Smarkets Python API Client Documentation

Release 9.2.0

Smarkets Limited

Mar 22, 2017

Contents

Release v9.2. (*Installation*)

The Smarkets Python API Client (smk_python_sdk) is a Python implementation of a Smarkets streaming API client.

Our intention for providing a Python reference implementation is to allow developers to quickly build applications in Python that make use of the Smarkets exchange.

CHAPTER 1

User Guide

This section provides a quick overview of the general design philosophy behind the Python API client as well as step-by-step instructions for getting started using the client.

Introduction

Philosophy

The main driving force behind the development of the Smarkets streaming API is efficiency. Many other applications in our industry rely on "polling" a service in order to provide a real-time view of changing data. While the streaming API can be used in a "synchronous" manner by making a request and waiting for a response, it is often far more efficient to design an application asynchronously. Asynchronous messaging allows for more flexible patterns and techniques like pipelining.

The streaming API also uses framed protocol buffers to define the wire format, so parsing messages is relatively fast and simple. We will endeavour to maintain backwards compatibility as we release newer revisions of the API. Protocol buffers definitions provide a decent facility for doing so.

MIT License

In selecting a software license, we aimed to choose one which allows third-party application developers flexibility in using the code we provide.

The Smarkets Python API Client is released under the MIT License.

Smarkets Python API Client License

Installation

This covers the basic installation of the Smarkets API Client which is obviously necessary to get started.

Requirements

These packages are necessary at runtime.

- *protobuf* Python package, version 2.5.0 or higher (*smk-python-sdk* package installation process will automatically install it)

These are required to build from source and run unit tests, etc.

- cURL
- mock
- piqi

Pip

Install the client with pip:

```
$ pip install smk-python-sdk
```

Updating the SDK

Minor updates (0.5.0 to 0.5.1) are backwards compatible. Major updates (for example 0.5.1 to 0.6.0) are not necessarily backwards compatible, please consult SDK *Change log*.

Github

Smarkets makes its open source development activities available on *GitHub*, and the Python API Client is available there.

Clone the public master branch:

```
$ git clone https://github.com/smarkets/smk_python_sdk.git
```

Or, download a .tar.gz:

```
$ curl -O https://github.com/smarkets/smk_python_sdk/tarball/master
```

Or, a .zip:

```
$ curl -O https://github.com/smarkets/smk_python_sdk/zipball/master
```

Installing from source

Once you have downloaded a copy of the source, you can install it into site-packages:

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

The build target will download the necessary files to generate the protobuf modules. Make sure you have satisfied the *requirements* listed above.

Quickstart

This quickstart guide assumes you have already installed the client. If you haven't, read *Installation* first.

Setting up logging

When developing, it's often useful to turn on debug logging console output:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

Starting a session

In order to do anything meaningful with the API, you must first start a new session. We import the base module and create our `SessionSettings` object:

```
>>> from smarkets.streaming_api.api import (
...     BUY, SELL, OrderCreate, Session, SessionSettings, StreamingAPIClient)
>>> username = 'username'
>>> password = 'password'
>>> settings = SessionSettings(username, password)
>>> settings.host = 'api.smarkets.com'
>>> settings.port = 3701
```

Then, we create the `Session` object which we will use to keep track of sequence numbers:

```
>>> session = Session(settings)
```

Finally, the `Client` class is the higher-level wrapper which allows us to send and handle messages:

```
>>> client = StreamingAPIClient(session)
```

Now, let's login!

```
>>> client.login()
```

We can also test our connectivity:

```
>>> client.ping()
>>> client.flush()
>>> client.read()  # this will read a 'pong' response
```

And logout:

```
>>> client.logout()
```

Placing a bet

The `Order` class provides the mechanism to send a message to create a new order:

```
>>> order = OrderCreate()
>>> order.quantity = 400000  # 40.0000 GBP payout
>>> order.price = 2500  # 25.00%
>>> order.side = BUY
>>> order.market_id = some_market
>>> order.contract_id = some_contract
```

The above order is a **buy** (or **back**) at 25.00% (or 4.0 in decimal format) for a £40.00 return. The buyer's liability if the execution is at 25.00% will be £10.00.

Now, we send the create message:

```
>>> client.order(order)
>>> client.flush()
```

Registering callback functions

We can register some relatively simple callback functions for various messages. This example uses the text_format module from the protocol buffers package to simply print the message to stdout:

```
>>> from google.protobuf import text_format
>>> def login_response_callback(message):
>>>     print "Received a eto.login_response: %s" % (
>>>         text_format.MessageToString(message))
>>> def global_callback(message_name, message):
>>>     print "[global] Received a %s: %s" % (
>>>         message_name, text_format.MessageToString(message.protobuf))
```

First, we register the callback for the eto.login_response message:

```
>>> client.add_handler('eto.login_response', login_response_callback)
```

We can also register a **global** handler which will be called for every message received:

```
>>> client.add_global_handler(global_callback)
```

Change log

9.2.0

- Update smk_api_common to v6.1.0: Introduce labels so users can tag special orders.

9.1.3

- Update smk_api_common to v6.0.6: https://github.com/smarkets/smk_api_common/issues/2 Version 9.1.2 is broken: do not use.

9.1.2

- order-cancelled, order-executed, order-quantity-reduced, order-execute-voided and order-reduced messages now include the following additional order state information: total executed quantity, average executed price, available quantity, origin price

9.1.1

- Fix: Don't set a uint64 value as the old account sequence which makes it overflow

9.1.0

- Add settings and settings-accepted messages
- Add executed avg price/quantity on order-cancelled
- Add a uint64 account sequence
- Add account sequences in order quantity reduce rejected

9.0.0

- The long deprecated uuid fields have been removed from the seto protocol. Use ints instead.

8.0.1

- The streaming API now allows to cancel all orders by market

8.0.0

- The streaming API now dispatches frames instead of payload. This lets you access raw bytes
- Fix examples in README
- Simplify requirements
- Require protobuf when installing
- Use Smarkets' piqi binary fork

7.1.0

- Account sequences in reduce quantity messages
- Keep in play orders
- Trading suspended reasons

7.0.1

- ParseFromString expects string not bytearray

7.0.0

- Use bytearray for buffers instead of byte strings (changes smarkets.streaming_api.framing API)
- BUGFIX: frames_decode_all may hang
- BUGFIX: frame_decode_all could miss frames

6.4.0

- Remove call to quantize in Odds.decimal

6.3.0

- Add functions to query available prices, ie ticks
- Fix flake8 and pin versions

6.2.0

- Upgrade smk_api_common to 5.2.0

6.1.0

- Bump smk_api_common to 5.1.0: reduction messages, cancel all feature
- Fix the broken flake8 build because of flake8-import-order

6.0.0

- Update smk_api_common and eto_common versions

5.0.0

- Removed obsolete smarkets.compatibility and smarkets.rest_api modules

4.1.2

- Update smk_api_common version

4.1.0

- 4.0.0 uploaded to PyPI is broken (it misses some files causing the package to initiate a full piqi -> protobuf -> Python build process on installation), 4.1.0 fixes it
- Made most of the package's dependencies optional
- Improved Python 3 compatibility (all tests pass now), the package isn't advertised as Python 3 compatible because there are some parts of it not tested on Python 3 yet.

4.0.0

Backwards compatible:

- Fixed installation on Python 3 (not the whole package is Python 3-compatible yet but installation works)

Backwards incompatible:

- Removed smarkets.datetime.iso8601_to_datetime (parse_datetime is recommended instead, do note they have different interfaces)

0.6.0

- Merge smkcommon project
- Refactor documentation

0.5.3

- Create separate logger for "flushing x payloads" message

0.5.2

- Update SETO definitions

0.5.1

- Stop requiring curl/piqi/protoc if installing distribution

0.5.0

- Handle order reference property
- Remove per-message streaming API callbacks
- Remove unused API

0.4.x/0.3.x

Change list available only in git log.

0.2.0

- Update to latest eto and seto definitions
- Add additional integration tests
- Add unit tests
- Add first pass at documentation

0.1.0

- Initial Release

CHAPTER 2

API Documentation

Specific modules, classes and methods are listed in this section.

smarkets package

Subpackages

smarkets.streaming_api package

smarkets.streaming_api.client module

smarkets.streaming_api.exceptions module

Core Smarkets API exceptions

exception smarkets.streaming_api.exceptions.**ConnectionError**
Bases: *smarkets.errors.Error*

TCP connection-related error

exception smarkets.streaming_api.exceptions.**DecodeError**
Bases: *smarkets.errors.Error*

Header decoding error

exception smarkets.streaming_api.exceptions.**DownloadError**
Bases: *smarkets.errors.Error*

Raised when a URL could not be fetched

exception smarkets.streaming_api.exceptions.**InvalidCallbackError**
Bases: *smarkets.errors.Error*

Invalid callback was specified

exception smarkets.streaming_api.exceptions.**InvalidUrlError**
Bases: *smarkets.errors.Error*

Raised when a URL is invalid

exception smarkets.streaming_api.exceptions.**ParseError**
Bases: *smarkets.errors.Error*

Error parsing a message or frame

exception smarkets.streaming_api.exceptions.**SocketDisconnected**
Bases: *smarkets.errors.Error*

Socket was disconnected while reading

smarkets.streaming_api.session module

Smarkets TCP-based session management

class smarkets.streaming_api.session.**Session**(*settings*, *inseq=1*, *outseq=1*, *account_sequence=None*)
Bases: object

Manages TCP communication via Smarkets streaming API

connect()
Connects to the API and logs in if not already connected

connected
Returns True if the socket is currently connected

disconnect()
Disconnects from the API

flush()
Flush payloads to the socket

logout()
Disconnects from the API

next_frame()
Get the next payload and increment inseq.

> **Warning:** Payload returned by *next_frame* has to be consumed before next call to *next_frame* happens.

Returns A payload or None if no payloads in buffer.

Return type smarkets.streaming_api.session.Frame or None

raw_socket
Get raw socket used for communication with remote endpoint. :rtype: socket.socket

send()
Serialise, sequence, add header, and send payload

class smarkets.streaming_api.session.**SessionSettings**(*username=None*, *password=None*, *token=None*, *host='stream.smarkets.com'*, *port=3801*, *ssl=True*, *socket_timeout=30*, *ssl_kwargs=None*, *tcp_nodelay=True*)

Bases: object

Encapsulate settings necessary to create a new session

class smarkets.streaming_api.session.**SessionSocket**(*settings*)

Bases: object

Wraps a socket with basic framing/deframing

connect()

Create a TCP socket connection.

Returns True if the socket needed connecting, False if not

connected

Returns True if the socket is currently connected

disconnect()

Close the TCP socket.

recv()

Read stuff from underlying socket.

Return type byte string

send(*byte_array*)

Returns Number of sent bytes

Return type int

Submodules

smarkets.configuration module

class smarkets.configuration.**ConfigurationReader**(*directories*, *parser_class=<class ConfigParser.SafeConfigParser>*)

Bases: object

Reads a configuration from a series of "inheriting" .ini files. You may specify what config file your file inherits from like this:

```
[inherit]
from=other.conf
```

Moreover files of the same name may be present in multiple directories ConfigurationReader is set to look for config files - in this case it will read configuration from all of them but in reverse order. For example, let's have:

- B.conf inherits from ''A.conf''
- files present:
  - /etc/conf/B.conf
  - /home/conf/A.conf

- /home/conf/B.conf

•reader configured like this:

```
reader = ConfigurationReader(('/etc/conf', '/home/conf'))
```

Order in which files will be read:

- •/home/conf/A.conf
- •/home/conf/B.conf
- •/etc/conf/B.conf

read(*filename*)
Reads the configuration into new instance of parser_class. :rtype: ConfigParser

read_into(*filename*, *config*)
Reads the configuration into config object. :type config: ConfigParser

smarkets.errors module

exception smarkets.errors.**Error**
Bases: exceptions.Exception

Base class for every Smarkets error

smarkets.errors.**swallow**(*exceptions*, *default=None*)
Swallow exception(s) when executing something. Works as function decorator and as a context manager:

```
>>> @swallow(NameError, default=2)
... def fun():
...     a = b  # noqa
...     return 1
...
>>> fun()
2
>>> with swallow(KeyError):
...     raise KeyError('key')
...
```

Parameters **default** – value to return in case of an exception

smarkets.functools module

smarkets.functools.**overrides**(*ancestor_class*)
Mark method as overriding ancestor_class' method.

Note: Overriding method can not have its own docstring.

Note: Method being overridden must be (re)defined in ancestor_class itself (see BadChild3 example below); *overrides()* will not follow the inheritance tree.

Usage:

```
>>> class Parent(object):
...     def method(self):
...         'parent docstring'
...
>>>
>>> class BadChild1(Parent):
...     @overrides(Parent)
...     def methd(self):
...         pass
...
...
Traceback (most recent call last):
OverrideError: No method 'methd' in class <class 'smarkets.functools.Parent'> to␣
↪override
>>>
>>> class BadChild2(Parent):
...     @overrides(Parent)
...     def method(self):
...         'child method docstring'
...
...
Traceback (most recent call last):
OverrideError: No docstrings allowed in overriding method
>>>
>>> class IntermediateChild(Parent):
...     pass
...
>>> class BadChild3(IntermediateChild):
...     @overrides(IntermediateChild)
...     def method(self):
...         pass
...
...
Traceback (most recent call last):
OverrideError: No method 'method' in class <class 'smarkets.functools.
↪IntermediateChild'> to override
>>>
>>> class GoodChild(Parent):
...     @overrides(Parent)
...     def method(self):
...         return 1
...
>>> child = GoodChild()
>>> str(child.method.__doc__)
'parent docstring'
>>> child.method()
1
```

Raises

OverrideError Method does not exist in parent class or overriding method has docstring.

exception smarkets.functools.**OverrideError**

Bases: exceptions.Exception

Method override fails

smarkets.functools.**lru_cache**(*maxsize=100*, *typed=False*)
Least-recently-used cache decorator.

If *maxsize* is set to None, the LRU features are disabled and the cache can grow without bound.

If *typed* is True, arguments of different types will be cached separately. For example, f(3.0) and f(3) will be treated as distinct calls with distinct results.

Arguments to the cached function must be hashable.

View the cache statistics named tuple (hits, misses, maxsize, currsize) with f.cache_info(). Clear the cache and statistics with f.cache_clear(). Access the underlying function with f.__wrapped__.

See: http://en.wikipedia.org/wiki/Cache_algorithms#Least_Recently_Used

smarkets.itertools module

smarkets.itertools.**listitems**(*d*)
Return *d* item list

smarkets.itertools.**listkeys**(*d*)
Return *d* key list

smarkets.itertools.**listvalues**(*d*)
Return *d* value list

smarkets.itertools.**merge_dicts**(**dicts*)
Return *dicts* merged together.

If keys clash the ubsequent dictionaries have priority over preceding ones.

```
>>> merge_dicts() == {}
True
>>> merge_dicts({'a': 2}) == {'a': 2}
True
>>> merge_dicts({'a': 2, 'b': 3}, {'a': 1, 'c': 4}) == {'a': 1, 'b': 3, 'c': 4}
True
```

smarkets.itertools.**listmap**()
map(function, sequence[, sequence, ...]) -> list

Return a list of the results of applying the function to the items of the argument sequence(s). If more than one sequence is given, the function is called with an argument list consisting of the corresponding item of each sequence, substituting None for missing values when not all sequences have the same length. If the function is None, return a list of the items of the sequence (or a list of tuples if more than one sequence).

smarkets.itertools.**inverse_mapping**(*d*)
Return a dictionary with input mapping keys as values and values as keys.

Raises

ValueError Input mapping values aren't uniqe.

smarkets.itertools.**is_sorted**(*sequence*, ***kwargs*)

Parameters **kwargs** – `sorted()` kwargs

smarkets.itertools.**copy_keys_if_present**(*source*, *destination*, *keys*)
Copy keys from source mapping to destination mapping while skipping nonexistent keys.

smarkets.itertools.**listmap**()
map(function, sequence[, sequence, ...]) -> list

Return a list of the results of applying the function to the items of the argument sequence(s). If more than one sequence is given, the function is called with an argument list consisting of the corresponding item of each sequence, substituting None for missing values when not all sequences have the same length. If the function is None, return a list of the items of the sequence (or a list of tuples if more than one sequence).

smarkets.signal module

class smarkets.signal.**Signal**
Bases: object

All instance methods of this class are thread safe.

add(*handler*)
Add signal handler. You can also do:

```
signal = Signal()
signal += handler
```

fire(***kwargs*)
Execute all handlers associated with this Signal.

You can also call signal object to get the same result:

```
signal = Signal()
signal()  # calls the signal handler
```

handle(*handler*)
Add signal handler. You can also do:

```
signal = Signal()
signal += handler
```

remove(*handler*)
Remove signal handler. You can also do:

```
signal = Signal()
# add a handler "handler"
signal -= handler
```

smarkets.uuid module

class smarkets.uuid.**Uuid**
Bases: *smarkets.uuid.UuidBase*

Represents a UUID

static base_n(*number*, *chars*)
Recursive helper for calculating a number in base len(chars)

chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'

classmethod from_hex(*hex_str*)
Convert a hex uuid into a Uuid :type hex_str: byte string or unicode string

classmethod from_int(*number*, *ttype*)
Convert an integer and tag type to a Uuid

classmethod from_slug(*slug*, *base=36*, *chars=None*)
Convert a slug into a Uuid

high
Higher 64 bits of number

low
Lower 64 bits of number

mask64 = 18446744073709551615L

static pad_uuid(*uuid*, *pad=32*, *padchar='0'*)
Pads a UUID with <pad> <padchar>s

shorthex
Short hex representation of Uuid

tags = {'Comment': UuidTagBase(name='Comment', int_tag=45476, prefix='b'), 'Account': UuidTagBase(name='Accou

tags_by_hex_str = {'9999': UuidTagBase(name='Session', int_tag=39321, prefix='s'), '0f00': UuidTagBase(name='Us

tags_by_int_tag = {4352: UuidTagBase(name='Event', int_tag=4352, prefix='e'), 44225: UuidTagBase(name='Accou

tags_by_prefix = {'a': UuidTagBase(name='Account', int_tag=44225, prefix='a'), 'c': UuidTagBase(name='Contract

to_hex(*pad=32*)
Convert to tagged hex representation

to_slug(*prefix=True*, *base=36*, *chars=None*, *pad=0*)
Convert to slug representation

classmethod unsplit64(*high*, *low*)
Converts a high/low 64-bit integer pair into a 128-bit large integer

class smarkets.uuid.**UuidBase**(*number*, *tag*)
Bases: tuple

number
Alias for field number 0

tag
Alias for field number 1

class smarkets.uuid.**UuidTag**
Bases: *smarkets.uuid.UuidTagBase*

Represents tag information

hex_str
Hex tag value

classmethod split_int_tag(*number*)
Splits a number into the ID and tag

tag_mult = 65536

tag_number(*number*)
Adds this tag to a number

class smarkets.uuid.**UuidTagBase**(*name*, *int_tag*, *prefix*)
Bases: tuple

int_tag
Alias for field number 1

name
Alias for field number 0

prefix
Alias for field number 2

smarkets.uuid.**int_to_slug**(*number*, *ttype*)
Convert a large integer to a slug

smarkets.uuid.**int_to_uuid**(*number*, *ttype*)
Convert an untagged integer into a tagged uuid

smarkets.uuid.**slug_to_int**(*slug*, *return_tag=None*, *split=False*)
Convert a slug to an integer, optionally splitting into high and low 64 bit parts

smarkets.uuid.**slug_to_uuid**(*slug*)
Convert a slug to a Smarkets UUID

smarkets.uuid.**uuid_to_int**(*uuid*, *return_tag=None*, *split=False*)
Convert a tagged uuid into an integer, optionally returning type

smarkets.uuid.**uuid_to_short**(*uuid*)
Converts a full UUID to the shortened version

smarkets.uuid.**uuid_to_slug**(*number*, *prefix=True*)
Convert a Smarkets UUID (128-bit hex) to a slug

Module contents

CHAPTER 3

Support

Stuck? Found a bug? If you are looking for help, please contact Smarkets directly via email on support@smarkets.com.

Python Module Index

S

Index

P

R

S

T

U