
storj documentation

Release 0.1.7

Storj

Dec 24, 2016

1	Installation	1
1.1	pip	1
1.2	source	1
2	command-line	3
3	API documentation	5
3.1	storj package	5
	Python Module Index	19

Installation

This section covers the [Storj](#) installation.

1.1 pip

```
$ pip install storj
```

1.2 source

The code is available at [github](#).

There are several alternatives for this installation method.

Here we only describe two possibilities:

- you can clone the repository:

```
$ git clone git@github.com:Storj/storj-python-sdk.git
```

- you can go to the [releases](#) tab and pick a [tarball](#) or [zip](#).

For example:

```
$ curl -OL https://github.com/Storj/storj-python-sdk/archive/1.0.0.tar.gz
```

Once you have a copy of the source and have extracted its files, you can install it using:

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

command-line

To install the command-line tool do:

```
$ pip install storj[cli]
```

API documentation

This section contains information on a specific function, class, or method.

3.1 storj package

3.1.1 Subpackages

storj.cli package

Module contents

Storj command-line interface package.

3.1.2 Submodules

storj.api module

Storj API module.

ecdsa_to_hex (*ecdsa_key*)

Return hexadecimal string representation of the ECDSA key.

Parameters **ecdsa_key** (*bytes*) – ECDSA key.

Raises `TypeError` – if the ECDSA key is not an array of bytes.

Returns hexadecimal string representation of the ECDSA key.

Return type `str`

storj.configuration module

APP_NAME = 'storj'

(*str*) – the application name.

CFG_EMAIL = 'email'

(*str*) – configuration parameter that holds the Storj account email address.

CFG_PASSWORD = 'password'

(*str*) – configuration parameter that holds the Storj account password.

read_config()

Reads configuration storj client configuration.

Mac OS X (POSIX): ~/.foo-bar

Unix (POSIX): ~/.foo-bar

Win XP (not roaming): C:\Documents and Settings\<user>\Application Data\storj

Win 7 (not roaming): C:\Users\<user>\AppData\Local\storj

Returns storj account credentials (email, password).

Return type (tuple[str, str])

storj.exception module

Storj exception module.

exception StorjBridgeApiError

Bases: `exceptions.Exception`

Generic Storj exception.

storj.http module

Storj HTTP module.

class Client (*email, password*)

Bases: `object`

api_url

str – the Storj API endpoint.

session

email

str – user email address.

password

str – user password.

private_key

public_key

public_key_hex

authenticate (*ecdsa_private_key=None*)

bucket_create (*name, storage=None, transfer=None*)

Create storage bucket.

See [API buckets: POST /buckets](#)

Parameters

- **name** (*str*) – name.
- **storage** (*int*) – storage limit (in GB).
- **transfer** (*int*) – transfer limit (in GB).

Returns bucket.

Return type (`model.Bucket`)

bucket_delete (*bucket_id*)

Destroy a storage bucket.

See [API buckets: DELETE /buckets/{id}](#)

Parameters **bucket_id** (*string*) – unique identifier.

bucket_files (*bucket_id*)

List all the file metadata stored in the bucket.

See [API buckets: GET /buckets/{id}/files](#)

Parameters **bucket_id** (*string*) – unique identifier.

Returns to be changed to model in the future.

Return type (`dict`)

bucket_get (*bucket_id*)

Return the bucket object.

See [API buckets: GET /buckets](#)

Parameters **bucket_id** (*str*) – bucket unique identifier.

Returns bucket.

Return type (`model.Bucket`)

bucket_list ()

List all of the buckets belonging to the user.

See [API buckets: GET /buckets](#)

Returns buckets.

Return type (`generator[model.Bucket]`)

bucket_set_keys (*bucket_id, bucket_name, keys*)

Update the bucket with the given public keys.

See [API buckets: PATCH /buckets/{bucket_id}](#)

Parameters

- **bucket_id** (*str*) – bucket unique identifier.
- **bucket_name** (*str*) – bucket name.
- **keys** (*list[str]*) – public keys.

Returns updated bucket information.

Return type (`storj.model.Bucket`)

bucket_set_mirrors (*bucket_id, file_id, redundancy*)

Establishes a series of mirrors for the given file.

See [API buckets: POST /buckets/{id}/mirrors](#)

Parameters

- **bucket_id** (*str*) – bucket unique identifier.
- **file_id** (*str*) – file unique identifier.
- **redundancy** (*int*) – number of replicas.

Returns the mirror settings.

Return type (*storj.model.Mirror*)

contact_list (*page=1, address=None, protocol=None, user_agent=None, connected=None*)

Lists contacts.

See [API contacts: GET /contacts](#)

Parameters

- **page** (*str*) – pagination indicator.
- **address** (*str*) – hostname or IP address.
- **protocol** (*str*) – SemVer protocol tag.
- **user_agent** (*str*) – Storj user agent string for farming client.
- **connected** (*bool*) – filter results by connection status.

Returns list of contacts.

Return type (list[*storj.model.Contact*])

contact_lookup (*node_id*)

Lookup for contact information of a node.

See [API contacts: GET /contacts/{nodeID}](#)

Parameters **node_id** (*str*) – node unique identifier.

Returns contact information

Return type (*storj.model.Contact*)

file_download (*bucket_id, file_id*)

file_metadata (*bucket_id, file_id*)

Get file metadata.

See [API buckets: GET /buckets/{id}/files/{file_id}/info](#)

Parameters

- **bucket_id** (*str*) – bucket unique identifier.
- **file_id** (*str*) – file unique identifier.

Returns file metadata.

Return type (*storj.model.File*)

file_pointers (*bucket_id, file_id, skip=None, limit=None*)

Get list of pointers associated with a file.

See [API buckets: GET /buckets/{id}/files/{file_id}](#)

Parameters

- **bucket_id** (*str*) – bucket unique identifier.
- **file_id** (*str*) – file unique identifier.
- **skip** (*str*) – pointer index to start the file slice.
- **limit** (*str*) – number of pointers to resolve tokens for.

Returns file pointers.

Return type (generator[*storj.model.FilePointer*])

file_remove (*bucket_id*, *file_id*)

Delete a file pointer from a specified bucket.

See [API buckets: DELETE /buckets/{id}/files/{file_id}](#)

Parameters

- **bucket_id** (*str*) – bucket unique identifier.
- **file_id** (*str*) – file unique identifier.

file_upload (*bucket_id*, *file*, *frame*)

Upload file.

See [API buckets: POST /buckets/{id}/files](#)

Parameters

- **bucket_id** (*str*) – bucket unique identifier.
- **file** (*storj.model.File*) – file to be uploaded.
- **frame** (*storj.model.Frame*) – frame used to stage file.

frame_add_shard (*shard*, *frame_id*)

Adds a shard item to the staging frame and negotiates a storage contract.

See [API frames: PUT /frames/{frame_id}](#)

Parameters

- **shard** (*storj.models.Shard*) – the shard.
- **frame_id** (*str*) – the frame unique identifier.

frame_create ()

Creates a file staging frame.

See [API frames: POST /frames](#)

Returns the frame.

Return type (*storj.model.Frame*)

frame_delete (*frame_id*)

Destroys the file staging frame by it's unique ID.

See [API frames: DELETE /frames/{frame_id}](#)

Parameters **frame_id** (*str*) – unique identifier.

frame_get (*frame_id*)

Fetches the file staging frame by it's unique ID.

See [API frame: GET /frames/{frame_id}](#)

Parameters **frame_id** (*str*) – unique identifier.

Returns a frame.

Return type (*storj.model.Frame*)

frame_list ()

Returns all open file staging frames.

See [‘API frame: GET /frames < https://storj.github.io/bridge/#!/frames/get_frames>’](#)

Returns all open file staging frames.

Return type (generator[*storj.model.Frame*])

key_delete (*public_key*)

Removes a public ECDSA keys.

See [API keys: DELETE /keys/{pubkey}](#)

Parameters **public_key** (*str*) – key to be removed.

key_dump ()

key_export ()

key_generate ()

key_import (*private_keyfile_path*, *public_keyfile_path*)

key_list ()

Lists the public ECDSA keys associated with the user.

See [API keys: GET /keys](#)

Returns public keys.

Return type (list[str])

key_register (*public_key*)

Register an ECDSA public key.

See [API keys: POST /keys](#)

Returns public keys.

Return type (list[*storj.model.Key*])

logger = <logging.Logger object>

password

(*str*) – user password

token_create (*bucket_id*, *operation*)

Creates a token for the specified operation.

See [API buckets: POST /buckets/{id}/tokens](#)

Parameters

- **bucket_id** (*str*) – bucket unique identifier.
- **operation** (*str*) – operation.

Returns

...

Return type (dict)

user_activate (*token*)

Activate user.

See [API users: GET /activations/{token}](#)

Parameters **token** (*str*) – activation token.

user_activation_email (*email*, *token*)

Send user activation email.

See [API users: POST /activations/{token}](#)

Parameters

- **email** (*str*) – user’s email address.
- **token** (*str*) – activation token.

user_create (*email*, *password*)

Create a new user with Storj bridge.

See [API users: POST /users](#)

Parameters

- **email** (*str*) – user’s email address.
- **password** (*str*) – user’s password.

user_deactivate (*token*)

Discard activation token.

See [API users: GET /activations/{token}](#)

Parameters **token** (*str*) – activation token.

user_delete (*email*)

Delete user account.

See [API users: DELETE /users/{email}](#)

Parameters **email** (*str*) – user’s email address.

user_reset_password (*email*)

Request a password reset.

See [API users: PATCH /users/{email}](#)

Parameters **email** (*str*) – user’s email address.

user_reset_password_confirmation (*token*)

Confirm a password reset request.

See [API users: GET /resets/{token}](#)

Parameters **token** (*str*) – password reset token.

storj.metadata module

Storj metadata module.

storj.model module

Storj model module.

class **Bucket** (*id=None*, *name=None*, *status=None*, *user=None*, *created=None*, *storage=None*, *transfer=None*, *pubkeys=None*, *publicPermissions=None*, *encryptionKey=None*)
Bases: `steenzout.object.Object`

Storage bucket.

A bucket is a logical grouping of files which the user can assign permissions and limits to.

id
str – unique identifier.

name
str – name.

status
str – bucket status (Active, ...).

user
str – user email address.

created
`datetime.datetime` – time when the bucket was created.

storage
int – storage limit (in GB).

transfer
int – transfer limit (in GB).

pubkeys

delete()

class Contact (*address=None, port=None, nodeID=None, lastSeen=None, protocol=None, userAgent=None*)
Bases: `steenzout.object.Object`

Contact.

address
str – hostname or IP address.

port
str – .

nodeID
str – node unique identifier.

lastSeen
str – .

protocol
str – SemVer protocol tag.

userAgent
str

lastSeen

class File (*bucket=None, hash=None, mimetype=None, filename=None, size=None, id=None, frame=None*)
Bases: `steenzout.object.Object`

bucket
bucket unique identifier.

hash

mimetype

filename

frame
`storj.model.Frame` – file frame.

size
shard_manager
content_type
delete()
download()
name

class FilePointer (*hash=None, token=None, operation=None, channel=None*)

Bases: `steenzout.object.Object`

File pointer.

Parameters

- **hash** (*str*) –
- **token** (*str*) – token unique identifier.
- **operation** (*str*) –
- **channel** (*str*) –

hash
str

token
storj.model.Token – token.

operation
str

channel
str

class Frame (*id=None, created=None, shards=None, locked=None, user=None, size=None*)

Bases: `steenzout.object.Object`

File staging frame.

id
str – unique identifier.

created
`datetime.datetime` – time when the bucket was created.

shards
`list[Shard]` – shards that compose this frame.

class KeyPair (*pkey=None, secret=None*)

Bases: `object`

ECDSA key pair.

Parameters

- **pkey** (*str*) – hexadecimal representation of the private key (secret exponent).
- **secret** (*str*) – master password.

keypair
`pycoin.key.Key.Key` – BIP0032-style hierarchical wallet.

Raises *NotImplementedError* when – a randomness source is not found.

address

(*)* – base58 encoded bitcoin address version of the nodeID.

node_id

(*str*) – NodeID derived from the public key (RIPEMD160 hash of public key).

private_key

(*str*) – private key.

public_key

(*str*) – public key.

class Keyring

Bases: `steenzout.object.Object`

export_keyring (*password, salt, user_pass*)

generate (*)*

import_keyring (*filepath*)

class MerkleTree (*leaves, prehashed=True*)

Bases: `steenzout.object.Object`

Simple merkle hash tree. Nodes are stored as strings in rows. Row 0 is the root node, row 1 is its children, row 2 is their children, etc

Arguments

leaves (*list[str]/types.generator[str]*): leaves of the tree, as hex digests

leaves

list[str] – leaves of the tree, as hex digests

depth

int – the number of levels in the tree

count

int – the number of nodes in the tree

rows

list[list[str]] – the levels of the tree

depth

Calculates the depth of the tree.

Returns tree depth.

Return type (*int*)

get_level (*depth*)

Returns the tree row at the specified depth

get_root (*)*

Return the root of the tree

leaves

(*list[str]/types.generator[str]*) – leaves of the tree.

class Mirror (*hash=None, mirrors=None, status=None*)

Bases: `steenzout.object.Object`

Mirror or file replica settings.

```

hash
    str

mirrors
    int – number of file replicas.

status
    str – current file replica status.

class Shard (id=None, hash=None, size=None, index=None, challenges=None, tree=None, exclude=None)
    Bases: steenzout.object.Object

    Shard.

    id
        str – unique identifier.

    hash
        str – hash of the data.

    size
        long – size of the shard in bytes.

    index
        int – numeric index of the shard in the frame.

    challenges
        list[str] – list of challenge numbers

    tree
        list[str] – audit merkle tree

    exclude
        list[str] – list of farmer nodeIDs to exclude

    add_challenge (challenge)
        Append challenge.

        Parameters challenge (str) – .

    add_tree (tree)
        Append tree.

    all ()

    get_private_record ()

    get_public_record ()

class ShardManager (filepath, shard_size, nchallenges=12)
    Bases: steenzout.object.Object

    File shard manager.

    filepath
        str – path to the file.

    index
        int – number of shards for the given file.

    nchallenges
        int – number of challenges to be generated.

    shard_size
        int/long – split file in chunks of this size.

```

shards

list[*Shard*] – list of shards

filepath

(*str*) – path to the file.

static hash (*data*)

Returns ripemd160 of sha256 of a string as a string of hex.

Parameters **data** (*str*) – content to be digested.

Returns the ripemd160 of sha256 digest.

Return type (*str*)

class Token (*token=None, bucket=None, operation=None, expires=None, encryptionKey=None*)

Bases: `steenzout.object.Object`

Token.

Parameters

- **token** (*str*) – token unique identifier.
- **bucket** (*str*) – bucket unique identifier.
- **()** (*operation*) –
- **expires** (*str*) – expiration date, in the RFC3339 format.
- **encryptionKey** (*str*) –

id

str – token unique identifier.

bucket

storj.model.Bucket – bucket.

operation

str

expires

datetime.datetime – expiration date, in UTC.

encryptionKey

str

storj.web_socket module

Storj web socket module.

class Client (*pointer, file_contents*)

Bases: `ws4py.client.threadedclient.WebSocketClient`

Web socket client.

json

generator[*storj.model.FilePointer*] – file pointers.

closed (*code, reason=None*)

opened ()

received_message (*m*)

3.1.3 Module contents

Storj package.

class BucketKeyManager (*bucket, authorized_public_keys*)

bucket

add (*key*)

all ()

clear ()

remove (*key*)

class BucketManager

Bases: `abc.ABCMeta`

Class to manage buckets.

static all ()

static create (*name, storage_limit=None, transfer_limit=None*)

Create bucket.

Parameters

- **name** (*str*) – .
- **()** (*transfer_limit*) – .
- **()** – .

static delete (*bucket_id*)

Remove bucket.

Parameters **bucket_id** (*int*) – bucket unique identifier.

static get (*bucket_id*)

class FileManager (*bucket_id*)

all ()

delete (*bucket_id, file_id*)

download (*file_id*)

upload (*file, frame*)

class TokenManager (*bucket_id*)

Bucket token manager.

bucket_id

int – bucket unique identifier.

create (*operation*)

Creates a token.

Parameters **operation** (*str*) – operation (PUSH or PULL).

class UserKeyManager

Bases: `abc.ABCMeta`

static add (*key*)

static all ()

static clear ()

static remove (*key*)

generate_new_key_pair ()

Generate a new key pair.

Returns

tuple(ecdsa.keys.SigningKey, ecdsa.keys.VerifyingKey):

key pair (private, public).

get_client ()

Returns a pre-configured Storj HTTP client.

Returns Storj HTTP client.

Return type (*storj.http.Client*)

S

- `storj`, [17](#)
- `storj.api`, [5](#)
- `storj.cli`, [5](#)
- `storj.configuration`, [5](#)
- `storj.exception`, [6](#)
- `storj.http`, [6](#)
- `storj.metadata`, [11](#)
- `storj.model`, [11](#)
- `storj.web_socket`, [16](#)

A

add() (BucketKeyManager method), 17
add() (UserKeyManager static method), 17
add_challenge() (Shard method), 15
add_tree() (Shard method), 15
address (Contact attribute), 12
address (KeyPair attribute), 14
all() (BucketKeyManager method), 17
all() (BucketManager static method), 17
all() (FileManager method), 17
all() (Shard method), 15
all() (UserKeyManager static method), 17
api_url (Client attribute), 6
APP_NAME (in module storj.configuration), 5
authenticate() (Client method), 6

B

bucket (BucketKeyManager attribute), 17
Bucket (class in storj.model), 11
bucket (File attribute), 12
bucket (Token attribute), 16
bucket_create() (Client method), 6
bucket_delete() (Client method), 7
bucket_files() (Client method), 7
bucket_get() (Client method), 7
bucket_id (TokenManager attribute), 17
bucket_list() (Client method), 7
bucket_set_keys() (Client method), 7
bucket_set_mirrors() (Client method), 7
BucketKeyManager (class in storj), 17
BucketManager (class in storj), 17

C

CFG_EMAIL (in module storj.configuration), 5
CFG_PASSWORD (in module storj.configuration), 5
challenges (Shard attribute), 15
channel (FilePointer attribute), 13
clear() (BucketKeyManager method), 17
clear() (UserKeyManager static method), 18
Client (class in storj.http), 6

Client (class in storj.web_socket), 16
closed() (Client method), 16
Contact (class in storj.model), 12
contact_list() (Client method), 8
contact_lookup() (Client method), 8
content_type (File attribute), 13
count (MerkleTree attribute), 14
create() (BucketManager static method), 17
create() (TokenManager method), 17
created (Bucket attribute), 12
created (Frame attribute), 13

D

delete() (Bucket method), 12
delete() (BucketManager static method), 17
delete() (File method), 13
delete() (FileManager method), 17
depth (MerkleTree attribute), 14
download() (File method), 13
download() (FileManager method), 17

E

ecdsa_to_hex() (in module storj.api), 5
email (Client attribute), 6
encryptionKey (Token attribute), 16
exclude (Shard attribute), 15
expires (Token attribute), 16
export_keyring() (Keyring method), 14

F

File (class in storj.model), 12
file_download() (Client method), 8
file_metadata() (Client method), 8
file_pointers() (Client method), 8
file_remove() (Client method), 9
file_upload() (Client method), 9
FileManager (class in storj), 17
filename (File attribute), 12
filepath (ShardManager attribute), 15, 16
FilePointer (class in storj.model), 13

Frame (class in storj.model), 13
 frame (File attribute), 12
 frame_add_shard() (Client method), 9
 frame_create() (Client method), 9
 frame_delete() (Client method), 9
 frame_get() (Client method), 9
 frame_list() (Client method), 9

G

generate() (Keyring method), 14
 generate_new_key_pair() (in module storj), 18
 get() (BucketManager static method), 17
 get_client() (in module storj), 18
 get_level() (MerkleTree method), 14
 get_private_record() (Shard method), 15
 get_public_record() (Shard method), 15
 get_root() (MerkleTree method), 14

H

hash (File attribute), 12
 hash (FilePointer attribute), 13
 hash (Mirror attribute), 14
 hash (Shard attribute), 15
 hash() (ShardManager static method), 16

I

id (Bucket attribute), 11
 id (Frame attribute), 13
 id (Shard attribute), 15
 id (Token attribute), 16
 import_keyring() (Keyring method), 14
 index (Shard attribute), 15
 index (ShardManager attribute), 15

J

json (Client attribute), 16

K

key_delete() (Client method), 10
 key_dump() (Client method), 10
 key_export() (Client method), 10
 key_generate() (Client method), 10
 key_import() (Client method), 10
 key_list() (Client method), 10
 key_register() (Client method), 10
 KeyPair (class in storj.model), 13
 keypair (KeyPair attribute), 13
 Keyring (class in storj.model), 14

L

lastSeen (Contact attribute), 12
 leaves (MerkleTree attribute), 14
 logger (Client attribute), 10

M

MerkleTree (class in storj.model), 14
 mimetype (File attribute), 12
 Mirror (class in storj.model), 14
 mirrors (Mirror attribute), 15

N

name (Bucket attribute), 12
 name (File attribute), 13
 nchallenges (ShardManager attribute), 15
 node_id (KeyPair attribute), 14
 nodeID (Contact attribute), 12

O

opened() (Client method), 16
 operation (FilePointer attribute), 13
 operation (Token attribute), 16

P

password (Client attribute), 6, 10
 port (Contact attribute), 12
 private_key (Client attribute), 6
 private_key (KeyPair attribute), 14
 protocol (Contact attribute), 12
 pubkeys (Bucket attribute), 12
 public_key (Client attribute), 6
 public_key (KeyPair attribute), 14
 public_key_hex (Client attribute), 6

R

read_config() (in module storj.configuration), 6
 received_message() (Client method), 16
 remove() (BucketKeyManager method), 17
 remove() (UserKeyManager static method), 18
 rows (MerkleTree attribute), 14

S

session (Client attribute), 6
 Shard (class in storj.model), 15
 shard_manager (File attribute), 13
 shard_size (ShardManager attribute), 15
 ShardManager (class in storj.model), 15
 shards (Frame attribute), 13
 shards (ShardManager attribute), 15
 size (File attribute), 12
 size (Shard attribute), 15
 status (Bucket attribute), 12
 status (Mirror attribute), 15
 storage (Bucket attribute), 12
 storj (module), 17
 storj.api (module), 5
 storj.cli (module), 5
 storj.configuration (module), 5

storj.exception (module), [6](#)
storj.http (module), [6](#)
storj.metadata (module), [11](#)
storj.model (module), [11](#)
storj.web_socket (module), [16](#)
StorjBridgeApiError, [6](#)

T

Token (class in storj.model), [16](#)
token (FilePointer attribute), [13](#)
token_create() (Client method), [10](#)
TokenManager (class in storj), [17](#)
transfer (Bucket attribute), [12](#)
tree (Shard attribute), [15](#)

U

upload() (FileManager method), [17](#)
user (Bucket attribute), [12](#)
user_activate() (Client method), [10](#)
user_activation_email() (Client method), [10](#)
user_create() (Client method), [11](#)
user_deactivate() (Client method), [11](#)
user_delete() (Client method), [11](#)
user_reset_password() (Client method), [11](#)
user_reset_password_confirmation() (Client method), [11](#)
userAgent (Contact attribute), [12](#)
UserKeyManager (class in storj), [17](#)