SNSAPI Documentation

Release 0.6.2

uxian, hupili

July 21, 2013

CONTENTS

Contents:

CHAPTER

ONE

SNSAPI MIDDLEWARE

1.1 snsbase.py

snsapi base class.

All plugins are derived from this class. It provides common authenticate and communicate methods.

snsapi.snsbase.**require_authed**(*func*)
A decorator to require auth before an operation

class snsapi.snsbase.**SNSBase**(*channel=None*)

fetch_code()

request_url(*url*)

_oauth2_first()
The first stage of oauth. Generate auth url and request.

_oauth2_second()
The second stage of oauth. Fetch authenticated code.

oauth2()
Authorizing using synchronized invocation of OAuth2.

Users need to collect the code in the browser's address bar to this client. callback_url MUST be the same one you set when you apply for an app in openSNS platform.

auth()
General entry for authorization. It uses OAuth2 by default.

auth_first()

auth_second()

open_brower(*url*)

_parse_code(*url*)
Parse code from a URL containing `code=xx` parameter

Parameters **url** – contain code and optionally other parameters

Returns JsonDict containing 'code' and **(optional) other URL parameters**

_token_filename()

save_token()
access token can be saved, it stays valid for a couple of days if successfully saved, invoke get_saved_token() to get it back

get_saved_token()

expire_after(*token=None*)
Calculate how long it is before token expire.

Returns

- >0: the time in seconds.
- 0: has already expired.
- -1: there is no token expire issue for this platform.

is_expired(*token=None*)
Check if the access token is expired.

It delegates the logic to 'expire_after', which is a more formal module to use. This interface is kept for backward compatibility.

is_authed()

need_auth()
Whether this platform requires two-stage authorization.

Note:

•Some platforms have authorization flow but we do not use it, e.g. Twitter, where we have a permanent key for developer They'll return False.

•If your platform do need authorization, please override this method in your subclass.

static **new_channel**(*full=False*)
Return a JsonDict object containing channel configurations.

Parameters **full** – Whether to return all config fields.

- False: only returns essential fields.
- True: returns all fields (essential + optional).

read_channel(*channel*)

setup_oauth_key(*app_key*, *app_secret*)
If you do not want to use read_channel, and want to set app_key on your own, here it is.

_http_get(*baseurl*, *params*)
Use HTTP GET to request a JSON interface

Parameters

- **baseurl** – Base URL before parameters
- **params** – a dict of params (can be unicode)

Returns

- Success: A JSON compatible structure
- Failure: A {}. Warning is logged.

_http_post(*baseurl*, *params*)
Use HTTP POST to request a JSON interface.

See `_http_get` for more info.

_unicode_encode(*s*)
Detect if a string is unicode and encode as utf-8 if necessary

_expand_url(*url*)
expand a shorten url

Parameters **url** – The url will be expanded if it is a short url, or it will return the origin url string. url should contain the protocol like "http://"

_cat(*length*, *text_list*, *delim='||'*)
Concatenate strings.

Parameters

- **length** – The output should not exceed length unicode characters.
- **text_list** – A list of text pieces. Each element is a tuple (text, priority). The _cat function will concatenate the texts using the order in text_list. If the output exceeds length, (part of) some texts will be cut according to the priority. The lower priority one tuple is assigned, the earlier it will be cut.

forward(**al*, ***ad*)
A general forwarding implementation using update method.

Parameters

- **message** – The Message object. The message you want to forward.
- **text** – A unicode string. The comments you add to the message.

Returns Successful or not: True / False

NOTE: This method require authorization before invokation.

_SNSBase__init_oauth2_client()

1.2 snsconf.py

snsapi Basic Hardcode Conf

See documentations of variables for more information.

For first time users, please ignore the following discussion in the same section. They are intended for SNSAPI middleware developers. I don't want to confuse you at the moement. When you are ready to refactor this piece of code, you can come back to read them discuss in the group.

This files may look weird at first glance, here's a short background story on how I get to this point:

- There are many debugging information printed on the console previously, which make stdin/stdout interface a mess.
- I just developed a wrapper for logging. Hope it can unify different log messages.
- 'snsapi' as a whole package will import all plugins at the initialization stage. This will trigger a 'xxx plugged!" message.
- Some calls to write logs happens before we have a chance to init SNSLog (Original plan is to let the upper layer init with its own preference).
- The workaround is to develop this hardcode conf files.

Guidelines to add things here:

- If something is to be configured before fully init of snsapi(which involves init those plugins), the configuration can go into this file.
- Otherwise, try best to let the upper layer configure it. Put the confs in the `../conf` folder.

class snsapi.snsconf.**SNSConf**
Hardcode Confs for SNSAPI

SNSAPI_CONSOLE_STDIN_ENCODING = 'utf-8'
For chinese version windows systems, you may want to change `SNSAPI_CONSOLE_STDOUT_ENCODING = 'utf-8'` and `SNSAPI_CONSOLE_STDIN_ENCODING = 'utf-8'` to 'gbk'. For others, check the encoding of your console and set it accordingly.

See the discussion: https://github.com/hupili/snsapi/issues/8

SNSAPI_CONSOLE_STDOUT_ENCODING = 'utf-8'
See `SNSAPI_CONSOLE_STDIN_ENCODING`.

SNSAPI_DIR_STATIC_DATA = '/home/docs/checkouts/readthedocs.org/user_builds/snsapi/envs/v0.6.2/local/lib/python2.

SNSAPI_DIR_STORAGE_CONF = '/home/docs/.snsapi/conf'

SNSAPI_DIR_STORAGE_ROOT = '/home/docs/.snsapi'

SNSAPI_DIR_STORAGE_SAVE = '/home/docs/.snsapi/save'

SNSAPI_LOG_INIT_LEVEL = 20

Possible values:

- SNSLog.DEBUG
- SNSLog.INFO
- SNSLog.WARNING
- SNSLog.ERROR
- SNSLog.CRITICAL

In Release version, set to WARNING

SNSAPI_LOG_INIT_LOGFILE = None

•None: Output to STDOUT. Good for Debug version.

•{Filename}: Log to {Filename}. Good for Relase version.

SNSAPI_LOG_INIT_VERBOSE = False
Examples,

True:

- [DEBUG][20120829-135506][sina.py][<module>][14]SinaAPI plugged!

False:

- [DEBUG][20120829-142322]SinaAPI plugged!

os = <module 'os' from '/home/docs/checkouts/readthedocs.org/user_builds/snsapi/envs/v0.6.2/lib/python2.7/os.pyc'>

exception snsapi.snsconf.**SNSConfNoInstantiation**
This exception is used to make sure you do not instantiate SNSConf class.

1.3 snscrypt.py

snsapi Cryptography Tools

class snsapi.snscrypt.**SNSCrypto**
snsapi cryptography tools

decrypt (*string*)
INPUT: hexadecimal string of encrypt output OUTPUT: any string

Reverse process of decrypt

encrypt (*string*)
INPUT: any string OUTPUT: hexadecimal string of encrypt output

The input string will first be encoded by BASE-64. This is to make sure the intermediate value is a printable string. It will be easier to change pyDes to other crypto utilities. After actual encryption delegated to other modules, the final output is also base64 encrypted, this makes it printable.

1.4 snslog.py

snsapi Log Tools

class snsapi.snslog.**SNSLog**
Provide the unified entry to write logs

The current backend is Python's logging module.

CRITICAL = 50

DEBUG = 10

ERROR = 40

INFO = 20

VERBOSE = False

WARNING = 30

static critical (*fmt*, **args*)

static debug (*fmt*, **args*)

static error (*fmt*, **args*)

static info (*fmt*, **args*)

static init (*logfile=None*, *level=30*, *verbose=True*)
Init the log basic configurations. It should be called only once over the entire execution.

If you invoke it for multiple times, only the first one effects. This is the behaviour of logging module.

static warn (*fmt*, **args*)

static warning (*fmt*, **args*)

exception snsapi.snslog.**SNSLogNoInstantiation**
docstring for SNSLogNoInstantiation

1.5 snstype.py

SNS type: status, user, comment

class snsapi.snstype.**AuthenticationInfo**(*auth_info=None*)

set_defaults()

class snsapi.snstype.**Message**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)
The Message base class for SNSAPI

Data Fields:

- platform: a string describing the platform where this message come from. See 'snsapi/platform.py' for more details.
- raw: the raw json or XML object returned from the platform spefiic API. This member is here to give upper layer developers the last chance of manipulating any available information. Having an understanding of the platform-specific returning format is esential.
- parsed: this member abstracts some common fields that all messages are supposed to have. e.g. 'username', 'time', 'text', etc.
- ID: a MessageID object. This ID should be enough to indentify a message across all different platforms.

For details of ID, please see the docstring of MessageID.

Mandatory fields of parsed are:

- time: a utc integer. (some platform returns parsed string)
- userid: a string. (called as "username" at some platform)
- username: a string. (called as "usernick" as some platform)
- text: a string. (can be 'text' in the returning json object, or parsed from other fields.)

Optional fields of 'parsed' are:

- deleted: Bool. For some OSN.
- reposts_count: an integer. For some OSN.
- comments_count: an integer. For some OSN.
- link: a string. For RSS; Parsed from microblog message; Parsed from email message; etc.
- title: a string. For RSS; Blog channel of some OSN.
- description: a string. For RSS digest text; Sharing channel of some OSN; etc.
- body: a string. The 'content' of RSS, the 'body' of HTML, or whatever sematically meaning the body of a document.
- text_orig: a string. The original text, also known as "root message" in some context. e.g. the earliest status in one thread.
- text_last: a string. The latest text, also known as "message" in some context. e.g. the reply or forwarding comments made by the last user.
- text_trace: a string. Using any (can be platform-specific) method to construt the trace of this message. e.g. the forwarding / retweeting / reposting sequence. There is no unified format yet.
- username_origin: a string. The username who posts 'text_orig'.

- `attachments`: an array of attachments, for one attachment, it

should be a dict like {'type': TYPE, 'format': [FORMAT], data': DATA}, and TYPE can be one of link, picture, album, video, blog, and FORMAT can be any of link, binary, and others and DATA is your data.

digest()
Digest the message content. This value is useful in for example forwarding services to auto-reply services, for those applications requires message deduplication.

It corresponds to dump().

Note: different messages may be regarded as the same according to this digest function.

digest_full()
It corresponds to dump_full()

digest_parsed()
It corresponds to dump_parsed()

dump()
Level 1 serialization: console output.

This level targets console output. It only digests essnetial information which end users can understand. e.g. the text of a status is digested whereas the ID fields is not digested.

To control the format, please rewrite dump() in derived Message class.

See also __str__(), __unicode__(), show()

dump_full()
Level 3 serialization: complete output.

This level targets more sophisticated applications. The basic function of SNSAPI is to unify different formats. That's what the first two level of serialization do. However, app developers may want more sophisticated processing. We serialize the full Message object through this function. In this way, app developers can get all information they need. Note that knowledge of the platform specific return format is essential. We conclude their fields in:

- https://github.com/hupili/snsapi/wiki/Status-Attributes

This wiki page may not always be up to date. Please refer to the offical API webpage for more info.

dump_parsed()
Level 2 serialization: interface output.

This level targets both Python class interface and STDIO/STDOUT interface. The output of all kinds of Messages conform to the same format. The json object can be used to pass information in/out SNSAPI using Python class. It is also able to pretty print, so that the STDOUT result is easy to parse in any language.

parse()
Parse self.raw and store result in self.parsed

platform = 'SNSAPI'

show()
Level 1 serialization and print to console

See dump()

class snsapi.snstype.**MessageID**(*platform=None*, *channel=None*)
All information to locate one status is here.

It shuold be complete so that:

- one can invoke reply() function of plugin on this object.
- Or one can invoke reply() function of container on this object.

There are two mandatory fields:

- platform: Name of the platform (e.g. RenrenStatus)
- channel: Name of the instantiated channel (e.g. 'renren_account_1'). Same as a channel's `.jsonconf['channel_name']`.

In order to reply one status, here's the information required by each platforms:

- Renren: the status_id and source_user_id
- Sina: status_id
- QQ: status_id

NOTE: This object is mainly for SNSAPI to identify a Message. Upper layer had better not to reference fields of this object directly. If you must reference this object, please do not touch those non-mandatory fields.

class `snsapi.snstype.`**`MessageList`**(*init_list=None*)
A list of Message object

`append`(*e*)

`extend`(*l*)

class `snsapi.snstype.`**`User`**(*jobj=None*)

1.6 utils.py

class `snsapi.utils.`**`FixedOffsetTimeZone`**(*offset*, *name*)
Fixed offset in minutes east from UTC.

See `third/timezone_sample.py` for more samples.

`dst`(*dt*)

`tzname`(*dt*)

`utcoffset`(*dt*)

class `snsapi.utils.`**`JsonDict`**(*jsonconf=None*)
The wrapper class for Python dict.

It is intended to host Json compatible objects. In the interative CLI, the users are expected to configure SNSAPI during execution. To present the current config in a nice way. We should add indentation for the dump method.

`get`(*attr*, *default_value=None*)
dict entry reading with fault tolerance.

Attr A str or a list of str.

Returns The value corresponding to the (first) key, or default val.

If attr is a list, we will try all the candidates until one 'get' is successful. If none of the candidates succeed, return a the `default_value`.

e.g. RSS format is very diverse. To my current knowledge, some formats have 'author' fields, but others do not:

- rss : no
- rss2 : yes
- atom : yes
- rdf : yes

NOTE:

- The original `default_value` is "(null)". Now we change

to `None`. `None` is more standard in Python and it does not have problem to convert to `str` (the usual way of using our data fields). It has the JSON counterpart: `null`.

class `snsapi.utils.`**`JsonObject`**
general json object that can bind any fields but also act as a dict.

class `snsapi.utils.`**`Serialize`**(**al*, ***ad*)
The common serialization SAP

static `dumps`(*obj*)

static `loads`(*string*)

`snsapi.utils.`**`console_input`**(*string=None*)
To make oauth2 testable, and more reusable, we use console_input to wrap raw_input. See http://stackoverflow.com/questions/2617057/supply-inputs-to-python-unittests.

`snsapi.utils.`**`console_output`**(*string*)
The sister function of console_input()!

Actually it has a much longer story. See Issue#8: the discussion of console encoding~

`snsapi.utils.`**`html_entity_unescape`**(*s*)
Escape HTML entities, such as "£" This interface always returns unicode no matter the input 's' is str or unicode.

`snsapi.utils.`**`obj2str`**(*obj*)
Convert Python object to string using SNSApi convention.

Parameters **obj** – A Python object that is "serializable". Check `Serialize` to what backend we use for serialization.

`snsapi.utils.`**`report_time`**(*func*)

`snsapi.utils.`**`str2obj`**(*string*)
Convert string to Python object using SNSApi convention.

Parameters **obj** – A string constructed by `obj2str`. Do not call this method on a string coming from an unkown source.

`snsapi.utils.`**`str2utc`**(*s*, *tc=None*)

Parameters **tc** – Timezone Correction (TC). A timezone suffix string. e.g. `" +08:00"`, '' HKT'', etc. Some platforms are know to return time string without TZ (e.g. Renren). Manually do the correction.

`snsapi.utils.`**`strip_html`**(*text*)

`snsapi.utils.`**`utc2str`**(*u*)

1.7 errors.py

Errors or Exceptions for SNSAPI

How to add an error type?

- Check out `Mark1` (in `errors.py`). Inherit your error type from a right class. The `#>>` comment denotes the level an error type is in.
- Check out `Makr2` (in `errors.py`). Add your new error type to the corresponding tree.

How to reference an error type?

- By convention, others should only import "snserror". e.g. `from errors import snserror`, or `from snsapi import snserror`.
- Use dot expression to enter the exact type. e.g. `snserror.config.nofile`.

exception snsapi.errors.**ConfigError**

load
alias of SNSPocketLoadConfigError

nofile
alias of NoConfigFile

save
alias of SNSPocketSaveConfigError

exception snsapi.errors.**MissAPPInfo**

exception snsapi.errors.**NoConfigFile**(*fname='conf/channel.json'*)

exception snsapi.errors.**NoPlatformInfo**

exception snsapi.errors.**NoSuchChannel**

exception snsapi.errors.**NoSuchPlatform**

exception snsapi.errors.**SNSAuthFail**

fetchcode
alias of SNSAuthFechCodeError

exception snsapi.errors.**SNSAuthFechCodeError**

exception snsapi.errors.**SNSEncodingError**

exception snsapi.errors.**SNSError**

exception snsapi.errors.**SNSOperation**

read
alias of SNSReadFail

write
alias of SNSWriteFail

exception snsapi.errors.**SNSPocketDuplicateName**(*cname*)

exception snsapi.errors.**SNSPocketError**

exception snsapi.errors.**SNSPocketLoadConfigError**(*msg=''*)

exception snsapi.errors.**SNSPocketSaveConfigError**

exception snsapi.errors.**SNSReadFail**

exception snsapi.errors.**SNSTypeError**(*value=''*)

parse
alias of SNSTypeParseError

exception snsapi.errors.**SNSTypeParseError**(*value=''*)

exception snsapi.errors.**SNSTypeWrongInput**(*value=''*)

exception snsapi.errors.**SNSWriteFail**(*value*)

class snsapi.errors.**snserror**

auth
alias of SNSAuthFail

config
alias of ConfigError

op
alias of SNSOperation

type
alias of SNSTypeError

1.8 platform.py

Platforms.

Upper layer can reference platforms from this module

1.9 snspocket.py

snspocket: the container class for snsapi's

class snsapi.snspocket.**SNSPocket**
The container class for snsapi's

add_channel(*jsonconf*)

auth(*channel=None*)
docstring for auth

clear_channel()

forward(*message*, *text*, *channel=None*)
forward a message

home_timeline(*count=None*, *channel=None*)
Route to home_timeline method of snsapi.

Parameters **channel** – The channel name. Use None to read all channels

list_channel(*channel=None*, *verbose=False*)

list_platform()

load_config(*fn_channel='/home/docs/.snsapi/conf/channel.json'*, *fn_pocket='/home/docs/.snsapi/conf/pocket.json'*)
Read configs: * channel.conf * pocket.conf

new_channel(*pl=None*, ***kwarg*)

reply(*message*, *text*, *channel=None*)
Route to reply method of snsapi.

Parameters

- **channel** – The channel name. Use None to automatically select one compatible channel.
- **status** – Message or MessageID object.

Text Reply text.

save_config(*fn_channel='/home/docs/.snsapi/conf/channel.json'*, *fn_pocket='/home/docs/.snsapi/conf/pocket.json'*)
Save configs: reverse of load_config

Configs can be modified during execution. snsapi components communicate with upper layer using Python objects. Pocket will be the unified place to handle file transactions.

update(*text*, *channel=None*)
Route to update method of snsapi.

Parameters **channel** – The channel name. Use None to update all channels

CHAPTER

TWO

SNSAPI PLUGGINS (SPECIFICS FOR DIFFERENT PLATFORM)

2.1 Sina

Sina Weibo client

class snsapi.plugin.sina.**SinaWeiboBase**(*channel=None*)

static new_channel(*full=False*)

read_channel(*channel*)

need_auth()

auth_first()

auth_second()

weibo_request(**al*, ***ad*)

General request method for Weibo V2 Api via OAuth.

Parameters

- **name** – The Api name shown on main page (http://open.weibo.com/wiki/API%E6%96%87%E6%A1%A3_V2). e.g. `friendships/create` (no "2/" prefix)
- **method** – HTTP request method: 'GET' or 'POST'
- **params** – Parameters from Api doc. No need to manually put `access_token` in.

Returns The http response from SinaWeibo (a JSON compatible structure).

NOTE: This method require authorization before invokation.

class snsapi.plugin.sina.**SinaWeiboStatusMessage**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

platform = 'SinaWeiboStatus'

parse()

class snsapi.plugin.sina.**SinaWeiboStatus**(*channel=None*)

Message

alias of SinaWeiboStatusMessage

home_timeline(**al*, ***ad*)
Get home timeline

Parameters **count** – number of statuses

NOTE: This method require authorization before invokation.

update(**al*, ***ad*)
update a status

•parameter text: the update message

•return: success or not

NOTE: This method require authorization before invokation.

reply(**al*, ***ad*)
reply to a status

•parameter text: the comment text

•return: success or not

NOTE: This method require authorization before invokation.

forward(**al*, ***ad*)
Forward a status on SinaWeibo:

•If message is from the same platform, forward it using special interface.

•Else, route the request to a general forward method of `SNSBase`.

•Decorate the text with previous comment sequence.

Parameters

- **message** – An `snstype.Message` object to forward
- **text** – Append comment text

Returns Success or not

NOTE: This method require authorization before invokation.

2.2 Tencent

Tencent Weibo Client

class `snsapi.plugin.tencent.`**TencentWeiboStatusMessage**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

platform = 'TencentWeiboStatus'

parse()

class `snsapi.plugin.tencent.`**TencentWeiboStatus**(*channel=None*)

Message
alias of `TencentWeiboStatusMessage`

static new_channel(*full=False*)

read_channel(*channel*)

need_auth()

auth_first()

auth_second()

home_timeline(**al*, ***ad*)
Get home timeline

- function : get statuses of yours and your friends'
- parameter count: number of statuses

NOTE: This method require authorization before invokation.

update(**al*, ***ad*)
update a status

- parameter text: the update message
- return: success or not

NOTE: This method require authorization before invokation.

reply(**al*, ***ad*)
reply to a status

- parameter text: the comment text
- return: success or not

NOTE: This method require authorization before invokation.

2.3 Renren

Renren Client

Codes are adapted from following sources:

- http://wiki.dev.renren.com/mediawiki/images/4/4c/Renren-oauth-web-demo-python-v1.0.rar

exception snsapi.plugin.renren.**RenrenAPIError**(*code*, *message*)

class snsapi.plugin.renren.**RenrenBase**(*channel=None*)

static new_channel(*full=False*)
docstring placeholder

read_channel(*channel*)
docstring placeholder

need_auth()
docstring placeholder

auth_first()
docstring placeholder

auth_second()
docstring placeholder

auth()
docstring placeholder

renren_request(*params=None*)
A general purpose encapsulation of renren API. It fills in system paramters and compute the signature.

class snsapi.plugin.renren.**RenrenShareMessage**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

platform = 'RenrenShare'

parse()

class snsapi.plugin.renren.**RenrenShare**(*channel=None*)

Message
alias of RenrenShareMessage

static new_channel(*full=False*)
docstring placeholder

home_timeline(**al*, ***ad*)
Get timeline of Renren statuses

Parameters **count** – Number of statuses

Returns At most count statuses (can be less).

NOTE: This method require authorization before invokation.

reply(**al*, ***ad*)
docstring placeholder

NOTE: This method require authorization before invokation.

class snsapi.plugin.renren.**RenrenStatusMessage**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

platform = 'RenrenStatus'

parse()

class snsapi.plugin.renren.**RenrenStatus**(*channel=None*)

Message
alias of RenrenStatusMessage

static new_channel(*full=False*)

home_timeline(**al*, ***ad*)
Get home timeline get statuses of yours and your friends' @param count: number of statuses

NOTE: This method require authorization before invokation.

update(**al*, ***ad*)
update a status @param text: the update message @return: success or not

NOTE: This method require authorization before invokation.

reply(**al*, ***ad*)
reply status @param status: StatusID object @param text: string, the reply message @return: success or not

NOTE: This method require authorization before invokation.

forward(**al*, ***ad*)
Forward a status on Renren:

•If message is from the same platform, forward it using special interface.

•Else, route the request to a general forward method of `SNSBase`.

Parameters

- **message** – An `snstype.Message` object to forward
- **text** – Append comment text

Returns Success or not

NOTE: This method require authorization before invokation.

class `snsapi.plugin.renren.`**`RenrenBlogMessage`**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

`platform` = **'RenrenBlog'**

`parse`()

class `snsapi.plugin.renren.`**`RenrenBlog`**(*channel=None*)

`Message`
alias of `RenrenBlogMessage`

static `new_channel`(*full=False*)

`home_timeline`(**al*, ***ad*)
Get blog timeline

Parameters **count** – Number of blogs

NOTE: This method require authorization before invokation.

`update`(**al*, ***ad*)
Post a blog

Parameters

- **text** – Blog post body.
- **title** – Blog post title. (optional)

Returns success or not

NOTE: This method require authorization before invokation.

`reply`(**al*, ***ad*)
Reply a renren blog

Parameters

- **mID** – MessageID object
- **text** – string, the reply message

Returns success or not

NOTE: This method require authorization before invokation.

2.4 RSS

RSS Feed

Contains:

- RSS Read-only feed platform.
- RSS Read/Write platform.
- RSS Summary platform.

class snsapi.plugin.rss.**RSSMessage**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

platform = 'RSS'

parse()

dump_full()
Override Message.dump_full because default .raw attribute of RSSMessage object is not JSON serializable.

Note: dumpped messages are meant for the consumption of other languages. We do not concern how to convert back. If you do that, make sure call str2obj on .raw yourself. Besides, make sure the source of the string is trustful.

For the use in Python, directly serialize the message for persistent storage. Do not recommend you use any of the three level of dump_xxx functions. Use of digest_xxx is OK.

class snsapi.plugin.rss.**RSS**(*channel=None*)

Supported Methods

- **auth()** [] a NULL stub.
- **home_timeline()** [] read and parse RSS feed. pretend it to be a 'special' SNS platform, where you can only read your wall but can not write to it.

Message
alias of RSSMessage

static new_channel(*full=False*)

read_channel(*channel*)

auth()

auth_first()

auth_second()

home_timeline(*count=20*)
Get home timeline

•function : get statuses of yours and your friends'

•parameter count: number of statuses

expire_after(*token=None*)

class snsapi.plugin.rss.**RSS2RWMessage**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

platform = 'RSS2RW'

parse()

class snsapi.plugin.rss.**RSS2RW**(*channel=None*)
Read/Write Channel for rss2

Message
alias of RSS2RWMessage

static new_channel(*full=False*)

read_channel(*channel*)

update(*message*)

Parameters message (Message or str) – For Message update it directly. RSS2RW guarantee the message can be read back with the same values in standard fields of Message.parsed. For str, we compose the virtual Message first using current time and configured author information.

class snsapi.plugin.rss.**RSSSummaryMessage**(*dct=None*, *platform=None*, *channel=None*, *conf={}*)

platform = 'RSSSummary'

parse()

class snsapi.plugin.rss.**RSSSummary**(*channel=None*)
Summary Channel for RSS

It provides more meaningful 'text' field.

Message
alias of RSSSummaryMessage

static new_channel(*full=False*)

CHAPTER

THREE

SAMPLE APPLICATIONS WRITTEN IN SNSAPI

3.1 hellosns

3.1.1 `test_read_all.py`

3.1.2 `test_write_all.py`

3.1.3 `test_read_reply.py`

3.2 forwarder

3.3 mysofa

3.4 clock

The scripts can not be autodoc-ed due to (possibly) non-ascii characters there. We'll add the doc once the problem is solved. Please read the app README instead: https://github.com/hupili/snsapi/tree/master/app/clock

CHAPTER

FOUR

INDICES AND TABLES

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

PYTHON MODULE INDEX

S