pysyncgateway Documentation

Release 1.2.0

Construct Technology Ltd

Oct 12, 2018

Contents:

CHAPTER 1

pysyncgateway

Basic object orientated library for communicating with Couchbase Sync Gateway, primarily through the admin port.

1.1 Resources

- Documentation on ReadTheDocs
- Package on PyPI
- Source code on GitHub
- Licensed on Apache License 2.0
- Changelog

Tested against Pythons 2.7 and 3.5 (WIP); Sync Gateway 1.5 community edition (walrus mode).

1.1.1 Quickstart

Install

You can install `pysyncgateway` from PyPi:

```
pip install pysyncgateway
```

Make an Admin Client

Assuming that you have a Sync Gateway running with its admin port on `http://localhost:4985/`, create an instance of *AdminClient* to connect:

```
>>> from pysyncgateway import AdminClient
>>> admin_client = AdminClient('http://localhost:4985/')
```

Check that your `admin_client` instance is connected to the admin port by loading the server info from Sync Gateway:

```
>>> server_info = admin_client.get_server()
>>> sorted(list(server_info))
[u'ADMIN', u'couchdb', u'vendor', u'version']
>>> server_info['ADMIN']
True
>>> server_info['version']
u'Couchbase Sync Gateway/1.5.1(4;cb9522c)'
```

You can use the admin client to load a list of databases currently on the Sync Gateway (the default Docker container is initialised with a database called ‘db’):

```
>>> print(admin_client.all_databases())
[<Database "http://localhost:4985/db/">]
```

Create Database

Create a new instance of *`Database`* to contain our test user and document:

```
>>> database = admin_client.get_database('test')
>>> database.create()
True
```

The new ‘test’ database will not contain any documents or users:

```
>>> database.all_docs()
[]
>>> database.all_users()
[]
```

Create some Documents

First create a *`Document`* with the ID ‘message’. This will have the “Hello World!” content and be in the ‘world’ channel (we’ll use this to test with our User later):

```
>>> hello_doc = database.get_document('message')
>>> hello_doc.data = {'content': 'Hello World!'}
>>> hello_doc.set_channels('world')
>>> hello_doc.create_update()
1
```

Now create a second document with ID ‘stuff’ - this is not saved in any channels:

```
>>> other_doc = database.get_document('stuff')
>>> other_doc.data = {'private_info': 'Secret things'}
>>> other_doc.create_update()
1
```

Finally, check with the admin client that those two documents are in the database.

```
>>> sorted(database.all_docs())
[<Document "http://localhost:4985/test/message">, <Document "http://localhost:4985/
↪test/stuff">]
```

Create a User

Now we need a *User* in the database to check that our created documents work OK - we create this from the database instance. At first the user instance will not be subscribed to any channels:

```
>>> user = database.get_user('friend')
>>> user.set_password('__PASSWORD__')
>>> user.create_update()
1
```

pysyncgateway provides a *UserClient* which we can now connect to the public port at http://localhost:4984/ with the credentials we created for the 'friend' User above. Again, load the server info to ensure that the client is connected - but this time there is no 'ADMIN' key in the response because the client is connected on the public port.

```
>>> from pysyncgateway import UserClient
>>> user_client = UserClient('http://localhost:4984/')
>>> user_client.auth('friend', '__PASSWORD__')
>>> server_info = user_client.get_server()
>>> sorted(list(server_info))
[u'couchdb', u'vendor', u'version']
```

Now check a list of the documents that the user can access. We first have to generate a second database instance - this one is for the user client rather than the admin client.

```
>>> user_database = user_client.get_database('test')
>>> user_database.all_docs()
[]
```

They have no access to any documents!

Grant access to the 'message' document by using the admin client to subscribe the 'friend' User to the 'world' channel:

```
>>> user.set_admin_channels('world')
>>> user.create_update()
2
```

Now the 'friend' user can retrieve the message document:

```
>>> user_docs = user_database.all_docs()
>>> user_docs
[<Document "http://localhost:4984/test/message">]
>>> message = user_docs[0]
>>> message.retrieve()
True
>>> message.data
{u'content': u'Hello World!'}
```

Success!

Clean up

Finally, the admin client can be used to remove the 'test' database. This will cascade into the Sync Gateway and remove all users and documents in that database:

```
>>> database.delete()
True
```

1.1.2 pysyncgateway package

Submodules

pysyncgateway.admin_client module

class `pysyncgateway.admin_client.`**`AdminClient`**(*url*)

Bases: *`pysyncgateway.client.Client`*

Sync Gateway admin client for performing actions on the private admin API.

`url`

str – Sync Gateway admin REST API URL.

`all_databases`()

Provide all Databases on the server.

`GET /_all_dbs`

Returns All databases found, connected with this client.

Return type list (*Database*)

Raises *`GatewayDown`* – When sync gateway instance can not be reached by client.

pysyncgateway.client module

class `pysyncgateway.client.`**`Client`**(*url*)

Bases: *`pysyncgateway.helpers.ComparableMixin`*, `object`

Abstract parent class for `AdminClient` and `UserClient`.

`_auth`

requests.HTTPBasicAuth – Initialises to `None` and is only used by `UserClient`.

`url`

str – Sync Gateway REST API URL.

`CONFLICT = 3`

`CREATED = 1`

`UPDATED = 2`

`delete`(***kwargs*)

`get`(***kwargs*)

`get_database`(*database_name*)

Get a `Database` instance connected to this client.

Parameters **`database_name`** (*`str`*) – Name of database.

Returns Database

`get_server`()

Returns Meta-information about the server.

Return type dict

`post`(***kwargs*)

`put`(***kwargs*)

pysyncgateway.data_dict module

class `pysyncgateway.data_dict.`**`DataDict`**

Bases: `dict`

DataDict is the developer facing dictionary of data contained within a Couchbase Document. It prevents settings items with reserved Couchbase keys like "_rev" or "_id", but still acts like a dictionary to make manipulating Document data easy.

NOTE: It is not design agnostic because it protects the "channels" key. Depending on your application and sync function, that might not be a list of channels in your data design.

`filtered_keys = (u'_id', u'_rev', u'channels')`

`classmethod from_dict`(*data*)

Given a dictionary *data*, create a new DataDict from a dictionary *data*, silently removing all filtered keys from that input.

Parameters **`data`** (*`dict`*) – Input data.

Returns New instance created with cleaned, copied *data*.

Return type *DataDict*

Raises `ValueError` – When passed a non-dict.

`to_dict`()

Returns A dictionary version of self

Return type dict

pysyncgateway.database module

class `pysyncgateway.database.`**`Database`**(*client*, *name*)

Bases: *`pysyncgateway.helpers.ComparableMixin`*, `object`

A Database on Sync Gateway.

`client`

AdminClient

`name`

str

`url`

str – URL to the database, created at init time, including trailing slash.

`all_docs`()

Get list of all Documents in database.

`GET /:name/_all_docs`

Warning: Use for testing only. From Simon @ Couchbase:

> We would strongly advise against using the `_all_docs` endpoint. As your database grows relying on the View that this calls to return to you every document key is inadvisable and does not scale well to very high numbers of documents.
>
> If you need to retrieve or update multiple documents please use the `_bulk_get` and `_bulk_docs` end points to supply a list of keys (or documents) for retrieval or update.

Returns An instance of Document for each document returned by the endpoint. For each instance the `data['_rev']` value is populated with the revision ID from `value.rev`.

Return type list (*Document*)

Raises *`DoesNotExist`* – Database can't be found on Sync Gateway.

`all_users`()

`GET /:name/_user/`

Returns All Users in Database.

Return type list (*User*)

`bulk_docs`(*docs*, *new_edits=False*)

Update multiple documents.

`POST /:name/_bulk_docs`

Parameters

- **`docs`** (*`list (Document)`*) – Documents to be created.
- **`new_edits`** (*`bool, Optional`*) – Value for the `new_edits` value passed in the POST data. When deleting open revisions, this should be set to `None` so that no `new_edits` value is sent in the POST data - this is required for the deletion to be successful. Defaults to `False`.

Returns Bulk document update was accepted.

Return type bool

Raises *`DoesNotExist`* – Database can't be found on Sync Gateway.

`create`()

Write this Database instance to Sync Gateway.

Uses test orientated settings (i.e. none - the empty dictionary `{}` is passed as data) to create database. This function is intended for test functionality, rather than for clients to be regularly creating databases.

`PUT /:name/`

Returns Creation was successful.

Return type bool

`delete`()

Remove database.

Whereas Sync Gateway will raise 404 if the database is not found, this fails silently with the intention that it can be used 'scatter gun' style at the end of test runs to clean up database lists.

`DELETE /:name/`

Returns Database was found and deleted.

Return type bool

`get`()

Return information about this Database from Sync Gateway.

`GET /:name/`

Returns Information loaded from SG.

Return type dict

Raises

- *DoesNotExist* – When database is not written to Sync Gateway regardless of whether the client is authorized or not.
- *ClientUnauthorized* – When database exists and client is not authorized.

get_document(*doc_id*)

Returns An instance of Document in this Database with provided `doc_id`.

Return type *Document*

get_query(*doc_id*)

Returns An instance of a query design document in this Database with the provided `doc_id`.

Return type *Query*

get_session()

Returns Session

get_user(*username*)

Returns An instance of User for the provided `username`.

Return type *User*

pysyncgateway.document module

class pysyncgateway.document.**Document**(*database*, *doc_id*)

Bases: *pysyncgateway.resource.Resource*

A Couchbase Document in a database.

channels

tuple (str) – Channels this document is in.

data

DataDict – Data from the Document using the *DataDict* manager. The *DataDict* instance prevents protected keys from entering the data, but does nothing to prevent mutation. Therefore it never contains the private SG fields '_id', '_rev', 'channels'.

doc_id

str – ID of document.

rev

str – Revision identifier of document. Set to empty string when no document has been retrieved.

to_delete

bool – Flag used by *Document.flatten_data()*. When set it generates data used to delete the Document when posted with *Database.bulk_docs()*.

open_revisions

list (Document) – List of previous revisions as Document instances. This will be populated when *Document.retrieve()* is called with `revs=True`.

url

str – URL for this resource on Sync Gateway.

create_update()

Save or update Document in Sync Gateway. Saves the received revision id into instance's `rev` attribute.

`PUT /<database_name>/<doc_id>`

Note: Works for updates but is not tested.

Returns `AdminClient.CREATED` if document was created (matches 201).

Return type int

Raises *`RevisionMismatch`* – When create (no revision) is tried on an existing Document or update is tried on an existing document, but the revision numbers do not match. Two args are passed to the exception: url of the document and any revision that was passed with the `PUT` request.

`delete``()`

Delete Document from its Database. Document must have been retrieved in order for a valid revision ID to be provided. If there isn't a cache of this information when a *delete* is asked for, then a pre-fetch will occur.

Uses the default *`Client.delete()`* action, but then inspects the response to ensure that `{"ok": true}`.

`DELETE /<name>/<doc_id>?rev=<rev>`

Returns Delete was successful.

Return type bool

Raises

- *`DoesNotExist`* – If Document can't be found (doc has to be loaded first to retrieve the revision number, which can yield a 404 if it doesn't exist). Also can be raised if the Database does not exist.
- *`RevisionMismatch`* – Provided `rev` parameter did not match the live revision ID on Sync Gateway at request time.

`flatten_data``()`

Used when posting multiple documents with *`Database.bulk_docs()`* to the `/_bulk_docs` endpoint. Set the `to_delete` flag on the Document instance if it should be deleted - only the particular revision will be marked for deletion, not all open revisions.

Returns Data for this document including `_rev` and `_id`.

Return type dict

Raises `ValueError` – Document needs a rev to be deletable.

`get_open_revisions``(`*include_deleted=False*`)`

Retrieve all leaf revisions of this document and update this instance with the currently winning revision's data.

Sync Gateway provides the leaf revision documents "as is" with no flag that one or other is the current document. Therefore this function makes two requests:

- A GET request using *`Document.retrieve()`*. This is used to find the currently winning revision.
- A GET request with `?open_revs=all` to collect all leaf nodes.

The list of leaf nodes is iterated and the currently winning revision is used to update this instance. Losing leaf revisions are blown up into *`Document`* instances and stored in the `open_revisions` list.

Parameters **`include_deleted`** (*bool, Optional*) – When set to `True`, Documents found that are not currently the winning revision and contain `{'_deleted': True}` are ignored. Defaults to `False`.

Returns Number of open revisions found including the current revision.

Return type int

Raises `RevisionMismatch` – When winning revision's rev was not found when the open revisions were loaded. This usually means that the winning revision was deleted before the open revisions could be retrieved. (untested and needs improvement to avoid race condition.)

`retrieve`()

Load document contents. Once loaded, `_rev` and `channels` are used to update the internal attributes before the data is sent to the DataDict.

`GET /<name>/<doc_id>`

Returns Load was successful.

Return type bool

Raises *`DoesNotExist`* – Document with provided doc_id can not be loaded.

Note: DataDict never contains the private Sync Gateway fields `_id`, `_rev`, `channels`.

Note: **Not** running through `/<name>/_raw/<doc_id>`.

Warning: Side effect: `self.channels` is updated based on returned the JSON using `self.set_channels()`.

Warning: Side effect: `self.data` is updated with data dictionary loaded from JSON.

Warning: Side effect: `self.rev` is updated from revision passed in JSON using `self.set_rev`.

`set_channels`(**channels*)

Validate each channel passed and save to channels attribute. No update to channels is made if one is bad, all are rejected.

Parameters **`*args`** (*str*) – Channels to be set for this document.

Returns None

Raises *`InvalidChannelName`* – When a bad channel name is passed.

`set_rev`(*revision_id*)

Set Document's revision id. This is used for subsequent updates and deletions. Not checked for any validity.

Parameters **`revision_id`** (*str*) –

Returns None

Raises `ValueError` – When revision_id is empty string.

pysyncgateway.exceptions module

exception `pysyncgateway.exceptions.`**`ClientUnauthorized`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

Client is not authorized to access this URL

exception `pysyncgateway.exceptions.`**`DoesNotExist`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

Generic exception to replace 404s. Used if databases, users or documents can't be loaded.

exception `pysyncgateway.exceptions.`**`GatewayDown`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

SyncGateway could not be reached on configured URL

exception `pysyncgateway.exceptions.`**`InvalidChannelName`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

exception `pysyncgateway.exceptions.`**`InvalidDataKey`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

exception `pysyncgateway.exceptions.`**`InvalidDatabaseName`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

exception `pysyncgateway.exceptions.`**`InvalidDocumentID`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

exception `pysyncgateway.exceptions.`**`InvalidPassword`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

exception `pysyncgateway.exceptions.`**`NotLoaded`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

Item from a `Resources` data attribute was requested, but that resource has not yet been successfully retrieved from Sync Gateway.

exception `pysyncgateway.exceptions.`**`PysyncgatewayException`**

Bases: `exceptions.Exception`

The root of all Evil >:D

exception `pysyncgateway.exceptions.`**`RevisionMismatch`**

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

Sync Gateway was not able to update a resource because either a rev number was not provided and the resource unexpectedly existed or the provided rev number did not match what Sync Gateway has.

`(str)`

URL resource that update was attempted for.

`(str)`

Revision that was sent with the update request.

exception `pysyncgateway.exceptions.`**`SyncGatewayClientErrorResponse`**(*status_code*, *json*)

Bases: *`pysyncgateway.exceptions.PysyncgatewayException`*

Sync Gateway responded with a 4xx error.

status_code
int – Error code in the response from Sync Gateway.

json
dict – Body of response from Sync Gateway.

classmethod from_response(*response*)

Parameters **response** (*requests.Response*) –

Returns SyncGatewayClientErrorResponse

pysyncgateway.helpers module

class pysyncgateway.helpers.**ComparableMixin**
Bases: `object`

Alex Martelli's suggestion from https://stackoverflow.com/a/1061350/1286705

pysyncgateway.helpers.**assert_valid_channel_name**(*name*)
Assert channel name passes Sync Gateway's requirements:

> Valid channel names consist of letter `[A-Z, a-z]`, digits `[0-9]`, and a few special characters `[= + / . , _ @]`. The empty string is not allowed. The special channel name `*` denotes all channels.

Docs are out of date: https://github.com/couchbase/sync_gateway/issues/656 Therefore adding `-` as allowed.

Parameters **name** (*str*) –

Raises `InvalidChannelName`

Examples

```
>>> from pysyncgateway.helpers import assert_valid_channel_name
```

1. Empty string is invalid:

```
>>> assert_valid_channel_name('')
Traceback (most recent call last):
...
InvalidChannelName: Empty channel name is not allowed
```

2. Special-special characters are not allowed:

```
>>> assert_valid_channel_name('stuff!channel')
Traceback (most recent call last):
...
InvalidChannelName: Special characters are not allowed in channels, first bad␣
↪character is "!"
```

3. White space is not allowed:

```
>>> assert_valid_channel_name('channel 2')
Traceback (most recent call last):
...
InvalidChannelName: Special characters are not allowed in channels, first bad␣
↪character is " "
```

4. When bad character matches at start of line, it's found:

```
>>> assert_valid_channel_name('$1 channel')
Traceback (most recent call last):
...
InvalidChannelName: Special characters are not allowed in channels, first bad␣
↪character is "$"
```

5. When name is just bad characters, first bad char is returned:

```
>>> assert_valid_channel_name('#&()`')
Traceback (most recent call last):
...
InvalidChannelName: Special characters are not allowed in channels, first bad␣
↪character is "#"
```

6. Some special characters are OK:

```
>>> assert_valid_channel_name('-ABC_project=+/.,@1234')
```

pysyncgateway.helpers.**assert_valid_database_name**(*name*)

Raise exception if database name does not match Couchbase requirements

From docs (http://docs.couchdb.org/en/stable/api/database/common.html#put–db):

The database name must begin with a lowercase letter.

The database name must contain only valid characters. The following characters are valid in database names:

- Lowercase letters: `a-z`
- Numbers: `0-9`
- Special characters: `_$()+-/`

NOTE hyphen is allowed, compared to channel names below.

Parameters **name** (*str*) –

Returns None

Raises InvalidDatabaseName – When passed `name` is invalid as a name for a database in Sync Gateway.

Examples

```
>>> from pysyncgateway.helpers import assert_valid_database_name
```

1. Empty string is invalid:

```
>>> assert_valid_database_name('')
Traceback (most recent call last):
...
InvalidDatabaseName: Empty database name is not allowed
```

2. Special-special characters are not allowed:

```
>>> assert_valid_database_name('stuff!db')
Traceback (most recent call last):
...
InvalidDatabaseName: Special characters are not allowed in database names,␣
↪first bad character is "!"
```

3. White space is not allowed:

```
>>> assert_valid_database_name('db 2')
Traceback (most recent call last):
...
InvalidDatabaseName: Special characters are not allowed in database names,␣
↪first bad character is " "
```

4. Capitals are not allowed:

```
>>> assert_valid_database_name('bIGdATA')
Traceback (most recent call last):
...
InvalidDatabaseName: Special characters are not allowed in database names,␣
↪first bad character is "I"
```

5. Name must start with a letter:

```
>>> assert_valid_database_name('-10degrees')
Traceback (most recent call last):
...
InvalidDatabaseName: Database names must start with a lowercase letter
```

6. Happy 'simple' database names are OK:

```
>>> assert_valid_database_name('construct-pm($)//x_x')
```

`pysyncgateway.helpers.`**`assert_valid_document_id`**(*doc_id*)

NOTE this could become prohibitive if the admin client can't delete documents that fall outside the boundaries of the set of allowed names. E.g. App creates bad document and admin needs to clean it up, but can't. Therefore this validation should only be used on creation.

Parameters **`doc_id`** (*str*) –

Returns `None`

Raises `InvalidDocumentID` – When provided `doc_id` is not a valid ID for a Document in Sync Gateway.

Examples

```
>>> from pysyncgateway.helpers import assert_valid_document_id
```

1. Empty string is invalid

```
>>> assert_valid_document_id('')
Traceback (most recent call last):
...
InvalidDocumentID: Empty document id is not allowed
```

2. Colon is not allowed in document id

```
>>> assert_valid_document_id('colon:nope')
Traceback (most recent call last):
...
InvalidDocumentID: Colon is not allowed in document ids
```

3. Question mark is not allowed (this is allowed in couchbase, but would require much urlencoding and mashing to get working via python IMO so banning it for now)

   If trying to quote, then `quote(doc_id, safe='')` will raise `KeyError` and Hell's Armies walk the earth.

```
>>> assert_valid_document_id('Questions? Nope')
Traceback (most recent call last):
...
InvalidDocumentID: Question mark is not allowed in document ids
```

4. Hash banned for the same reason as question mark.

```
>>> assert_valid_document_id('hashes### like a boss')
Traceback (most recent call last):
...
InvalidDocumentID: Hash is not allowed in document ids
```

5. Special-special characters are allowed, and can start the doc id.

```
>>> assert_valid_document_id('$stuff!channel')
```

6. White space is allowed.

```
>>> assert_valid_document_id('channel 2')
```

7. Loads of special characters are OK

```
>>> assert_valid_document_id('*-=|+/.,@(1234)')
```

`pysyncgateway.helpers.`**`sg_method`**(*func*, **args*, ***kwargs*)

Wrap a normal request Verb (E.g. `requests.get`) in exception handling logic.

Raises

- `DoesNotExist` – 404 was received for a request.
- `GatewayDown` – SyncGateway can't be reached.
- `RevisionMismatch` – When a 409 conflict is received from Sync Gateway. Two args are passed to the exception: url of the resource and any revision that was passed.
- `SyncGatewayClientErrorResponse` – When any "not OK" response (according to `requests.Response.ok` is received that is not a 404.

pysyncgateway.query module

class `pysyncgateway.query.`**`Query`**(*database*, *doc_id*)

Bases: *`pysyncgateway.resource.Resource`*

Query a design document.

data
DataDict – Data from the design document using the `DataDict` manager.

doc_id
str – ID of design document.

url
str – URL for this resource on Sync Gateway.

build_view_url(*view_name*)

Parameters **view_name** (*str*) –

Returns URL for querying view.

Return type str

create_update()
Create or update design document with data.

`PUT /<database_name>/_design/<doc_id>`

Returns Design document was created or updated successfully.

Return type bool

delete()
Delete design document.

Returns Design document deleted.

Return type bool

Raises `DoesNotExist` – Design document or Database can not be found.

query_view(*view_name*, *key=None*, *stale=True*, *timeout=None*)
Load a view function from this design Document. Document must be written to Sync Gateway and view's Javascript function must be valid.

Parameters

- **view_name** (*str*) – View's name.
- **key** (*Optional*) – Value to use to search the view's key (the left part of the map). Any type can be passed as long as:
  - `key is not None`
  - `key` can be serialized to a string by `json.dumps()`.

  To query a view with multiple keys set `key` as an iterable which will be serialized to JSON as an array. E.g. `key=['left_id', 'right_id']`.
- **stale** (*bool, Optional*) – Allow stale results in the view. This is currently the default value in Sync Gateway, so is only passed when set to `False`. Default `True`.

  `'false'` is an undocumented option for this param. See https://github.com/couchbase/sync_gateway/issues/727#issuecomment-83588984

  It also doesn't work in Walrus mode, see https://github.com/constructpm/pysyncgateway/issues/7
- **timeout** (*int, Optional*) – Set a time out of seconds as per requests' spec which means that if the Sync Gateway does not respond to the GET request within the `timeout` period, a `ReadTimeout` will be raised. Default `None` which means that requests' default is used.

Returns Decoded JSON for view data result. Will usually be a dictionary contain keys 'Collator', 'rows' and 'total_rows'.

Return type dict

Raises

- `DoesNotExist` – When database, design document or contained view can not be found.
- `requests.exceptions.ReadTimeout` – When Sync Gateway does not respond within given `timeout`.

`retrieve``()`

Returns Design document was retrieved.

Return type bool

Raises `DoesNotExist` – Design document or Database can not be found.

Side effects: data: Updates internal data dictionary with data loaded from JSON.

pysyncgateway.resource module

`class` `pysyncgateway.resource.`**`Resource`**`(`*database*`)`

Bases: *`pysyncgateway.helpers.ComparableMixin`*, `object`

A Couchbase object stored within a Database, identified by a URL and accessible through REST verbs. Data is stored in a DataDict manager.

`_data`

DataDict

`database`

Database

`url`

str

`data`

pysyncgateway.session module

`class` `pysyncgateway.session.`**`Session`**`(`*database*`)`

Bases: *`pysyncgateway.resource.Resource`*

Sessions on the Database are undocumented as of Sync Gateway 1.5 in both the public and admin REST APIs. But since they provide a useful test mechanism by showing a list of channels that the authenticated User has been subscribed to, they are included in this library.

`data`

DataDict – Data from the session using the `DataDict` manager.

`database`

Database

`url`

str – URL for the session in the Database on Sync Gateway.

`get_channels``()`

Helper to pick user's subscribed channels from retrieved session data.

Returns Sorted list of channel names found in response including the special `!` public channel.

Return type list

Raises

- `NotLoaded` – When session has not been retrieved.
- `KeyError` – When the `userCtx` data in the session response does not contain a `channels` dictionary.

get_name()

Helper to pick user's `name` field from retrieved session data.

Returns Name of user according to Sync Gateway.

Return type str

Raises

- `NotLoaded` – When session has not been retrieved.
- `KeyError` – When the `userCtx` data in the session response does not contain a `name` field.

retrieve()

Collect session information for the current client in the `database` attribute.

For unauthorized users on the public API and requests on the admin API this looks like:

```
{
    'authentication_handlers': ['default', 'cookie'],
    'ok': True,
    'userCtx': {
        'channels': {},
        'name': None,
    },
}
```

For authenticated users on the public API, in this example with the name `__USERNAME__` and channels `a`, `b` and `c`, this looks like:

```
{
    'authentication_handlers': ['default', 'cookie'],
    'ok': True,
    'userCtx': {
        'channels': {
            '!': 1,
            'a': 1,
            'b': 1,
            'c': 1,
        },
        'name': '__USERNAME__',
    },
}
```

Returns Success

Return type bool

Raises `DoesNotExist` – When database does not exist on Sync Gateway, regardless of whether client is authorized or not.

Warning: Side effect: Updates `self.data` with response from Sync Gateway on success.

pysyncgateway.stats module

class `pysyncgateway.stats.`**`Stats`**(*client*)

Bases: `object`

Stats object from the expvars endpoint on the Sync Gateway. See https://github.com/couchbase/sync_gateway/wiki/expvars

NOTE: Stats come from the server and not from the database.

`client`

AdminClient – Used to communicate with the server.

`data`

dict – Statistical data populated after retrieval. Will be empty dictionary before retrieval.

`url`

str – Location of stats.

`retrieve`()

Load stats, parse info and stash in data attr.

Returns load was successful.

Return type bool

Raises

- `GatewayDown` – When the endpoint can't be reached.
- `ValueError` – When something non-JSON is loaded.

Side effects: data: Populated with dictionary parsed from the JSON response.

pysyncgateway.user module

class `pysyncgateway.user.`**`User`**(*database*, *name*)

Bases: *`pysyncgateway.resource.Resource`*

A User in a Database.

`admin_channels`

tuple(str) – Channels that this User has been added to by admin or sync function.

`data`

DataDict

`name`

str – Username of this user.

`password`

str – User's password. Can be *None* and is only set with *set_password*.

`retrieved`

bool – User been retrieved from sync gateway. Acts as a flag to know if there should be a password sent at *create_update* time.

url
str – URL for this User on sync gateway.

create_update()
Create a new or update an existing user for currently connected database. When creating a new User then their password must have been set on this instance using *set_password*, or the sync gateway will reject the PUT with a 400.

NOTE it's not possible to change the username (name) of a user with this function.

NOTE Does not handle roles, email or disabled state.

PUT /<database_name>/_user/<name>

Returns

***AdminClient.CREATED* if a new user was created (matches 201),**
AdminClient.UPDATED if an existing user was updated,

Return type int

Raises

- `DoesNotExist` – When Database does not exist.
- `SyncGatewayClientErrorResponse` – When sync gateway returns a client error HTTP code.

delete()
Delete User from Database.

Returns User was deleted.

Return type bool

Raises `DoesNotExist` – User can't be found in this Database or Database does not exist.

retrieve()
Get User's info from sync gateway.

When User is found the response's payload is kept in the *data* attribute.

When User is not found (404), any existing *data* is erased. Sync gateway response can optionally contain an *admin_channels* field. When none are returned then *admin_channels* attribute of the User is set to the empty tuple.

GET /<database_name>/_user/<name>

Returns Retrieval was successful.

Return type bool

Raises `DoesNotExist` – User can't be found in this Database or Database does not exist.

set_admin_channels(**channels*)
Validate each channel passed in *channels* and save to *admin_channels* attribute. No update to channels is made if one is bad, all are rejected.

Parameters **`*args`** – Variable number of channels as str.

Raises `InvalidChannelName` – Bad channel name is received.

set_password(*password*)

Parameters **`password`** (*`str`*) – A password for the User. Must not be empty (this is a basic security measure and is not enforced by sync gateway.

Raises `InvalidPassword` – Password provided for User was invalid.

pysyncgateway.user_client module

class `pysyncgateway.user_client.`**`UserClient`**(*url*)

Bases: `pysyncgateway.client.Client`

`auth`(*username, password*)

Authorise client with provided credentials. Does not check with Sync Gateway that credentials are correct until a request is made.

Parameters

- **`username`** (*`str`*) – User name.
- **`password`** (*`str`*) – Password.

Returns None

Module contents

class `pysyncgateway.`**`AdminClient`**(*url*)

Bases: `pysyncgateway.client.Client`

Sync Gateway admin client for performing actions on the private admin API.

`url`

str – Sync Gateway admin REST API URL.

`all_databases`()

Provide all Databases on the server.

`GET /_all_dbs`

Returns All databases found, connected with this client.

Return type list (*Database*)

Raises `GatewayDown` – When sync gateway instance can not be reached by client.

class `pysyncgateway.`**`Database`**(*client, name*)

Bases: `pysyncgateway.helpers.ComparableMixin`, `object`

A Database on Sync Gateway.

`client`

AdminClient

`name`

str

`url`

str – URL to the database, created at init time, including trailing slash.

`all_docs`()

Get list of all Documents in database.

`GET /:name/_all_docs`

Warning: Use for testing only. From Simon @ Couchbase:

> We would strongly advise against using the `_all_docs` endpoint. As your database grows relying on the View that this calls to return to you every document key is inadvisable and does not scale well to very high numbers of documents.
>
> If you need to retrieve or update multiple documents please use the `_bulk_get` and `_bulk_docs` end points to supply a list of keys (or documents) for retrieval or update.

Returns An instance of Document for each document returned by the endpoint. For each instance the `data['_rev']` value is populated with the revision ID from `value.rev`.

Return type list (*Document*)

Raises *DoesNotExist* – Database can't be found on Sync Gateway.

all_users()

`GET /:name/_user/`

Returns All Users in Database.

Return type list (*User*)

bulk_docs(*docs*, *new_edits=False*)

Update multiple documents.

`POST /:name/_bulk_docs`

Parameters

- **docs** (*list (Document)*) – Documents to be created.
- **new_edits** (*bool, Optional*) – Value for the `new_edits` value passed in the POST data. When deleting open revisions, this should be set to `None` so that no `new_edits` value is sent in the POST data - this is required for the deletion to be successful. Defaults to `False`.

Returns Bulk document update was accepted.

Return type bool

Raises *DoesNotExist* – Database can't be found on Sync Gateway.

create()

Write this Database instance to Sync Gateway.

Uses test orientated settings (i.e. none - the empty dictionary `{}` is passed as data) to create database. This function is intended for test functionality, rather than for clients to be regularly creating databases.

`PUT /:name/`

Returns Creation was successful.

Return type bool

delete()

Remove database.

Whereas Sync Gateway will raise 404 if the database is not found, this fails silently with the intention that it can be used 'scatter gun' style at the end of test runs to clean up database lists.

`DELETE /:name/`

Returns Database was found and deleted.

Return type bool

get()

Return information about this Database from Sync Gateway.

`GET /:name/`

Returns Information loaded from SG.

Return type dict

Raises

- *DoesNotExist* – When database is not written to Sync Gateway regardless of whether the client is authorized or not.
- *ClientUnauthorized* – When database exists and client is not authorized.

get_document(*doc_id*)

Returns An instance of Document in this Database with provided `doc_id`.

Return type *Document*

get_query(*doc_id*)

Returns An instance of a query design document in this Database with the provided `doc_id`.

Return type *Query*

get_session()

Returns Session

get_user(*username*)

Returns An instance of User for the provided `username`.

Return type *User*

class pysyncgateway.**Document**(*database*, *doc_id*)

Bases: `pysyncgateway.resource.Resource`

A Couchbase Document in a database.

channels

tuple (str) – Channels this document is in.

data

DataDict – Data from the Document using the *DataDict* manager. The *DataDict* instance prevents protected keys from entering the data, but does nothing to prevent mutation. Therefore it never contains the private SG fields '_id', '_rev', 'channels'.

doc_id

str – ID of document.

rev

str – Revision identifier of document. Set to empty string when no document has been retrieved.

to_delete

bool – Flag used by `Document.flatten_data()`. When set it generates data used to delete the Document when posted with `Database.bulk_docs()`.

open_revisions

list (Document) – List of previous revisions as Document instances. This will be populated when `Document.retrieve()` is called with `revs=True`.

url

str – URL for this resource on Sync Gateway.

create_update()

Save or update Document in Sync Gateway. Saves the received revision id into instance's `rev` attribute.

`PUT /<database_name>/<doc_id>`

Note: Works for updates but is not tested.

Returns `AdminClient.CREATED` if document was created (matches 201).

Return type int

Raises *`RevisionMismatch`* – When create (no revision) is tried on an existing Document or update is tried on an existing document, but the revision numbers do not match. Two args are passed to the exception: url of the document and any revision that was passed with the `PUT` request.

delete()

Delete Document from its Database. Document must have been retrieved in order for a valid revision ID to be provided. If there isn't a cache of this information when a *delete* is asked for, then a pre-fetch will occur.

Uses the default *`Client.delete()`* action, but then inspects the response to ensure that `{"ok": true}`.

`DELETE /<name>/<doc_id>?rev=<rev>`

Returns Delete was successful.

Return type bool

Raises

- *`DoesNotExist`* – If Document can't be found (doc has to be loaded first to retrieve the revision number, which can yield a 404 if it doesn't exist). Also can be raised if the Database does not exist.
- *`RevisionMismatch`* – Provided `rev` parameter did not match the live revision ID on Sync Gateway at request time.

flatten_data()

Used when posting multiple documents with *`Database.bulk_docs()`* to the `/_bulk_docs` endpoint. Set the `to_delete` flag on the Document instance if it should be deleted - only the particular revision will be marked for deletion, not all open revisions.

Returns Data for this document including `_rev` and `_id`.

Return type dict

Raises `ValueError` – Document needs a rev to be deletable.

get_open_revisions(*include_deleted=False*)

Retrieve all leaf revisions of this document and update this instance with the currently winning revision's data.

Sync Gateway provides the leaf revision documents "as is" with no flag that one or other is the current document. Therefore this function makes two requests:

- A GET request using *`Document.retrieve()`*. This is used to find the currently winning revision.
- A GET request with `?open_revs=all` to collect all leaf nodes.

The list of leaf nodes is iterated and the currently winning revision is used to update this instance. Losing leaf revisions are blown up into `Document` instances and stored in the `open_revisions` list.

Parameters **`include_deleted`** (*`bool, Optional`*) – When set to `True`, Documents found that are not currently the winning revision and contain `{'_deleted': True}` are ignored. Defaults to `False`.

Returns Number of open revisions found including the current revision.

Return type int

Raises `RevisionMismatch` – When winning revision's rev was not found when the open revisions were loaded. This usually means that the winning revision was deleted before the open revisions could be retrieved. (untested and needs improvement to avoid race condition.)

`retrieve``()`

Load document contents. Once loaded, `_rev` and `channels` are used to update the internal attributes before the data is sent to the DataDict.

`GET /<name>/<doc_id>`

Returns Load was successful.

Return type bool

Raises *`DoesNotExist`* – Document with provided doc_id can not be loaded.

Note: DataDict never contains the private Sync Gateway fields `_id`, `_rev`, `channels`.

Note: **Not** running through `/<name>/_raw/<doc_id>`.

Warning: Side effect: `self.channels` is updated based on returned the JSON using `self.set_channels()`.

Warning: Side effect: `self.data` is updated with data dictionary loaded from JSON.

Warning: Side effect: `self.rev` is updated from revision passed in JSON using `self.set_rev`.

`set_channels``(`**channels*`)`

Validate each channel passed and save to channels attribute. No update to channels is made if one is bad, all are rejected.

Parameters **`*args`** (*`str`*) – Channels to be set for this document.

Returns None

Raises *`InvalidChannelName`* – When a bad channel name is passed.

`set_rev``(`*revision_id*`)`

Set Document's revision id. This is used for subsequent updates and deletions. Not checked for any validity.

Parameters **`revision_id`** (*`str`*) –

Returns None

Raises `ValueError` – When revision_id is empty string.

class `pysyncgateway.`**`Query`**(*database*, *doc_id*)

Bases: *`pysyncgateway.resource.Resource`*

Query a design document.

`data`

DataDict – Data from the design document using the `DataDict` manager.

`doc_id`

str – ID of design document.

`url`

str – URL for this resource on Sync Gateway.

`build_view_url`(*view_name*)

Parameters **`view_name`** (*`str`*) –

Returns URL for querying view.

Return type str

`create_update`()

Create or update design document with data.

`PUT /<database_name>/_design/<doc_id>`

Returns Design document was created or updated successfully.

Return type bool

`delete`()

Delete design document.

Returns Design document deleted.

Return type bool

Raises `DoesNotExist` – Design document or Database can not be found.

`query_view`(*view_name*, *key=None*, *stale=True*, *timeout=None*)

Load a view function from this design Document. Document must be written to Sync Gateway and view's Javascript function must be valid.

Parameters

- **`view_name`** (*`str`*) – View's name.
- **`key`** (*`Optional`*) – Value to use to search the view's key (the left part of the map). Any type can be passed as long as:
  - `key is not None`
  - `key` can be serialized to a string by `json.dumps()`.

  To query a view with multiple keys set `key` as an iterable which will be serialized to JSON as an array. E.g. `key=['left_id', 'right_id']`.
- **`stale`** (*`bool, Optional`*) – Allow stale results in the view. This is currently the default value in Sync Gateway, so is only passed when set to `False`. Default `True`.

  `'false'` is an undocumented option for this param. See https://github.com/couchbase/sync_gateway/issues/727#issuecomment-83588984

It also doesn't work in Walrus mode, see https://github.com/constructpm/pysyncgateway/issues/7

- **timeout** (*int, Optional*) – Set a time out of seconds as per requests' spec which means that if the Sync Gateway does not respond to the GET request within the `timeout` period, a `ReadTimeout` will be raised. Default `None` which means that requests' default is used.

Returns Decoded JSON for view data result. Will usually be a dictionary contain keys 'Collator', 'rows' and 'total_rows'.

Return type dict

Raises

- `DoesNotExist` – When database, design document or contained view can not be found.
- `requests.exceptions.ReadTimeout` – When Sync Gateway does not respond within given `timeout`.

retrieve()

Returns Design document was retrieved.

Return type bool

Raises `DoesNotExist` – Design document or Database can not be found.

Side effects: data: Updates internal data dictionary with data loaded from JSON.

class pysyncgateway.**Session**(*database*)

Bases: *pysyncgateway.resource.Resource*

Sessions on the Database are undocumented as of Sync Gateway 1.5 in both the public and admin REST APIs. But since they provide a useful test mechanism by showing a list of channels that the authenticated User has been subscribed to, they are included in this library.

data

DataDict – Data from the session using the `DataDict` manager.

database

Database

url

str – URL for the session in the Database on Sync Gateway.

get_channels()

Helper to pick user's subscribed channels from retrieved session data.

Returns Sorted list of channel names found in response including the special `!` public channel.

Return type list

Raises

- `NotLoaded` – When session has not been retrieved.
- `KeyError` – When the `userCtx` data in the session response does not contain a `channels` dictionary.

get_name()

Helper to pick user's `name` field from retrieved session data.

Returns Name of user according to Sync Gateway.

Return type str

Raises

- `NotLoaded` – When session has not been retrieved.
- `KeyError` – When the `userCtx` data in the session response does not contain a `name` field.

retrieve()

Collect session information for the current client in the `database` attribute.

For unauthorized users on the public API and requests on the admin API this looks like:

```
{
    'authentication_handlers': ['default', 'cookie'],
    'ok': True,
    'userCtx': {
        'channels': {},
        'name': None,
    },
}
```

For authenticated users on the public API, in this example with the name `__USERNAME__` and channels `a`, `b` and `c`, this looks like:

```
{
    'authentication_handlers': ['default', 'cookie'],
    'ok': True,
    'userCtx': {
        'channels': {
            '!': 1,
            'a': 1,
            'b': 1,
            'c': 1,
        },
        'name': '__USERNAME__',
    },
}
```

Returns Success

Return type bool

Raises `DoesNotExist` – When database does not exist on Sync Gateway, regardless of whether client is authorized or not.

Warning: Side effect: Updates `self.data` with response from Sync Gateway on success.

class pysyncgateway.**Stats**(*client*)

Bases: `object`

Stats object from the expvars endpoint on the Sync Gateway. See https://github.com/couchbase/sync_gateway/wiki/expvars

NOTE: Stats come from the server and not from the database.

client

AdminClient – Used to communicate with the server.

data
dict – Statistical data populated after retrieval. Will be empty dictionary before retrieval.

url
str – Location of stats.

retrieve()
Load stats, parse info and stash in data attr.

Returns load was successful.

Return type bool

Raises

- `GatewayDown` – When the endpoint can't be reached.
- `ValueError` – When something non-JSON is loaded.

Side effects: data: Populated with dictionary parsed from the JSON response.

class pysyncgateway.**User**(*database*, *name*)
Bases: `pysyncgateway.resource.Resource`

A User in a Database.

admin_channels
tuple(str) – Channels that this User has been added to by admin or sync function.

data
DataDict

name
str – Username of this user.

password
str – User's password. Can be *None* and is only set with *set_password*.

retrieved
bool – User been retrieved from sync gateway. Acts as a flag to know if there should be a password sent at *create_update* time.

url
str – URL for this User on sync gateway.

create_update()
Create a new or update an existing user for currently connected database. When creating a new User then their password must have been set on this instance using *set_password*, or the sync gateway will reject the PUT with a 400.

NOTE it's not possible to change the username (name) of a user with this function.

NOTE Does not handle roles, email or disabled state.

PUT /<database_name>/_user/<name>

Returns

***AdminClient.CREATED* if a new user was created (matches 201),**
AdminClient.UPDATED if an existing user was updated,

Return type int

Raises

- `DoesNotExist` – When Database does not exist.
- `SyncGatewayClientErrorResponse` – When sync gateway returns a client error HTTP code.

delete()

Delete User from Database.

Returns User was deleted.

Return type bool

Raises `DoesNotExist` – User can't be found in this Database or Database does not exist.

retrieve()

Get User's info from sync gateway.

When User is found the response's payload is kept in the *data* attribute.

When User is not found (404), any existing *data* is erased. Sync gateway response can optionally contain an *admin_channels* field. When none are returned then *admin_channels* attribute of the User is set to the empty tuple.

GET /<database_name>/_user/<name>

Returns Retrieval was successful.

Return type bool

Raises `DoesNotExist` – User can't be found in this Database or Database does not exist.

set_admin_channels(**channels*)

Validate each channel passed in *channels* and save to *admin_channels* attribute. No update to channels is made if one is bad, all are rejected.

Parameters **`*args`** – Variable number of channels as str.

Raises `InvalidChannelName` – Bad channel name is received.

set_password(*password*)

Parameters **`password`** (`str`) – A password for the User. Must not be empty (this is a basic security measure and is not enforced by sync gateway.

Raises `InvalidPassword` – Password provided for User was invalid.

class pysyncgateway.**UserClient**(*url*)

Bases: `pysyncgateway.client.Client`

auth(*username*, *password*)

Authorise client with provided credentials. Does not check with Sync Gateway that credentials are correct until a request is made.

Parameters

- **`username`** (`str`) – User name.
- **`password`** (`str`) – Password.

Returns None

1.1.3 Release checklist

Items to be completed for each release. Given a new version called `x.y.z`:

- Create a branch for the new release. Usually called something like `bump-vx.y.z`.

- Update `__version__` in __about__.py with the new version number `'x.y.z'`.
- Update CHANGELOG.
  - Add a new subtitle below `Unreleased` after the note about latest documentation, in the format `x.y.z_ - yyyy/mm/dd`, where `yyyy/mm/dd` is the reverse formatted date of the day the release is created.
  - Update the `.. _Unreleased:` link at the bottom of the page to compare `vx.y.z...HEAD`.
  - Under the `_Unreleased` link, create a new link for the release `.. _x.y.z: https:/[...]/compare/va.b.c...vx.y.z`, where `va.b.c` is the previous release.
- Commit changes and push `bump-vx.y.z` branch for testing.
- Now is a good time to build and check the documentation locally.
- When branch `bump-vx.y.z` is green, then merge it to `master`.
- Update master locally and ensure that you remain on master for the rest of the process.
- Test that a build can be shipped to test PyPI with `make testpypi`. (Every build runs the full clean test suite locally to ensure that nothing has broken before building)
- After successful push, check the TestPyPI page.
- Then tag the repo with `make tag`. Add a short message about what the key change is.
- Make the new tag public with `git push origin --tags`.
- Build and push to PyPI with `make pypi`.
- After successful push, check the PyPI page.

Post release checks

- Visit the CHANGELOG and ensure that the new release's comparison link works with the new tag.
- Check the RTD builds to ensure that the latest documentation version has been picked up and that the `stable` docs are pointed at it.

  A new docs release will not have been created for the new tag as per this issue. Click "Build Version:" on the builds page for the new tag to be picked up.

Python Module Index

Index