storj documentation

Release 0.1.5

Storj

Nov 29, 2016

Contents

CHAPTER 1

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
```

CHAPTER 2

command-line

To install the command-line tool do:

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

CHAPTER 3

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 identitifer.
- **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*)
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 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 – numberic 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

T

U