Star API Documentation

Version 0.0.1

Florian Strzelecki

08 December 2013

Table des matières

CHAPITRE 1

Le module starapi

Le module `starapi` est le seul *import* dont vous avez besoin. Il contient deux classes `Handler` et `BaseHandler` qui permettent d'interroger l'API Star.

1.1 Utilisation

Pour utiliser le client python de l'API, il suffit de l'installer, et d'importer le module `starapi`. Ensuite, muni de vôtre clé d'API, vous pouvez instancier un `Handler` qui formatera les requêtes pour vous.

Lorsque vous appelez une fonction du `Handler`, les paramètres sont vérifiés, puis une url est construite avec les paramètres formatés comme il faut. Enfin, un appel HTTP est effectué, et le résultat est directement retourné comme réponse à l'appel de la fonction.

Comme le module *requests* est utilisé, c'est donc un objet *Response* de ce même module qui est retourné à chaque appel.

1.1.1 Format de réponse

Le format de réponse des handlers est un objet Response du module *requests*. Le contenu de l'attribut *text* est alors le résultat de la requête.

Dans le cas du Handler de base, il s'agit d'une réponse formatée en XML, qui peut être interprétée (par exemple) avec *lxml* :

```
>>> r = api.getlines()
>>> root = root.fromstring(r.text.encode('utf-8'))
>>> root.xpath('/opendata/answer/status')
[<Element status at 0x1ed2eb0>]
>>> status = root.xpath('/opendata/answer/status')[0]
>>> status.get('code')
'0'
>>> status.get('message')
'OK'
```

Remarque importante : la valeur de l'attribut *text* est une chaîne de caractère unicode. Comme *lxml.etree* attend une chaîne encodée lorsque le XML contient une entête, pensez à encoder d'abords le résultat en *utf-8*.

1.1.2 Structure du XML de réponse

La structure de la réponse XML est toujours la même lorsque la requête a aboutie correctement :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opendata>
    <request><!-- REQUEST URL --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <!-- DATA RESPONSE -->
        </data>
    </answer>
</opendata>
```

La balise *request* contient l'url de la requête appelée, avec tous ses paramètres.

La balise *data* contient le contenu XML de la réponse, en tant que noeud normal du document XML (ce n'est donc pas du CDATA à interprété, mais bien une suite de noeud XML valide).

En cas d'erreur, l'élément */opendata/answer/data* n'existe pas, et un code d'erreur est présent dans la balise *status*.

1.1.3 Status code

Les codes de retour de l'API les plus fréquents sont les suivants :

Code	Message	Signification
0	OK	Tout va bien.
3	The requested command could not be found. Please check spelling.	Commande introuvable : probablement un bug du client à signaler.
103	A required parameter is not filled. Please, check parameters.	Paramètre non fourni, le bug est de vôtre côté !

1.2 Constantes, fonctions et décorateurs

Une bonne partie du travail du client consiste à appeler des URLs de l'API en choisissant la bonne version, la bonne commande, et la bonne clé.

Pour simplifier une partie du travail, les méthodes de la classe `Handler` sont décorées avec le décorateur `api_command()`.

Cette dernière prend en premier paramètre la version de l'API qui supporte la commande, qui peut prendre comme valeur l'une des constantes suivantes :

`starapi.`**`KEOLIS_VERSION_1`**

`starapi.`**`KEOLIS_VERSION_2`**

`starapi.`**`KEOLIS_VERSION_2_1`**

Respectivement ayant pour valeur *1.0*, *2.0* et *2.1*.

`starapi.`**`api_command`**(*version*, *command=None*)

Paramètres
- **version** (*string*) – La version de l'API ciblée.
- **command** (*string*) – Le nom de la commande ciblée.

Type retourné function

Cette fonction est un décorateur qui permet d'encapsuler l'envoie de la commande en tant que requête HTTP GET à l'API Keolis, sans avoir besoin de gérer la version et/ou le nom de la commande dans le corps de la fonction qui traite les paramètres de la commande.

La fonction décorée doit prendre en premier paramètre un API Handler (soit un `BaseHandler` soit un `Handler`), et doit retourner un *dict* contenant les paramètres à envoyer à l'API.

Le paramètre *command* est optionnel. S'il est omis, alors le nom de la fonction décorée est utilisé à la place.

Lors de l'appel de la fonction décorée, elle est appelée avec les paramètres fournis, retourne un dictionnaire, et un appel à l'API du STAR est effectué.

Le retour est alors un objet *Response* du module *requests*. Voir aussi la documentation de request

Utilisation :

```
@api_command(KEOLIS_VERSION_1)
def getdistrict(self):
    """
    Get districts
    See: http://data.keolis-rennes.com/fr/les-donnees/api-version-1/getdistrict.html
    """
    return {}
```

1.3 Les Handlers

1.3.1 Base Handler

class `starapi.`**`BaseHandler`**(*key*)

Classe de base de client handler. La clé à fournir en paramètre d'instanciation est la clé fournie à l'inscription auprès du site de la Star.

`call_api`(*command*, *version*, *params*)

Paramètres

- **command** (*string*) – Le nom de la commande.
- **version** (*string*) – Le numéro de version de l'API.
- **params** (*dict*) – Le dictionnaire des paramètres à fournir.

Procède à un appel HTTP GET de la commande *command* à l'API dans la version *version*.

1.3.2 Handler

class `starapi.`**`Handler`**(*key*)

Classe qui sert de client handler à l'API. C'est elle qui gère l'ensemble des commandes à envoyer à l'API.

Pour fonctionner, le handler a besoin de la clé de l'application inscrite auprès du site de la Star comme application.

Les méthodes de cette classe implémentent les différentes commandes de l'API et fait un usage intensif du décorateur `api_command()`.

`getstation`(*network=None*, *request=None*, ***kwargs*)

Paramètres

- **network** (*string*) – 'levelostar' par défaut (aucune autre valeur connue)
- **request** (*string*) – 'all', 'proximity', 'district' ou 'number'
- **mode** (*string*) – Request 'proximity', valeur 'id' ou 'coord'
- **id** (*string*) – Request 'proximity' mode 'id', identifiant de la station.
- **lat** (*string*) – Request 'proximity' mode 'coord', latitude du point autour duquel chercher
- **long** (*string*) – Request 'proximity' mode 'coord', longitude du point autour duquel chercher.

– **value** (*string*) – Request 'district' ou 'number', valeur de recherche.

Référence : getstation

Exemple de réponse :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <station>
                <id>42</id>
                <number>42</number>
                <name>PONT DE STRASBOURG</name>
                <state>1</state>
                <latitude>48.109756</latitude>
                <longitude>-1.656451</longitude>
                <slotsavailable>11</slotsavailable>
                <bikesavailable>0</bikesavailable>
                <pos>0</pos>
                <district>Thabor - St Hélier</district>
                <lastupdate>2012-11-27T19:32:02+01:00</lastupdate>
            </station>
            <!-- another "station" tags -->
        </data>
    </answer>
</opendata>
```

getdistrict()

Référence : getdistrict

Exemple de réponse :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <district>
                <id>34</id>
                <name>Sud-Gare</name>
            </district>
            <!-- another "district" tags -->
        </data>
    </answer>
</opendata>
```

getbikestations(*network=None*, *station=None*, ***kwargs*)

Paramètres

– **network** (*string*) – 'levelostar' par défaut (aucune autre valeur connue)
– **station** (*string*) – 'all' (défaut), 'district', 'number' ou 'proximity'
– **mode** (*string*) – Station 'proximity' : mode de proximité : 'id' ou 'coord'.
– **id** (*string*) – Station 'proximity', mode 'id' : identifiant de la station autour de laquelle chercher.
– **lat** (*string*) – Station 'proximity', mode 'coord' : latitude du point autour duquel chercher.
– **long** (*string*) – Station 'proximity', mode 'coord' : longitude du point autour duquel chercher.
– **value** (*string*) – Station 'district' ou 'number' : valeur sur laquelle filtrer.

Référence : getbikestations

Exemple de réponse :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <station>
                <number>54</number>
                <name>PONTCHAILLOU (TER)</name>
                <address>26 AVENUE DU 41ÈME RÉGIMENT D'INFANTER </address>
                <state>1</state>
                <latitude>48.119316</latitude>
                <longitude>-1.691517</longitude>
                <slotsavailable>23</slotsavailable>
                <bikesavailable>1</bikesavailable>
                <pos>0</pos>
                <district>Villejean-Beauregard</district>
                <lastupdate>2012-12-02T18:15:03+01:00</lastupdate>
            </station>
            <!-- another "station" tags -->
        </data>
    </answer>
</opendata>
```

getbikedistricts (*network=None*)

Paramètres network (*string*) – 'levelostar' par défaut (aucune autre valeur connue)

Référence : getbikedistricts

Exemple de réponse :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <district>
                <id>34</id>
                <name>Sud-Gare</name>
            </district>
            <!-- another "district" tags -->
        </data>
    </answer>
</opendata>
```

getlinesalerts (*network=None*, *mode=None*, ***kwargs*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' (défaut) ou 'line'
- **line** (*string*) – Mode 'line', la ligne sur laquelle chercher.

Référence : getlinesalerts

Exemple de réponse :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <alert>
                <title>Travaux Bd Villebois Mareuil</title>
                <starttime>2012-12-01T00:00:00+01:00</starttime>
                <endtime>2012-12-12T00:00:00+01:00</endtime>
                <lines>
                    <line>32</line>
                </lines>
                <majordisturbance>0</majordisturbance>
                <detail>A partir du lundi 3 décembre, dès le premier départ et pendant la du
Ligne 32  vers Triangle&#13;
L'arrêt Cimetière de l'Est est reporté à l'arrêt Cimetière de l'Est de la ligne 11 vers Sain
L'arrêt Villebois Mareuil est reporté à l'arrêt provisoire situé boulevard Léon Bourgeois.&#
Ligne 32  vers Beaulieu Atalante&#13;
L'arrêt Villebois Mareuil est reporté à l'arrêt Villebois Mareuil de la ligne 1 vers Cesson-
L'arrêt Cimetière de l'Est est reporté à l'arrêt Cimetière de l'Est de la ligne 11 vers Stad
                <link></link>
            </alert>
            <!-- another "alert" tags -->
        </data>
    </answer>
</opendata>
```

getlines (*self*, *network=None*, *mode=None*, *size=None*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' par défaut (aucune autre valeur connue)
- **size** (*string*) – Taille en pixel dans '21' (défaut), '100', '300' ou '1000'

Référence getlines

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <baseurl>http://data.keolis-rennes.com/fileadmin/documents/Picto_lignes/Pictos_l
            <line>
                <name>1</name>
                <picto>L1.png</picto>
            </line>
            <!-- another "line" tags -->
        </data>
    </answer>
</opendata>
```

getequipments (*network=None*, *mode=None*, ***kwargs*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' (defaut), 'station' ou 'id'
- **station** (*string*) – Mode 'station' : nom de la station
- **id** (*string*) – Mode 'id' : id de l'équipement.

Référence getequipments

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <equipment>
                <id>ASC_VU_2</id>
                <station>VU</station>
                <type>ASCENSEUR</type>
                <fromfloor>-1</fromfloor>
                <tofloor>0</tofloor>
                <platform>2</platform>
                <lastupdate>2012-12-02 07:16:13</lastupdate>
            </equipment>
            <!-- another "equipment" tags -->
        </data>
    </answer>
</opendata>
```

getequipmentsstatus (*network=None*, *mode=None*, ***kwargs*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' (defaut), 'station' ou 'id'
- **station** (*string*) – Mode 'station' : nom de la station
- **id** (*string*) – Mode 'id' : id de l'équipement

Référence getequipmentsstatus

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <equipment>
                <id>ASC_VU_2</id>
                <state>1</state>
                <lastupdate>2012-12-02 07:16:13</lastupdate>
            </equipment>
            <!-- another "equipment" tags -->
        </data>
    </answer>
</opendata>
```

getmetrostations (*network=None*, *mode=None*, ***kwargs*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' (defaut), 'proximity' ou 'station'
- **proximity_type** (*string*) – Mode 'proximity' : 'station' ou 'coords'
- **lat** (*string*) – Mode 'proximity', type 'coords' : latitude du point autour duquel chercher.
- **long** (*string*) – Mode 'proximity', type 'coords' : longitude du point autour duquel chercher.
- **station** (*string*) – Mode 'proximity', type 'station' : La station autour de laquelle chercher. Mode 'station' : La station à chercher.

Référence getmetrostations

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <station>
                <id>VU</id>
                <name>Villejean-Université</name>
                <latitude>48.12125000</latitude>
                <longitude>-1.703950000</longitude>
                <hasPlatformDirection1>1</hasPlatformDirection1>
                <hasPlatformDirection2>1</hasPlatformDirection2>
                <rankingPlatformDirection1>14</rankingPlatformDirection1>
                <rankingPlatformDirection2>16</rankingPlatformDirection2>
                <floors>-1</floors>
                <lastupdate>2012-12-02T18:29:54+01:00</lastupdate>
            </station>
            <!-- another "station" tags -->
        </data>
    </answer>
</opendata>
```

getmetrostationsstatus (*network=None*, *mode=None*, ***kwargs*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' (défaut), 'proximity' ou 'station'
- **proximity_type** (*string*) – Mode 'proximity' : 'station' ou 'coords'
- **lat** (*string*) – Mode 'proximity', type 'coords' : latitude du point autour duquel chercher.
- **long** (*string*) – Mode 'proximity', type 'coords' : longitude du point autour duquel chercher.
- **station** (*string*) – Mode 'proximity' : La station autour de laquelle chercher.
  Mode 'station' : La station de métro à chercher.

Référence : getmetrostationsstatus

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <station>
                <id>JFK</id>
                <status>1</status>
                <lastUpdate>2012-12-02T18:28:32+01:00</lastUpdate>
            </station>
            <!-- another "station" tags -->
        </data>
    </answer>
</opendata>
```

getrelayparks (*network=None*, *latitude=None*, *longitude=None*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **latitude** (*string*) – latitude du point autour duquel chercher.

– **longitude** (*string*) – longitude du point autour duquel chercher.

Référence : getrelayparks

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <relaypark>
                <name>Henri Freville</name>
                <latitude>48.0886255</latitude>
                <longitude>-1.670222</longitude>
                <carparkavailable>201</carparkavailable>
                <carparkcapacity>406</carparkcapacity>
                <lastupdate>2010-09-27T08:30:49+02:00</lastupdate>
                <state>0</state>
            </relaypark>
            <!-- another "relaypark" tags -->
        </data>
    </answer>
</opendata>
```

getpos (*network=None*, *mode=None*, ***kwargs*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' (défaut), 'proximity' ou 'zone'
- **latitude** (*string*) – Mode 'proximity' : latitude du point autour duquel chercher.
- **longitude** (*string*) – Mode 'proximity' : longitude du point autour duquel chercher.
- **city** (*string*) – Mode 'zone' : la ville sur laquelle filtrer.
- **district** (*string*) – Mode 'zone' (optionnel) : le district sur lequel filtrer.

Référence : getpos

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <pos>
                <name>Relais H / Gare SNCF</name>
                <type>Tabac Presse</type>
                <address>Place de la gare</address>
                <zipcode>35000</zipcode>
                <city>RENNES</city>
                <district>Gares</district>
                <phone>02 99 41 91 44</phone>
                <schedule />
                <latitude>48.1041574</latitude>
                <longitude>-1.6726879</longitude>
            </pos>
            <!-- another "pos" tags -->
        </data>
    </answer>
</opendata>
```

getcities (*network=None*, *mode=None*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'all' par défaut (aucune autre valeur connue)

Référence : getcities

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <city>
                <name>RENNES</name>
                <district>26</district>
                <id>1</id>
            </city>
            <!-- another "city" tags -->
        </data>
    </answer>
</opendata>
```

getcitydistricts (*network=None*, *mode=None*, *city=None*)

Paramètres

- **network** (*string*) – 'star' par défaut (aucune autre valeur connue)
- **mode** (*string*) – 'city' par défaut (aucune autre valeur connue)
- **city** (*string*) – Le nom de la ville dont on veut les districts.

Référence : getcitydistricts

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <district>
                <name>Beauregard</name>
                <id>1</id>
            </district>
            <!-- another "district" tags -->
        </data>
    </answer>
</opendata>
```

getbusnextdepartures (*mode=None*, ***kwargs*)

Paramètres

- **mode** (*string*) – Mode parmis 'stop', 'line', et 'stopline'
- **route** (*string|list*) – Mode 'line' : la route sur laquelle rechercher.
  Mode 'stopline' : liste de route sur lesquelles chercher (10 maximum).
- **direction** (*string|list*) – Mode 'line' : la direction sur laquelle rechercher.
  Mode 'stopline' : liste des directions sur lesquelles chercher (10 maximum).
- **stop** (*list*) – Mode 'stop' : liste d'identifiant d'arrêt (5 maximum).
  Mode 'stopline' : liste d'identifiant d'arrêt (10 maximum)

Référence : getbusnextdepartures

Exemple de réponse :

```
<opendata>
    <request><!-- snip --></request>
    <answer>
        <status code="0" message="OK"/>
        <data>
            <stopline>
                <stop>856</stop>
                <route>965</route>
                <direction>0</direction>
                <departures>
                    <departure accurate="1" headsign="Champs Blancs">2012-08-25T15:01:35+02:
                    <departure accurate="0" headsign="Champs Blancs">2012-08-25T15:16:00+02:
                </departures>
            </stopline>
            <!-- another "stopline" tags -->
        </data>
    </answer>
</opendata>
```

La STAR (société qui gère les transports en commun de Rennes Métropole) fournit des données sur son service de transport, dont des données temps réels.

Lors de l'ouverture de données temps réel en Open-Data, il était plus que temps de se pencher dessus, et de proposer un client en python pour interroger l'API Open-Data de la STAR.

Cette documentation propose de décrire le fonctionnement de ce client.

Amusez-vous bien, et bon voyage.

CHAPITRE 2

Licence et notes aux utilisateurs

Ce client est développé sous licence LGPL, par un développeur du Collectif Open-Data Rennes, association indépendante de Kéolis, de la STAR, et de Rennes Métropole.

Ce n'est donc pas un client "officiel", mais nous espérons qu'il sera satisfaisant.

CHAPITRE 3

Installation

Après avoir téléchargé les sources du projet, l'installation se fait avec setuptools, avec les droits administrateurs : :

```
python setup.py install
```

Ceci doit installer le module `starapi` ainsi que sa dépendance principale, le module *requests*.

CHAPITRE 4

Indices and tables

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

Index des modules Python

S