sptrans Documentation

Release 0.1.0

Diogo Baeder

October 31, 2013

CONTENTS

Python library for the SPTrans API.

CHAPTER ONE

CHANGELOG

1.1 0.1.0

- First version of the library
- All methods from the SPTrans API implemented
- Some documentation is still under development

CHAPTER TWO

SPTRANS PACKAGE

2.1 v0 Module

Module for the v0 version of the SPTrans API.

The first thing you have to do, in order to use it, is to instantiate a client and authenticate to the API:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
```

Then you can use the other methods to grab data from the API.

exception `sptrans.v0.`**`AuthenticationError`**

Bases: `exceptions.Exception`

Raised when the authentication fails - for example, with a wrong token -.

class `sptrans.v0.`**`Client`**

Bases: `object`

Main client class.

Warning: Any method (except `authenticate()`) may raise `RequestError` if the client is not authenticated anymore. So keep an eye on the authentication state.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
```

`authenticate`(*token*)

Authenticates to the webservice.

Parameters **token** (`str`) – The API token string.

Raises `AuthenticationError` when there's an error during authentication.

`get_forecast`(*stop_code=None*, *route_code=None*)

Gets the arrival forecast, provided a route code or a stop code or both.

You must provide at least one of the parameters. If you provide only a *stop_code*, it will return a forecast for all routes that attend a certain stop. If you provide only a *route_code*, it will return a forecast for all stops that a certain route attends to. If you provide both, it will return a forecast for the specific route attending to the specific stop provided.

Parameters

- **stop_code** (`int`) – The stop code to use for matching.
- **route_code** (`int`) – The stop code to use for matching.

Returns A single `ForecastWithStop` object, when passing only *stop_code* or both.

Returns A single `ForecastWithStops` object, when passing only *route_code*.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')

forecast = client.get_forecast(stop_code=1234)
for route in forecast.stop.routes:
    for vehicle in route.vehicles:
        print(vehicle.prefix)

forecast = client.get_forecast(stop_code=1234, route_code=2345)
for route in forecast.stop.routes:
    for vehicle in route.vehicles:
        print(vehicle.prefix)

forecast = client.get_forecast(route_code=2345)
for stop in forecast.stops:
    for vehicle in stop.vehicles:
        print(vehicle.prefix)
```

`get_positions`(*code*)

Gets the vehicles with their current positions, provided a route code.

Parameters **code** (`int`) – The route code to use for matching.

Returns A single `Positions` object.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')

positions = client.get_positions(1234)
print(positions.time)

for vehicle in positions.vehicles:
    print(vehicle.prefix)
```

`list_lanes`()

Lists all the bus lanes in the city.

Returns A generator that yields `Lane` objects.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
for lane in client.list_lanes():
    print(lane.code, lane.name)
```

search_routes(*keywords*)
Searches for routes that match the provided keywords.

Parameters **keywords** (str) – The keywords, in a single string, to use for matching.

Returns A generator that yields Route objects.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
for route in client.search_routes('butanta'):
    print(route.code, route.sign)
```

search_stops(*keywords*)
Searches for bus stops that match the provided keywords.

Parameters **keywords** (str) – The keywords, in a single string, to use for matching.

Returns A generator that yields Stop objects.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
for stop in client.search_stops('butanta'):
    print(stop.code, stop.name)
```

search_stops_by_lane(*code*)
Searches for bus stops that are contained in a lane specified by its code.

Parameters **code** (int) – The lane code to use for matching.

Returns A generator that yields Stop objects.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
for stop in client.search_stops_by_lane(1234):
    print(stop.code, stop.name)
```

search_stops_by_route(*code*)
Searches for bus stops that are passed by the route specified by its code.

Parameters **code** (`int`) – The route code to use for matching.

Returns A generator that yields `Stop` objects.

Example:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
for stop in client.search_stops_by_route(1234):
    print(stop.code, stop.name)
```

class `sptrans.v0.`**`ForecastWithStop`**

Bases: `sptrans.v0.ForecastWithStop`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a bus stop forecast with routes and the time when the information was retrieved.

Variables

- **time** – (`datetime.datetime`) The time when the information was retrieved.
- **stop** – (`StopWithRoutes`) The bus stop with `routes`.

class `sptrans.v0.`**`ForecastWithStops`**

Bases: `sptrans.v0.ForecastWithStops`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a list of bus stops forecast with vehicles and the time when the information was retrieved.

Variables

- **time** – (`datetime.datetime`) The time when the information was retrieved.
- **stops** – (`list` of `StopWithVehicles`) The bus stops.

class `sptrans.v0.`**`Lane`**

Bases: `sptrans.v0.Lane`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a bus lane.

Variables

- **code** – (`int`) The lane code.
- **cot** – (`int`) The lane "cot" (?).
- **name** – (`str`) The lane name.

class `sptrans.v0.`**`Positions`**

Bases: `sptrans.v0.Positions`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a sequence of vehicles positions, with the time when the information was retrieved.

Variables

- **time** – (`datetime.datetime`) The time when the information was retrieved.
- **vehicles** – (`list`) The list of `vehicles`.

exception `sptrans.v0.`**`RequestError`**

Bases: `exceptions.Exception`

Raised when the request failes to be accomplished.

Normally this is due to the client not being authenticated anymore. In this case, just authenticate again, and it should be back at work.

class `sptrans.v0.`**`Route`**

Bases: `sptrans.v0.Route`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a route.

Variables

- **code** – (`int`) The route code.
- **circular** – (`bool`) Wether it's a circular route or not (without a secondary terminal).
- **sign** – (`str`) The first part of the route sign.
- **direction** – (`int`) The route direction. "1" means "main to secondary terminal", "2" means "secondary to main terminal".
- **type** – (`int`) The route type.
- **main_to_sec** – (`str`) The name of the route when moving from the main terminal to the second one.
- **sec_to_main** – (`str`) The name of the route when moving from the second terminal to the main one.
- **info** – (`str`) Extra information about the route.

class `sptrans.v0.`**`RouteWithVehicles`**

Bases: `sptrans.v0.RouteWithVehicles`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a route with a sequence of vehicles with their current positions.

Variables

- **sign** – (`str`) The first part of the route sign.
- **code** – (`int`) The route code.
- **direction** – (`int`) The route direction. "1" means "main to secondary terminal", "2" means "secondary to main terminal".
- **main_to_sec** – (`str`) The name of the route when moving from the main terminal to the second one.
- **sec_to_main** – (`str`) The name of the route when moving from the second terminal to the main one.
- **quantity** – (`int`) The quantity of vehicles.
- **vehicles** – (`list`) The list of `vehicles`.

class `sptrans.v0.`**`Stop`**

Bases: `sptrans.v0.Stop`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a bus stop.

Variables

- **code** – (`int`) The stop code.
- **name** – (`str`) The stop name.
- **address** – (`str`) The stop address.
- **latitude** – (`float`) The stop latitude.
- **longitude** – (`float`) The stop longitude.

class `sptrans.v0.`**`StopWithRoutes`**

Bases: `sptrans.v0.StopWithRoutes`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a bus stop with a list of routes that pass through this stop.

Variables

- **code** – (`int`) The stop code.
- **name** – (`str`) The stop name.
- **latitude** – (`float`) The stop latitude.
- **longitude** – (`float`) The stop longitude.
- **routes** – (`list`) The list of `routes` that pass through this stop.

class `sptrans.v0.`**`StopWithVehicles`**

Bases: `sptrans.v0.StopWithVehicles`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a bus stop with a list of vehicles that pass through this stop.

Variables

- **code** – (`int`) The stop code.
- **name** – (`str`) The stop name.
- **latitude** – (`float`) The stop latitude.
- **longitude** – (`float`) The stop longitude.
- **vehicles** – (`list`) The list of `vehicles`.

class `sptrans.v0.`**`Vehicle`**

Bases: `sptrans.v0.Vehicle`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a vehicle (bus) with its position.

Variables

- **prefix** – (`str`) The vehicle prefix painted in the bus.
- **accessible** – (`bool`) Wether the vehicle is accessible or not.
- **latitude** – (`float`) The vehicle latitude.
- **longitude** – (`float`) The vehicle longitude.

class `sptrans.v0.`**`VehicleForecast`**

Bases: `sptrans.v0.VehicleForecast`, `sptrans.v0.TupleMapMixin`

A namedtuple representing a vehicle (bus) with its position and forecast to arrive at a certain stop.

Variables

- **prefix** – (`str`) The vehicle prefix painted in the bus.
- **accessible** – (`bool`) Wether the vehicle is accessible or not.
- **arriving_at** – (`datetime.datetime`) The time that the vehicle is expected to arrive.
- **latitude** – (`float`) The vehicle latitude.
- **longitude** – (`float`) The vehicle longitude.

CHAPTER

THREE

CONTRIBUTING

First of all, take a look at the repository here.

You may have noticed that there are two images in the README: one that tells the status of the build (via Continuous Integration in travis-ci) and another that tells the code coverage in coveralls.

This is because I'm a bit obsessive with automated tests, and though I don't think 100% code coverage makes a program free of bugs, it does a great job getting as close as possible to this achievement.

By the way, the build doesn't even pass if the production code is not fully covered, so if you want to contribute with it, please also add some tests.

3.1 Setting up the environment

1. Make sure you have pip, virtualenv and virtualenvwrapper installed. OK, all of these are optional, but it would be a nice moment for you to start using them if you haven't yet;
2. Create a virtual environment for the project, and activate it;
3. Install the requirements for running the tests:

   ```
   $ pip install -r requirements.txt
   ```

4. Run the tests:

   ```
   $ tox
   ```

   Optionally, you can run the tests with an SPTrans API token, so that the functional tests are also run:

   ```
   $ SPTRANS_TOKEN=this-is-my-token tox
   ```

CHAPTER

FOUR

ABOUT

This library was developed as a Python client to the SPTrans API. It was made in English so that people from other countries can used it, and contribute to it, so you may expect some discrepancies between the library and the SPTrans API itself.

CHAPTER

FIVE

GETTING STARTED

5.1 Installing

This library is tested and works under Python 2.7 and 3.3.

Plain and simple:

```
$ pip install sptrans
```

Or, if you want the latest version under development:

```
$ pip install -e git+https://github.com/diogobaeder/sptrans.git#egg=sptrans
```

Or, if you still don't want to use pip, just grab the code at GitHub and install it with:

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

5.2 Usage

(*Looking for the library API? Here's a shortcut!*)

Before using the library, you need to have an API token issued at the SPTrans website.

As soon as you have the token, use the `client` itself to authenticate to the API:

```
from sptrans.v0 import Client


client = Client()
client.authenticate('this is my token')
```

Now checkout the other methods available in the `Client` class, to see how the library can help you retrieving data.

CHAPTER

SIX

DOCUMENTATION

- *Changelog*
- *library API*
- *Contributing*

CHAPTER

SEVEN

LICENSE

Copyright (c) 2013, Diogo Baeder All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

> Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
>
> Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CHAPTER

EIGHT

INDICES AND TABLES

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

PYTHON MODULE INDEX