

---

# **DynamoDB-mock Documentation**

***Release 0.3.0***

**Jean-Tiare Le Bigot**

October 11, 2012



# CONTENTS



# OVERVIEW

**DynamoDB** is a minimalistic NoSQL engine provided by Amazon as a part of their AWS product.

**DynamoDB** is great in production environment but sucks when testing your application. Tables needs roughly 1 min to be created, deleted or updated. Items operation rates depends on how much you pay and tests will conflict if 2 developers run them at the same time.

**ddbmock** brings a tiny in-memory(tm) implementation of DynamoDB API. It can either be run as a stand alone server or as a regular library helping you to build lightning fast unit and functional tests :)

**ddbmock** does *not* intend to be production ready. It *will* **loose** you data. you've been warned! I currently recommend the “boto extension” mode for unit-tests and the “server” mode for functional tests.



# DOCUMENTATION

## 2.1 User guide

### 2.1.1 Getting started with DynamoDB-mock

DynamoDB is a minimalistic NoSQL engine provided by Amazon as a part of their AWS product.

DynamoDB is great in production environment but sucks when testing your application. Tables needs roughly 1 min to be created, deleted or updated. Items operation rates depends on how much you pay and tests will conflict if 2 developers run them at the same time.

**ddbmock** brings a tiny in-memory(tm) implementation of DynamoDB API. It can either be run as a stand alone server or as a regular library helping you to build lightning fast unit and functional tests :)

**ddbmock** does *not* intend to be production ready. It *will loose* you data. you've been warned! I currently recommend the "boto extension" mode for unit-tests and the "server" mode for functional tests.

#### Installation

```
$ pip install ddbmock
```

#### Example usage

##### Run as Regular client-server

Ideal for test environment. For stage and production I highly recommend using DynamoDB servers. ddbmock comes with no warranty and *will loose your data(tm)*.

```
$ pserve development.ini # launch the server on 0.0.0.0:6543
```

```
import boto
from ddbmock import connect_ddbmock

# Use the provided helper to connect your *own* endpoint
db = connect_ddbmock()

# Done ! just use it wherever in your project as usual.
db.list_tables() # get list of tables (empty at this stage)
```

Note: if you do not want to import ddbmock only for the helper, here is a reference implementation:

```
def connect_ddbmock(host='localhost', port=6543):
    import boto
    from boto.regioninfo import RegionInfo
    endpoint = '{}:{}'.format(host, port)
    region = RegionInfo(name='ddbmock', endpoint=endpoint)
    return boto.connect_dynamodb(region=region, port=port, is_secure=False)
```

### Run as a standalone library

Ideal for unit testing or small scale automated functional tests. Nice to play around with boto DynamoDB API too :)

```
import boto
from ddbmock import connect_boto

# Wire-up boto and ddbmock together
db = connect_boto()

# Done ! just use it wherever in your project as usual.
db.list_tables() # get list of tables (empty at this stage)
```

## 2.1.2 Current Status

This documents reflects ddbmock status as of 3/10/12. It may be outdated.

### Methods support

- CreateTable DONE
- DeleteTable DONE
- UpdateTable DONE
- DescribeTable DONE
- GetItem DONE
- PutItem DONE
- DeleteItem DONE
- UpdateItem ALMOST
- BatchGetItem WIP
- BatchWriteItem TODO
- Query WIP
- Scan WIP

There is basically no support for Limit, ExclusiveStartKey, ScanIndexForward and their associated features at all in ddbmock. This affects all “WIP” functions.

UpdateItem has a different behavior when the target item did not exist prior the update operation. In particular, the ADD operator will always behave as though the item existed before.



## Comparison Operators

Some comparison might not work as expected on binary data as it is performed on the base64 representation instead of the binary one. Please report a bug if this is a problem for you, or, even better, open a pull request :)

All operators exists as lower case functions in `ddbmock.database.comparison`. This list can easily be extended to add new/custom operators.

### Common to Query and Scan

- `EQ DONE`
- `LE DONE`
- `LT DONE`
- `GE DONE`
- `GT DONE`
- `BEGINS_WITH DONE`
- `BETWEEN DONE`

### Specific to Scan

- `NULL DONE`
- `NOT_NULL DONE`
- `CONTAINS DONE`
- `NOT_CONTAINS DONE`
- `IN DONE`

`IN` operator is the only that can not be imported directly as it overlaps builtin `in` keyword. If you need it, either import it with `getattr` on the module or as `in_test` which, anyway, is its internal name.

## Rates and size limitations

basically, none are supported yet

### Request rate

- Throttle read operations when provisioned throughput exceeded. TODO
- Throttle write operations when provisioned throughput exceeded. TODO
- Maximum throughput is 10,000. DONE
- Minimum throughput is 1. DONE
- Report accurate throughput. WONT FIX

ddbmock currently reports the consumed throughput based on item count. Their size is ignored from the computation. While it is theoretically possible, it would no be accurate anyway because we can not reproduce exactly Amazon's storage efficiency.

Actually, this is even trickier as the throughput is normally spreaded among partitions which ddbmock does not handle.

### Request size

- Limit response size to 1MB. TODO
- Limit request size to 1MB. TODO
- Limit `BatchGetItem` to 100 per request. TODO
- Limit `BatchWriteItem` to 25 per request. TODO

### Table management

- No more than 255 tables. TODO
- No more than 10 `CREATING` tables. TODO
- No more than 10 `DELETING` tables. TODO
- No more than 1 `UPDATING` table. TODO
- No more than 1 Throughput decrease/calendar day. BUGGY (24h instead of calendar day)
- No more than \*2 Throughput increase/update. DONE
- At least 10% change per update. TODO

### Types and items Limitations

- Table names can only be between 3 and 255 bytes long. DONE
- Table names can only contains a-z, A-Z, 0-9, '\_', '-', and '.'. DONE
- No more than 64kB/Item including fieldname and indexing overhead. TODO
- Primary key names can only be between 1 and 255 bytes long. DONE
- Attribute value can *not* be Null. DONE
- `hash_key` value smaller than 2048 bytes. TODO
- `range_key` value smaller than 1024 bytes. TODO
- Numbers can have up to 38 digits precision and can be between  $10^{-128}$  to  $10^{+126}$ . PARTIAL

## 2.1.3 Change log - Migration guide.

### ddbmock 0.3

Initial ddbmock release. This is *alpha quality* software. Some import features such as “Exclusive Start Key”, “Reverse” and “Limit” as well as `BatchWriteItem` have not been written (yet).

### Additions

- entry-point WEB (network mode)
- entry-point Boto (standalone mode)
- support for `CreateTable` method
- support for `DeleteTable` method

- support for `UpdateTable` method
- support for `DescribeTable` method
- support for `GetItem` method
- support for `PutItem` method
- support for `DeleteItem` method
- support for `UpdateItem` method (small approximations)
- support for `BatchGetItem` method (initial)
- support for `Query` method (initial)
- support for `Scan` method (initial)
- all comparison operators
- aggressive input validation

#### Known bugs - limitations

- no support for `BatchWriteItem`
- no support for “Exclusive Start Key”, “Reverse” and “Limit” in

`Query` and `Scan` - no support for “UnprocessedKeys” in `BatchGetItem` - Web entry-point is untested, fill bugs if necessary :)

## 2.2 Indices and tables

- *genindex*
- *modindex*
- *search*



# CONTRIBUTE

Want to contribute, report a bug or request a feature ? The development goes on BitBucket:

- **Download:** <http://pypi.python.org/pypi/ddbmock>
- **Report bugs:** <https://bitbucket.org/Ludia/dynamodb-mock/issues>
- **Fork the code:** <https://bitbucket.org/Ludia/dynamodb-mock/overview>