
get a Enum instance of the schema

Parameters **key** (SwaggerNode *or* *str*) – receive a SwaggerNode or a $ref string of schema

Return type *SwaggerNode*

class etcd3.swagger_helper.**SwaggerNode**(*root*, *node*, *path*, *parent=None*, *name=None*)
Bases: object

the node of swagger_spec object

can represent a path, a schema or a ordinary node

as a schema, it can generate a model object of the definition, decode or encode the payload

__init__(*root*, *node*, *path*, *parent=None*, *name=None*)
Initialize self. See help(type(self)) for accurate signature.

encode(*data*)
encode the data to as the schema defined

Parameters **data** – data to encode

Returns encoded data

decode(*data*)
decode the data as the schema defined

Parameters **data** – data to decode

Returns decoded data

getModel()
get the model of the schema

Null handling: Since etcd's swagger spec is converted from gogoproto files, modelizing will follow the gogoprotobuf deserialization rule:

int -> 0 bool -> False object -> None array -> None string -> ""

6.1.9 etcd3.utils

etcd3.utils.**lru_cache**(*maxsize=100*)
Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: http://en.wikipedia.org/wiki/Cache_algorithms#Least_Recently_Used

etcd3.utils.**memoize**(*fn*)
Decorator. Caches a function's return value each time it is called. If called later with the same arguments, the cached value is returned (not reevaluated).

etcd3.utils.**memoize_in_object**(*fn*)
Decorator. Caches a method's return value each time it is called, in the object instance If called later with the same arguments, the cached value is returned (not reevaluated).

etcd3.utils.**incr_last_byte**(*data*)
Get the last byte in the array and increment it

etcd3.utils.**merge_two_dicts**(*x*, *y*)

etcd3.utils.**check_param**(*at_least_one_of=None*, *at_most_one_of=None*)
check if at least/most one of params is given

```
>>> @check_param(at_least_one_of=['a', 'b'])
>>> def fn(a=None, b=None):
...     pass
>>> fn()
TypeError: fn() requires at least one argument of a,b
```

etcd3.utils.**run_coro**(*coro*)

Parameters **coro** (*asyncio.coroutine*) – the coroutine to run

class etcd3.utils.**cached_property**(*func*)
Bases: object

A property that is only computed once per instance and then replaces itself with an ordinary attribute. Deleting the attribute resets the property.

Source: https://github.com/bottlepy/bottle/blob/0.11.5/bottle.py#L175

__init__(*func*)
Initialize self. See help(type(self)) for accurate signature.

etcd3.utils.**iter_json_string**(*chunk*, *start=0*, *lb=123*, *rb=125*, *resp=None*, *err_cls=<class 'ValueError'>*)

etcd3.utils.**enum_value**(*e*)

etcd3.utils.**retry**(*func*, *max_tries=3*, *log=<module 'logging' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/logging/__init__.py'>*, *err_cls=<class 'Exception'>*)

etcd3.utils.**exec_cmd**(*cmd*, *envs=None*, *raise_error=True*)

exception etcd3.utils.**Etcd3Warning**
Bases: UserWarning

etcd3.utils.**find_executable**(*executable*, *path=None*)
Find if 'executable' can be run. Looks for it in 'path' (string that lists directories separated by 'os.pathsep'; defaults to os.environ['PATH']). Checks for all executable extensions. Returns full path or None if no command is found.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

Symbols

A

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W