Python Statsd Documentation

Release 1.0

Rick van Hattem

April 14, 2012

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

statsd is a client for Etsy's statsd server, a front end/proxy for the Graphite stats collection and graphing server.

- **Graphite**
  - http://graphite.wikidot.com
- **Statsd**
  - code: https://github.com/etsy/statsd
  - blog post: http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/

CHAPTER

TWO

INSTALL

To install simply execute *python setup.py install*. If you want to run the tests first, run *python setup.py nosetests*

CHAPTER

THREE

USAGE

To get started real quick, just try something like this:

3.1 Basic Usage

3.1.1 Timers

```
>>> import statsd
>>>
>>> timer = statsd.Timer('MyApplication')
>>>
>>> timer.start()
>>> # do something here
>>> timer.stop('SomeTimer')
```

3.1.2 Counters

```
>>> import statsd
>>>
>>> counter = statsd.Counter('MyApplication')
>>> # do something here
>>> counter += 1
```

3.2 Advanced Usage

```
>>> import statsd
>>>
>>> # Open a connection to `server` on port `1234` with a `50%` sample rate
>>> statsd_connection = statsd.Connection(
...     name='server',
...     port=1234,
...     sample_rate=0.5,
... )
>>>
>>> # Create a client for this application
>>> statsd_client = statsd.Client(__name__, statsd_connection)
>>>
```

```
>>> class SomeClass(object):
...     def __init__(self):
...         # Create a client specific for this class
...         self.statsd_client = statsd_client.get_client(
...             self.__class__.__name__)
...
...     def do_something(self):
...         # Create a `timer` client
...         timer = self.statsd_client.get_client(class_=statsd.Timer)
...
...         # start the measurement
...         timer.start()
...
...         # do something
...         timer.interval('intermediate_value')
...
...         # do something else
...         timer.stop('total')
```

CHAPTER

FOUR

STATSD MODULE REFERENCE

Contents:

4.1 statsd.connection

class `statsd.connection.`**`Connection`**(*host=None*, *port=None*, *sample_rate=None*)

Statsd Connection

Parameters

- **host** – The statsd host to connect to, defaults to *localhost*
- **port** – The statsd port to connect to, defaults to *8125*
- **sample_rate** – The sample rate, defaults to *1* (meaning always)

`send`(*data*, *sample_rate=None*)

Send the data over UDP while taking the sample_rate in account

The sample rate should be a number between *0* and *1* which indicates the probability that a message will be sent. The sample_rate is also communicated to *statsd* so it knows what multiplier to use.

4.2 statsd.client

class `statsd.client.`**`Client`**(*name*, *connection=None*)

Statsd Client Object

Parameters

- **task_id** – see `name`.
- **task_id** – see `connection`.

`connection` = **None**

The `Connection` to use, creates a new connection if no connection is given

`get_client`(*name=None*, *class_=None*)

Get a (sub-)client with a separate namespace This way you can create a global/app based client with subclients per class/function

Parameters

- **name** – The name to use, if the name for this client was *spam* and the *name* argument is *eggs* than the resulting name will be *spam.eggs*

- **class** – The `Client` subclass to use (e.g. `Timer` or `Counter`)

`name` **= None**
The name of the client, everything sent from this client will be prefixed by name

4.3 statsd.timer

class `statsd.timer.`**`Timer`**(*name*, *connection=None*)
Statsd Timer Object

Additional documentation is available at the parent class `Client`

```
>>> timer = Timer('application_name')
>>> timer.start()
>>>  # do something
>>> timer.stop('executed_action')
```

`decorate`(*function_or_name*)
Decorate a function to time the execution

The method can be called with or without a name. If no name is given the function defaults to the name of the function.

Parameters function_or_name – The name to post to or the function to wrap

```
>>> from statsd import Timer
>>> timer = Timer('application_name')
>>>
>>> @timer.decorate
... def some_function():
...     # resulting timer name: application_name.some_function
...     pass
>>>
>>> @timer.decorate('my_timer')
... def some_function():
...     # resulting timer name: application_name.my_timer
...     pass
```

`intermediate`(*subname*)
Send the time that has passed since our last measurement

Parameters subname – The subname to report the data to (appended to the client name)

`send`(*subname*, *delta*)
Send the data to statsd via self.connection

Parameters

- **subname** – The subname to report the data to (appended to the client name)
- **delta** – The time delta (time.time() - time.time()) to report

`start`()
Start the timer and store the start time, this can only be executed once per instance

`stop`(*subname='total'*)
Stop the timer and send the total since *start()* was run

Parameters subname – The subname to report the data to (appended to the client name)

4.4 statsd.counter

class `statsd.counter.`**`Counter`**(*name*, *connection=None*)

Class to implement a statd counter

Additional documentation is available at the parent class `Client`

The values can be incremented/decremented by using either the *increment()* and *decrement()* methods or by simply adding/deleting from the object.

```
>>> counter = Counter('application_name')
>>> counter += 10
```

```
>>> counter = Counter('application_name')
>>> counter -= 10
```

`decrement`(*subname=None*, *delta=1*)

Decrement the counter with *delta*

Parameters

- **subname** – The subname to report the data to (appended to the client name)
- **delta** – The delta to remove from the counter

```
>>> counter = Counter('application_name')
>>> counter.decrement('counter_name', 10)
>>> counter.decrement(delta=10)
>>> counter.decrement('counter_name')
```

`increment`(*subname=None*, *delta=1*)

Increment the counter with *delta*

Parameters

- **subname** – The subname to report the data to (appended to the client name)
- **delta** – The delta to add to the counter

```
>>> counter = Counter('application_name')
>>> counter.increment('counter_name', 10)
>>> counter.increment(delta=10)
>>> counter.increment('counter_name')
```

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *search*

PYTHON MODULE INDEX