# InfluxDB Documentation

***Release 4.1.1***

**John Shahid**

**Aug 24, 2017**

# Contents

**Release** 4.1.1

**Date** Aug 24, 2017

**Keywords** python, time series, database

CHAPTER 1

Contents

# InfluxDB-Python

InfluxDB-Python is a client for interacting with InfluxDB. Maintained by @aviau (https://github.com/aviau).

**Help needed:** Development of this library is made by the community and help is needed. A co-maintainer would be welcome. To contribute, take a look at the issues list of simply contact @aviau. InfluxDB is an open-source distributed time series database, find more about InfluxDB at http://influxdata.com/

## InfluxDB pre v1.1.0 users

InfluxDB 1.1.0 was released and it is the new recommended version. InfluxDB 0.8.x users may still use the legacy client by using `from influxdb.influxdb08 import InfluxDBClient` instead.

## Installation

Install, upgrade and uninstall InfluxDB-Python with these commands:

```
$ pip install influxdb
$ pip install --upgrade influxdb
$ pip uninstall influxdb
```

On Debian/Ubuntu, you can install it with this command:

```
$ sudo apt-get install python-influxdb
```

## Dependencies

The InfluxDB-Python distribution is supported and tested on Python 2.7, 3.3, 3.4, 3.5, 3.6, PyPy and PyPy3.

**Note:** Python 3.2 is currently untested. See `.travis.yml`.

Main dependency is:

- Requests: HTTP library for human beings (http://docs.python-requests.org/)

Additional dependencies are:

- pandas: for writing from and reading to DataFrames (http://pandas.pydata.org/)
- Sphinx: Tool to create and manage the documentation (http://sphinx-doc.org/)
- Nose: to auto-discover tests (http://nose.readthedocs.org/en/latest/)
- Mock: to mock tests (https://pypi.python.org/pypi/mock)

## Documentation

InfluxDB-Python documentation is available at http://influxdb-python.readthedocs.org

You will need Sphinx installed to generate the documentation.

The documentation can be generated by running:

```
$ tox -e docs
```

Generated documentation can be found in the *docs/build/html/* directory.

## Examples

Here's a basic example (for more see the examples directory):

```
$ python

>>> from influxdb import InfluxDBClient

>>> json_body = [
    {
        "measurement": "cpu_load_short",
        "tags": {
            "host": "server01",
            "region": "us-west"
        },
        "time": "2009-11-10T23:00:00Z",
        "fields": {
            "value": 0.64
        }
    }
]

>>> client = InfluxDBClient('localhost', 8086, 'root', 'root', 'example')

>>> client.create_database('example')

>>> client.write_points(json_body)

>>> result = client.query('select value from cpu_load_short;')

>>> print("Result: {0}".format(result))
```

### Testing

Make sure you have tox by running the following:

```
$ pip install tox
```

To test influxdb-python with multiple version of Python, you can use Tox:

```
$ tox
```

### Support

For issues with, questions about, or feedback for InfluxDB, please look into our community page: http://influxdb.com/community/.

### Development

All development is done on Github. Use Issues to report problems or submit contributions.

### TODO

The TODO/Roadmap can be found in Github bug tracker: https://github.com/influxdata/influxdb-python/issues

### Source code

The source code is currently available on Github: https://github.com/influxdata/influxdb-python

## API Documentation

To connect to a InfluxDB, you must create a *InfluxDBClient* object. The default configuration connects to InfluxDB on `localhost` with the default ports. The below instantiation statements are all equivalent:

```
from influxdb import InfluxDBClient

# using Http
client = InfluxDBClient(database='dbname')
client = InfluxDBClient(host='127.0.0.1', port=8086, database='dbname')
client = InfluxDBClient(host='127.0.0.1', port=8086, username='root', password='root',
↪ database='dbname')

# using UDP
client = InfluxDBClient(host='127.0.0.1', database='dbname', use_udp=True, udp_
↪port=4444)
```

To write pandas DataFrames or to read data into a pandas DataFrame, use a *DataFrameClient* object. These clients are initiated in the same way as the *InfluxDBClient*:

```
from influxdb import DataFrameClient

client = DataFrameClient(host='127.0.0.1', port=8086, username='root', password='root
↪', database='dbname')
```

**Note:** Only when using UDP (use_udp=True) the connections is established.

## InfluxDBClient

**class** influxdb.**InfluxDBClient**(*host=u'localhost'*, *port=8086*, *username=u'root'*, *password=u'root'*, *database=None*, *ssl=False*, *verify_ssl=False*, *timeout=None*, *retries=3*, *use_udp=False*, *udp_port=4444*, *proxies=None*)

InfluxDBClient primary client object to connect InfluxDB.

The *InfluxDBClient* object holds information necessary to connect to InfluxDB. Requests can be made to InfluxDB directly through the client.

**Parameters**

- **host** (*str*) – hostname to connect to InfluxDB, defaults to 'localhost'
- **port** (*int*) – port to connect to InfluxDB, defaults to 8086
- **username** (*str*) – user to connect, defaults to 'root'
- **password** (*str*) – password of the user, defaults to 'root'
- **database** (*str*) – database name to connect to, defaults to None
- **ssl** (*bool*) – use https instead of http to connect to InfluxDB, defaults to False
- **verify_ssl** (*bool*) – verify SSL certificates for HTTPS requests, defaults to False
- **timeout** (*int*) – number of seconds Requests will wait for your client to establish a connection, defaults to None
- **retries** (*int*) – number of retries your client will try before aborting, defaults to 3. 0 indicates try until success
- **use_udp** (*bool*) – use UDP to connect to InfluxDB, defaults to False
- **udp_port** (*int*) – UDP port to connect to InfluxDB, defaults to 4444
- **proxies** (*dict*) – HTTP(S) proxy to use for Requests, defaults to {}

**alter_retention_policy**(*name*, *database=None*, *duration=None*, *replication=None*, *default=None*)

Mofidy an existing retention policy for a database.

**Parameters**

- **name** (*str*) – the name of the retention policy to modify
- **database** (*str*) – the database for which the retention policy is modified. Defaults to current client's database
- **duration** (*str*) – the new duration of the existing retention policy. Durations such as 1h, 90m, 12h, 7d, and 4w, are all supported and mean 1 hour, 90 minutes, 12 hours, 7 day, and 4 weeks, respectively. For infinite retention, meaning the data will never be deleted, use 'INF' for duration. The minimum retention period is 1 hour.
- **replication** (*int*) – the new replication of the existing retention policy
- **default** (*bool*) – whether or not to set the modified policy as default

**Note:** at least one of duration, replication, or default flag should be set. Otherwise the operation will fail.

**close**()
Close http session.

**create_database**(*dbname*)
Create a new database in InfluxDB.

**Parameters** **dbname** (*str*) – the name of the database to create

**create_retention_policy**(*name*, *duration*, *replication*, *database=None*, *default=False*)
Create a retention policy for a database.

**Parameters**

- **name** (*str*) – the name of the new retention policy
- **duration** (*str*) – the duration of the new retention policy. Durations such as 1h, 90m, 12h, 7d, and 4w, are all supported and mean 1 hour, 90 minutes, 12 hours, 7 day, and 4 weeks, respectively. For infinite retention - meaning the data will never be deleted - use 'INF' for duration. The minimum retention period is 1 hour.
- **replication** (*str*) – the replication of the retention policy
- **database** (*str*) – the database for which the retention policy is created. Defaults to current client's database
- **default** (*bool*) – whether or not to set the policy as default

**create_user**(*username*, *password*, *admin=False*)
Create a new user in InfluxDB.

**Parameters**

- **username** (*str*) – the new username to create
- **password** (*str*) – the password for the new user
- **admin** (*boolean*) – whether the user should have cluster administration privileges or not

**delete_series**(*database=None*, *measurement=None*, *tags=None*)
Delete series from a database.

Series can be filtered by measurement and tags.

**Parameters**

- **database** (*str*) – the database from which the series should be deleted, defaults to client's current database
- **measurement** (*str*) – Delete all series from a measurement
- **tags** (*dict*) – Delete all series that match given tags

**drop_database**(*dbname*)
Drop a database from InfluxDB.

**Parameters** **dbname** (*str*) – the name of the database to drop

**drop_retention_policy**(*name*, *database=None*)
Drop an existing retention policy for a database.

**Parameters**

- **name** (*str*) – the name of the retention policy to drop
- **database** (*str*) – the database for which the retention policy is dropped. Defaults to current client's database

**drop_user**(*username*)
Drop a user from InfluxDB.

**Parameters** **username** (*str*) – the username to drop

**classmethod from_dsn**(*dsn*, ***kwargs*)
Generate an instance of InfluxDBClient from given data source name.

Return an instance of *InfluxDBClient* from the provided data source name. Supported schemes are "influxdb", "https+influxdb" and "udp+influxdb". Parameters for the *InfluxDBClient* constructor may also be passed to this method.

**Parameters**

- **dsn** (*string*) – data source name
- **kwargs** (*dict*) – additional parameters for *InfluxDBClient*

**Raises** **ValueError** – if the provided DSN has any unexpected values

**Example**

```
>> cli = InfluxDBClient.from_dsn('influxdb://username:password@\
localhost:8086/databasename', timeout=5)
>> type(cli)
<class 'influxdb.client.InfluxDBClient'>
>> cli = InfluxDBClient.from_dsn('udp+influxdb://username:pass@\
localhost:8086/databasename', timeout=5, udp_port=159)
>> print('{0._baseurl} - {0.use_udp} {0.udp_port}'.format(cli))
http://localhost:8086 - True 159
```

**Note:** parameters provided in ***kwargs* may override dsn parameters

**Note:** when using "udp+influxdb" the specified port (if any) will be used for the TCP connection; specify the UDP port with the additional *udp_port* parameter (cf. examples).

**get_list_database**()
Get the list of databases in InfluxDB.

**Returns** all databases in InfluxDB

**Return type** list of dictionaries

**Example**

```
>> dbs = client.get_list_database()
>> dbs
[{u'name': u'db1'}, {u'name': u'db2'}, {u'name': u'db3'}]
```

**get_list_privileges**(*username*)
Get the list of all privileges granted to given user.

**Parameters** **username** (*str*) – the username to get privileges of

**Returns** all privileges granted to given user

**Return type** list of dictionaries

**Example**

```
>> privileges = client.get_list_privileges('user1')
>> privileges
[{u'privilege': u'WRITE', u'database': u'db1'},
 {u'privilege': u'ALL PRIVILEGES', u'database': u'db2'},
 {u'privilege': u'NO PRIVILEGES', u'database': u'db3'}]
```

**get_list_retention_policies**(*database=None*)
Get the list of retention policies for a database.

**Parameters** **database** (*str*) – the name of the database, defaults to the client's current database

**Returns** all retention policies for the database

**Return type** list of dictionaries

**Example**

```
>> ret_policies = client.get_list_retention_policies('my_db')
>> ret_policies
[{u'default': True,
  u'duration': u'0',
  u'name': u'default',
  u'replicaN': 1}]
```

**get_list_users**()
Get the list of all users in InfluxDB.

**Returns** all users in InfluxDB

**Return type** list of dictionaries

**Example**

```
>> users = client.get_list_users()
>> users
[{u'admin': True, u'user': u'user1'},
 {u'admin': False, u'user': u'user2'},
 {u'admin': False, u'user': u'user3'}]
```

**grant_admin_privileges**(*username*)
Grant cluster administration privileges to a user.

**Parameters** **username** (*str*) – the username to grant privileges to

**Note:** Only a cluster administrator can create/drop databases and manage users.

**grant_privilege**(*privilege*, *database*, *username*)
Grant a privilege on a database to a user.

**Parameters**

- **privilege** (*str*) – the privilege to grant, one of 'read', 'write' or 'all'. The string is case-insensitive
- **database** (*str*) – the database to grant the privilege on

- **username** (*str*) – the username to grant the privilege to

**query**(*query*, *params=None*, *epoch=None*, *expected_response_code=200*, *database=None*, *raise_errors=True*, *chunked=False*, *chunk_size=0*)
Send a query to InfluxDB.

**Parameters**

- **query** (*str*) – the actual query string
- **params** (*dict*) – additional parameters for the request, defaults to {}
- **epoch** (*str*) – response timestamps to be in epoch format either 'h', 'm', 's', 'ms', 'u', or 'ns',defaults to *None* which is RFC3339 UTC format with nanosecond precision
- **expected_response_code** (*int*) – the expected status code of response, defaults to 200
- **database** (*str*) – database to query, defaults to None
- **raise_errors** (*bool*) – Whether or not to raise exceptions when InfluxDB returns errors, defaults to True
- **chunked** (*bool*) – Enable to use chunked responses from InfluxDB. With `chunked` enabled, one ResultSet is returned per chunk containing all results within that chunk
- **chunk_size** (*int*) – Size of each chunk to tell InfluxDB to use.

**Returns** the queried data

**Return type** *ResultSet*

**request**(*url*, *method=u'GET'*, *params=None*, *data=None*, *expected_response_code=200*, *headers=None*)
Make a HTTP request to the InfluxDB API.

**Parameters**

- **url** (*str*) – the path of the HTTP request, e.g. write, query, etc.
- **method** (*str*) – the HTTP method for the request, defaults to GET
- **params** (*dict*) – additional parameters for the request, defaults to None
- **data** (*str*) – the data of the request, defaults to None
- **expected_response_code** (*int*) – the expected response code of the request, defaults to 200
- **headers** (*dict*) – headers to add to the request

**Returns** the response from the request

**Return type** `requests.Response`

**Raises**

- ***InfluxDBServerError*** – if the response code is any server error code (5xx)
- ***InfluxDBClientError*** – if the response code is not the same as *expected_response_code* and is not a server error code

**revoke_admin_privileges**(*username*)
Revoke cluster administration privileges from a user.

**Parameters** **username** (*str*) – the username to revoke privileges from

**Note:** Only a cluster administrator can create/ drop databases and manage users.

**revoke_privilege**(*privilege*, *database*, *username*)
Revoke a privilege on a database from a user.

**Parameters**

- **privilege** (*str*) – the privilege to revoke, one of 'read', 'write' or 'all'. The string is case-insensitive
- **database** (*str*) – the database to revoke the privilege on
- **username** (*str*) – the username to revoke the privilege from

**send_packet**(*packet*, *protocol=u'json'*)
Send an UDP packet.

**Parameters**

- **packet** (*(if protocol is 'json') dict (if protocol is 'line') sequence of line protocol strings*) – the packet to be sent
- **protocol** (*str*) – protocol of input data, either 'json' or 'line'

**set_user_password**(*username*, *password*)
Change the password of an existing user.

**Parameters**

- **username** (*str*) – the username who's password is being changed
- **password** (*str*) – the new password for the user

**switch_database**(*database*)
Change the client's database.

**Parameters** **database** (*str*) – the name of the database to switch to

**switch_user**(*username*, *password*)
Change the client's username.

**Parameters**

- **username** (*str*) – the username to switch to
- **password** (*str*) – the password for the username

**write**(*data*, *params=None*, *expected_response_code=204*, *protocol=u'json'*)
Write data to InfluxDB.

**Parameters**

- **data** (*(if protocol is 'json') dict (if protocol is 'line') sequence of line protocol strings*) – the data to be written
- **params** (*dict*) – additional parameters for the request, defaults to None
- **expected_response_code** (*int*) – the expected response code of the write operation, defaults to 204
- **protocol** (*str*) – protocol of input data, either 'json' or 'line'

**Returns** True, if the write operation is successful

**Return type** bool

**write_points**(*points*, *time_precision=None*, *database=None*, *retention_policy=None*, *tags=None*, *batch_size=None*, *protocol=u'json'*)

Write to multiple time series names.

**Parameters**

- **points** (*list of dictionaries, each dictionary represents a point*) – the list of points to be written in the database
- **time_precision** (*str*) – Either 's', 'm', 'ms' or 'u', defaults to None
- **database** (*str*) – the database to write the points to. Defaults to the client's current database
- **tags** (*dict*) – a set of key-value pairs associated with each point. Both keys and values must be strings. These are shared tags and will be merged with point-specific tags, defaults to None
- **retention_policy** (*str*) – the retention policy for the points. Defaults to None
- **batch_size** (*int*) – value to write the points in batches instead of all at one time. Useful for when doing data dumps from one database to another or when doing a massive write operation, defaults to None
- **protocol** (*str*) – Protocol for writing data. Either 'line' or 'json'.

**Returns** True, if the operation is successful

**Return type** bool

**Note:** if no retention policy is specified, the default retention policy for the database is used

## DataFrameClient

**class** influxdb.**DataFrameClient**(**a*, ***kw*)

DataFrameClient default class instantiation.

**err = ImportError('No module named pandas',)**

## SeriesHelper

**class** influxdb.**SeriesHelper**(***kw*)

Subclass this helper eases writing data points in bulk.

All data points are immutable, ensuring they do not get overwritten. Each subclass can write to its own database. The time series names can also be based on one or more defined fields. The field "time" can be specified when creating a point, and may be any of the time types supported by the client (i.e. str, datetime, int). If the time is not specified, the current system time (utc) will be used.

Annotated example:

```
class MySeriesHelper(SeriesHelper):
    class Meta:
        # Meta class stores time series helper configuration.
        series_name = 'events.stats.{server_name}'
        # Series name must be a string, curly brackets for dynamic use.
        fields = ['time', 'server_name']
        # Defines all the fields in this time series.
```

```
        ### Following attributes are optional. ###
        client = TestSeriesHelper.client
        # Client should be an instance of InfluxDBClient.
        :warning: Only used if autocommit is True.
        bulk_size = 5
        # Defines the number of data points to write simultaneously.
        # Only applicable if autocommit is True.
        autocommit = True
        # If True and no bulk_size, then will set bulk_size to 1.
```

**classmethod commit**(*client=None*)
Commit everything from datapoints via the client.

**Parameters** **client** – InfluxDBClient instance for writing points to InfluxDB.

**Attention** any provided client will supersede the class client.

**Returns** result of client.write_points.

## ResultSet

See the *Query response object: ResultSet* page for more information.

**class** influxdb.resultset.**ResultSet**(*series*, *raise_errors=True*)
A wrapper around a single InfluxDB query result.

**error**
Error returned by InfluxDB.

**get_points**(*measurement=None*, *tags=None*)
Return a generator for all the points that match the given filters.

**Parameters**

- **measurement** (*str*) – The measurement name
- **tags** (*dict*) – Tags to look for

**Returns** Points generator

**items**()
Return the set of items from the ResultSet.

**Returns** List of tuples, (key, generator)

**keys**()
Return the list of keys in the ResultSet.

**Returns** List of keys. Keys are tuples (serie_name, tags)

**static point_from_cols_vals**(*cols*, *vals*)
Create a dict from columns and values lists.

**Parameters**

- **cols** – List of columns
- **vals** – List of values

**Returns** Dict where keys are columns.

**raw**
Raw JSON from InfluxDB.

# Exceptions

**class** influxdb.exceptions.**InfluxDBClientError**(*content*, *code=None*)
Raised when an error occurs in the request.

**class** influxdb.exceptions.**InfluxDBServerError**(*content*)
Raised when a server error occurs.

# Query response object: ResultSet

Using the `InfluxDBClient.query()` function will return a `ResultSet` Object.

A ResultSet can be browsed in several ways. Its `get_points` method can be used to retrieve points generators that filter either by measurement, tags, or both.

## Getting all points

Using `rs.get_points()` will return a generator for all the points in the ResultSet.

## Filtering by measurement

Using `rs.get_points('cpu')` will return a generator for all the points that are in a serie with measurement name `cpu`, no matter the tags.

```
rs = cli.query("SELECT * from cpu")
cpu_points = list(rs.get_points(measurement='cpu'))
```

## Filtering by tags

Using `rs.get_points(tags={'host_name': 'influxdb.com'})` will return a generator for all the points that are tagged with the specified tags, no matter the measurement name.

```
rs = cli.query("SELECT * from cpu")
cpu_influxdb_com_points = list(rs.get_points(tags={"host_name": "influxdb.com"}))
```

## Filtering by measurement and tags

Using measurement name and tags will return a generator for all the points that are in a serie with the specified measurement name AND whose tags match the given tags.

```
rs = cli.query("SELECT * from cpu")
points = list(rs.get_points(measurement='cpu', tags={'host_name': 'influxdb.com'}))
```

See the *API Documentation* page for more information.

# InfluxDB Python Examples

## Tutorials - Basic

```
# -*- coding: utf-8 -*-
"""Tutorial on using the InfluxDB client."""

import argparse

from influxdb import InfluxDBClient


def main(host='localhost', port=8086):
    """Instantiate a connection to the InfluxDB."""
    user = 'root'
    password = 'root'
    dbname = 'example'
    dbuser = 'smly'
    dbuser_password = 'my_secret_password'
    query = 'select value from cpu_load_short;'
    json_body = [
        {
            "measurement": "cpu_load_short",
            "tags": {
                "host": "server01",
                "region": "us-west"
            },
            "time": "2009-11-10T23:00:00Z",
            "fields": {
                "Float_value": 0.64,
                "Int_value": 3,
                "String_value": "Text",
                "Bool_value": True
            }
        }
    ]

    client = InfluxDBClient(host, port, user, password, dbname)

    print("Create database: " + dbname)
    client.create_database(dbname)

    print("Create a retention policy")
    client.create_retention_policy('awesome_policy', '3d', 3, default=True)

    print("Switch user: " + dbuser)
    client.switch_user(dbuser, dbuser_password)

    print("Write points: {0}".format(json_body))
    client.write_points(json_body)

    print("Querying data: " + query)
    result = client.query(query)

    print("Result: {0}".format(result))

    print("Switch user: " + user)
```

```
    client.switch_user(user, password)

    print("Drop database: " + dbname)
    client.drop_database(dbname)


def parse_args():
    """Parse the args."""
    parser = argparse.ArgumentParser(
        description='example code to play with InfluxDB')
    parser.add_argument('--host', type=str, required=False,
                        default='localhost',
                        help='hostname of InfluxDB http API')
    parser.add_argument('--port', type=int, required=False, default=8086,
                        help='port of InfluxDB http API')
    return parser.parse_args()


if __name__ == '__main__':
    args = parse_args()
    main(host=args.host, port=args.port)
```

## Tutorials - pandas

```
# -*- coding: utf-8 -*-
"""Tutorial for using pandas and the InfluxDB client."""

import argparse
import pandas as pd

from influxdb import DataFrameClient


def main(host='localhost', port=8086):
    """Instantiate the connection to the InfluxDB client."""
    user = 'root'
    password = 'root'
    dbname = 'demo'
    # Temporarily avoid line protocol time conversion issues #412, #426, #431.
    protocol = 'json'

    client = DataFrameClient(host, port, user, password, dbname)

    print("Create pandas DataFrame")
    df = pd.DataFrame(data=list(range(30)),
                      index=pd.date_range(start='2014-11-16',
                                          periods=30, freq='H'))

    print("Create database: " + dbname)
    client.create_database(dbname)

    print("Write DataFrame")
    client.write_points(df, 'demo', protocol=protocol)

    print("Write DataFrame with Tags")
    client.write_points(df, 'demo',
```

```
                        {'k1': 'v1', 'k2': 'v2'}, protocol=protocol)

    print("Read DataFrame")
    client.query("select * from demo")

    print("Delete database: " + dbname)
    client.drop_database(dbname)


def parse_args():
    """Parse the args from main."""
    parser = argparse.ArgumentParser(
        description='example code to play with InfluxDB')
    parser.add_argument('--host', type=str, required=False,
                        default='localhost',
                        help='hostname of InfluxDB http API')
    parser.add_argument('--port', type=int, required=False, default=8086,
                        help='port of InfluxDB http API')
    return parser.parse_args()


if __name__ == '__main__':
    args = parse_args()
    main(host=args.host, port=args.port)
```

## Tutorials - SeriesHelper

```
# -*- coding: utf-8 -*-
"""Tutorial how to use the class helper `SeriesHelper`."""

from influxdb import InfluxDBClient
from influxdb import SeriesHelper

# InfluxDB connections settings
host = 'localhost'
port = 8086
user = 'root'
password = 'root'
dbname = 'mydb'

myclient = InfluxDBClient(host, port, user, password, dbname)

# Uncomment the following code if the database is not yet created
# myclient.create_database(dbname)
# myclient.create_retention_policy('awesome_policy', '3d', 3, default=True)


class MySeriesHelper(SeriesHelper):
    """Instantiate SeriesHelper to write points to the backend."""

    class Meta:
        """Meta class stores time series helper configuration."""

        # The client should be an instance of InfluxDBClient.
        client = myclient
```

```
        # The series name must be a string. Add dependent fields/tags
        # in curly brackets.
        series_name = 'events.stats.{server_name}'

        # Defines all the fields in this time series.
        fields = ['some_stat', 'other_stat']

        # Defines all the tags for the series.
        tags = ['server_name']

        # Defines the number of data points to store prior to writing
        # on the wire.
        bulk_size = 5

        # autocommit must be set to True when using bulk_size
        autocommit = True


# The following will create *five* (immutable) data points.
# Since bulk_size is set to 5, upon the fifth construction call, *all* data
# points will be written on the wire via MySeriesHelper.Meta.client.
MySeriesHelper(server_name='us.east-1', some_stat=159, other_stat=10)
MySeriesHelper(server_name='us.east-1', some_stat=158, other_stat=20)
MySeriesHelper(server_name='us.east-1', some_stat=157, other_stat=30)
MySeriesHelper(server_name='us.east-1', some_stat=156, other_stat=40)
MySeriesHelper(server_name='us.east-1', some_stat=155, other_stat=50)

# To manually submit data points which are not yet written, call commit:
MySeriesHelper.commit()

# To inspect the JSON which will be written, call _json_body_():
MySeriesHelper._json_body_()
```

CHAPTER 2

# Indices and tables

- genindex
- search

# Index