
Star API Client Documentation

Version 0.0.3-dev

Florian Strzelecki

24 February 2017

1	Comment utiliser le module starapi	1
1.1	Client API	1
1.2	La couche basse : handler et python	2
2	Le module starapi	5
2.1	Les clients	5
2.2	Autres éléments et bas niveau	11
2.3	Utilitaires	19
3	Licence et notes aux utilisateurs	21
4	Installation	23
5	Tables d'index	25
	Index des modules Python	27

Comment utiliser le module starapi

Après avoir installé le module `starapi` vous avez besoin de deux choses :

- Une clé d’API fournie par Keolis,
- Faire un import du module `starapi` et utiliser les classes dont vous avez besoin.

La façon la plus simple pour utiliser le module est ensuite de profiter des clients d’API (l’un pour Levelo Star, et l’autre pour tout ce qui est bus et métro), qui sont décrits ci-après.

Une façon plus avancée consiste à utiliser (voire à écrire vous-même) un *Handler* dit de “bas niveau”, qui vous permet une meilleure granularité.

Client API

Il existe deux clients pour l’API, dit de “haut-niveau” :

- Le client pour le réseau de vélo du Star : `starapi.LeveloStar`.
- Le client pour le réseau de bus et de métro du Star : `starapi.NetworkStar`.

Chacune des méthodes de ces clients permet de faire des requêtes en obtenant directement des objets python : *dict*, *list*, etc. ; n’hésitez pas à en lire la documentation pour savoir ce qui est disponible.

Shortcuts

Vous pouvez simplement obtenir un client pour ces API avec les fonctions `starapi.levelostar()` et `starapi.networkstar()` en fournissant votre clé d’API en paramètre, comme ceci :

```
>>> import starapi
>>> star_busmetro_client = starapi.networkstar('YOUR_API_KEY')
>>> star_velo_client = starapi.levelostar('YOUR_API_KEY')
```

Exemple

Prenons un cas simple, pas à pas : obtenir la liste des districts du réseau Le Vélo Star. Tout d’abord, il faut créer un client pour l’API :

```
>>> import starapi
>>> client = starapi.levelostar('YOUR_KEOLIS_KEY_HERE')
```

Ensuite, obtenir la liste des districts. Cette ligne fait appel HTTP “GET” pour obtenir le résultat :

```
>>> districts = client.get_districts()
```

Enfin, *districts* est une liste (comme l'indique la documentation de `starapi.LeveloStar.get_districts()`) qui se dévoile comme toute les listes : avec une boucle for :

```
>>> for district in districts:
...     print('[%s] %s' % (district.get('id'), district.get('name')))
...
[34] Sud-Gare
[35] Francisco Ferrer-Vern-Poterie
[36] Jeanne d'Arc-Longs Champs-Atalante
[37] Bréquigny
[38] Le Blosne
[39] Villejean-Beauregard
[40] St Martin
[41] Maurepas - Patton
[42] Bourg l'Evêque-La Touche-Moulin du Comte
[43] Thabor - St Héliier
[44] Cleunay-Arsenal-Redon
[45] Centre-Ville
```

Maintenant que vous avez cette liste de “quartier” utilisé par Levelostar, vous pourriez rechercher la liste des stations de l'un de ces quartiers.

Voici donc les stations du quartier “Sud-Gare” :

```
>>> bike_stations = client.get_stations_in_district('Sud-Gare')
>>> for bike_station in bike_stations:
...     print('%s (%s)' % (bike_station.get('name'), bike_station.get('number')))
STE THERESE (71)
MERMOZ (70)
JACQUES CARTIER (61)
GARES (SUD) (45)
CLEMENCEAU (62)
BINQUENAIS (67)
```

La couche basse : handler et python

Si vous préférez maîtriser plus finement le traitement des réponses, vous avez accès à un “handler” de bas niveau muni de votre clé d'API, vous pouvez instancier un *Handler* qui formatera les requêtes pour vous - et qui ne fera que ça d'ailleurs.

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.

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 0x1led2eb0>]
>>> 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*.

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*.

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 votre côté !

Le module starapi

Le module `starapi` est le seul `import` dont vous avez besoin. Il contient plusieurs classes qui permettent d'interroger l'API de la Star.

Les clients

Il existe deux clients : un pour le réseau de vélo, un pour le réseau de bus et de métro.

Shortcut

Les "shortcuts" sont là pour vous aider à faire au plus simple. Ce sont de bons points d'entrée pour obtenir un client d'API et l'utiliser pour faire des requêtes.

`starapi.levelostar` (*key*)

Paramètres`key` (*string*) – La clé d'API fournie par Kéolis.

Type retourné`LeveloStar`

Retourne un client d'API configuré pour Le Velo Star.

Cette fonction retourne un client pour le réseau de vélo, configuré avec le handler par défaut, prêt à l'emploi. Il requiert simplement la clé d'API fournie par Kéolis pour se connecter à son API.

It just works.

```
>>> import starapi
>>> client = starapi.levelostar('YOUR_API_KEY')
>>> stations = client.get_stations()
>>> for station in stations[:4]:
...     print(station.get('name'))
...
ZAC SAINT SULPICE
VILLEJEAN-UNIVERSITE
TURMEL
TNB
```

`starapi.networkstar` (*key*)

Paramètres`key` (*string*) – La clé d'API fournie par Kéolis.

Type retourné`NetworkStar`

Retourne un client d'API configuré pour le réseau Bus & Métro Star.

Cette fonction retourne un client pour le réseau de bus et de métro, configuré avec le handler par défaut, prêt à l'emploi. Il requiert simplement la clé d'API fournie par Kéolis pour se connecter à son API.

Again, it just works.

```
>>> import starapi
>>> client = starapi.networkstar('YOUR_API_KEY')
>>> stations = client.get_subway_stations()
>>> for station in stations[0:4]:
...     print(station.get('name'))
...
Anatole France
Le Blosne
Charles de Gaulle
Gares
```

Client d'API

La classe `StarClient` sert de classe de base aux deux clients d'API qui partagent ainsi des méthodes communes.

class `starapi.StarClient`

Classe de client de base.

`__init__` (*[handler=None]*)

Paramètre`handler` (*Handler*) – starapi handler (optionnel)

Initialise une instance de `StarClient`.

set_handler (*handler*)

Paramètre`handler` (*Handler*) – starapi handler

Définit le handler qui permet d'effectuer les requêtes à l'API du Star.

Le Vélo Star

Pour effectuer des requêtes concernant Le Vélo Star, c'est cette classe qu'il faut utiliser :

class `starapi.LeveloStar`

Cette classe étend la classe de client de base `StarClient`.

get_stations_in_district (*district*)

Paramètre`district` (*string*) – Le nom du district.

Type retourné une liste de dictionnaires représentant des stations.

Retourne la liste des stations du district appelé *district*.

Chaque élément de la liste est un dictionnaire représentant une station.

get_stations_near_position (*longitude, latitude*)

Paramètres

– **longitude** (*string*) – La longitude de la position à proximité.

– **latitude** (*string*) – La latitude de la position à proximité.

Type retourné une liste de dictionnaires représentant des stations.

Retourne la liste des stations à proximité de la position définie par sa *longitude* et sa *latitude*. Jusqu'à 3 stations sont ainsi retournées.

Chaque élément de la liste est un dictionnaire représentant une station.

get_stations_near_station (*number*)

Paramètre`number` (*string*) – L'identifiant de la station à proximité.

Type retourné une liste de dictionnaires représentant des stations.

Retourne la liste des stations à proximité de la station identifiée par *station_number*. Jusqu'à 3 stations sont ainsi retournées.

Chaque élément de la liste est un dictionnaire représentant une station.

get_station (*number*)

Paramètres *number* (*string*) – L'identifiant de la station à recherché.

Type retourné dict représentant la station (si elle existe), *None* sinon.

Lève ValueError – Lorsqu'aucune station n'est trouvée.

Retourne la station recherchée par son numéro d'identifiant.

Generic methods

These methods are considered “generic” as they are equivalent to a call to API with very few parsing of response - and so they are very near the low-level handler.

get_stations (**args, **kwargs*)

Type retourné list

Retourne la liste des stations filtrée en fonction des paramètres.

Chaque élément de la liste est un dictionnaire représentant une station.

Les paramètres sont les mêmes que pour la méthode *starapi.Handler.getbikestations()*.

get_districts ()

Type retourné list

Retourne la liste des districts.

Chaque élément de la liste est un dictionnaire avec les clés *id* et *name*.

Le Réseau de bus et de métro Star

class *starapi.NetworkStar*

Cette classe étend la classe de client de base *StarClient*.

get_cities ()

Type retourné list

Retourne la liste des villes sur lequel est déployé le réseau de transport.

Chaque élément de la liste est un dictionnaire de la forme suivante :

```
{
  'id': '1',
  'name': 'RENNES',
  'district': '26'
}
```

get_districts_in_city (*city*)

Paramètres *city* (*string*) – Nom ou identifiant de la ville.

Type retourné list

Retourne la liste des quartiers d'une ville. Au moment de l'écriture de cette documentation, seule la ville de Rennes (id *1*, name *RENNES*) dispose de quartier identifié, et retournera donc du contenu.

Chaque élément de la liste est un dictionnaire de la forme suivante :

```
{
  'id': '1',
  'name': 'Beauregard'
}
```

get_lines (*[size=21]*)

Paramètre*size* (*string*) – La taille des vignettes souhaitées.

Type retournélist

Retourne une liste de ligne de bus, avec une url de base pour la vignette en fonction de la taille demandée. Chaque élément de la liste est un dictionnaire de la forme suivante :

```
{
  'name': '1',
  'picto': 'L1.png',
  'baseurl': 'http://data.keolis-rennes.com/fileadmin/documents/Picto_lignes/Pictos_lignes'
```

Au moment de l'écriture de cette documentation, les quatres urls de base sont les suivantes :

- Taille 21x21 pixel : http://data.keolis-rennes.com/fileadmin/documents/Picto_lignes/Pictos_lignes_21x21/
- Taille 100x100 pixel : http://data.keolis-rennes.com/fileadmin/documents/Picto_lignes/Pictos_lignes_100x100/
- Taille 300x300 pixel : http://data.keolis-rennes.com/fileadmin/documents/Picto_lignes/Pictos_lignes_300x300/
- Taille 1000x1000 pixel : http://data.keolis-rennes.com/fileadmin/documents/Picto_lignes/Pictos_lignes_1000x1000/

get_lines_small ()

Type retournélist

Retourne le résultat de l'appel à la méthode *get_lines* () pour la taille de vignette 21.

get_lines_medium ()

Type retournélist

Retourne le résultat de l'appel à la méthode *get_lines* () pour la taille de vignette 100.

get_lines_normal ()

Type retournélist

Retourne le résultat de l'appel à la méthode *get_lines* () pour la taille de vignette 300.

get_lines_big ()

Type retournélist

Retourne le résultat de l'appel à la méthode *get_lines* () pour la taille de vignette 1000.

get_alert_for_line (*line*)

Paramètre*line* (*string*) – Le nom de la ligne.

Type retournélist

Retourne la liste des alertes qui touche une ligne.

get_lines_alerts (**args, **kwargs*)

Type retournélist

Retourne la liste des alertes sur le réseau de transport.

Chaque élément de la liste est un dictionnaire de la forme suivante :

```
{
  'title': u'Marché de Corps Nuds',
  'majordisturbance': '0',
  'lines': ['59'],
  'detail': u"Le dimanche matin, jusqu'à 14h environ : Marché à Corps Nuds.",
  'link': None,
  'starttime': '2010-09-27T09:33:30+02:00',
  'endtime': '2013-01-31T00:00:00+01:00'
}
```

get_subway_stations (**args, **kwargs*)

Type retournélist

Retourne une liste de stations de métro selon les critères donnés.

Chaque élément de la liste est de la forme suivante :

```
{
  'lastupdate': '2013-12-08T02:37:43+01:00',
  'name': 'Anatole France',
  'longitude': '-1.687540000',
  'hasPlatformDirection1': '1',
  'floors': '-1',
  'rankingPlatformDirection2': '18',
  'rankingPlatformDirection1': '12',
  'latitude': '48.11812000',
  'hasPlatformDirection2': '1',
  'id': 'ANF'
}
```

get_subway_station (*station*)

Paramètres*station* (*string*) – Identifiant de la station

Type retournédict

Retourne les informations d'une seule station de métro sous forme d'un dictionnaire.

Example

```
>>> import starapi
>>> client = starapi.networkstar('YOUR_STAR_API_KEY')
>>> station = client.get_subway_station('ANF')
>>> print('%s (%s)' % (station.get('name'), station.get('id')))
Anatole France (ANF)
```

get_subway_stations_near_position (*longitude*, *latitude*)

Paramètres

—**longitude** (*string*) – Longitude de la position.

—**latitude** (*string*) – Latitude de la position.

Type retournélist

Retourne une liste de stations (3 maximum) de métro proche de la position donnée.

get_subway_stations_near_station (*station*)

Paramètres*station* (*string*) – L'identifiant de la station.

Type retournélist

Retourne une liste de stations (3 maximum) proche de la station donnée.

Example

```
>>> import starapi
>>> client = starapi.networkstar('YOUR_STAR_API_KEY')
>>> stations = client.get_subway_stations_near_station('ANF')
>>> len(stations)
3
>>> print(', '.join('%s (%s)' % (station.get('name'), station.get('id')) for station in stations))
Pontchaillou (PON), Sainte-Anne (STA), République (REP)
```

get equipments (**args*, ***kwargs*)

Paramètres

—**mode** (*string*) – Le mode d'appel de la requête, soit *all*, *station* ou *id*.

—**station** (*string*) – Mode *station* : l'identifiant de la station de métro.

—**id** (*string*) – Mode *id* : l'identifiant de l'équipement.

Type retournélist

Retourne une liste d'équipements. Chaque élément de la liste est un *dict* représentant un équipement d'une station de métro :

```
{
  'lastupdate': '2013-02-02 05:10:04',
  'tofloor': '0',
  'fromfloor': '-1',
  'platform': '1',
  'station': 'ANF',
  'type': 'ASCENSEUR',
  'id': 'ASC_ANF_1'
}
```

get equipments station (*station*)

Paramètres*station* (*string*) – Identifiant de la station de métro.

Type retournélist

Retourne la liste d'équipements de la station de métro indiquée par son identifiant.

get equipment (*equipment_id*)

Paramètre*equipment_id* (*string*) – Identifiant de l'équipement

Type retournédict

LèveValueError – Lorsqu'aucun équipement n'est trouvé.

Retourne un dictionnaire représentant les données d'un équipement de station de métro.

Voir aussi la méthode *get equipments ()*.

get equipments status (**args, **kwargs*)

Paramètres

—**mode** (*string*) – Le mode d'appel de la requête, soit *all*, *station* ou *id*.

—**station** (*string*) – Mode *station* : l'identifiant de la station de métro.

—**id** (*string*) – Mode *id* : l'identifiant de l'équipement.

Type retournélist

Retourne une liste de status d'équipements. Chaque élément de la liste est un *dict* représentant un status d'équipement d'une station de métro :

```
{
  'lastupdate': '2013-02-02 05:10:04',
  'state': '1',
  'id': 'ASC_ANF_1'
}
```

get equipments status station (*station*)

Paramètres*station* (*string*) – Identifiant de la station de métro.

Type retournélist

Retourne la liste des status d'équipements de la station de métro indiquée par son identifiant.

get equipment status (*equipment_id*)

Paramètre*equipment_id* (*string*) – Identifiant de l'équipement

Type retournédict

LèveValueError – Lorsqu'aucun équipement n'est trouvé.

Retourne un dictionnaire représentant les données du status d'un équipement de station de métro.

Voir aussi la méthode *get equipments status ()*.

Autres éléments et bas niveau

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 fonction

Cette fonction est un décorateur qui permet d'encapsuler l'envoi 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 {}
```

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()*.

```
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*.

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 Héliier</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’.
- number** (*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*)

Paramètresnetwork (*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 Stac
          <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_1
    <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’ (default), ‘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’ (default), ‘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>
    </data>
  </answer>
</opendata>

```

```

        </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’ (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’, 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>
    </data>
  </answer>
</opendata>

```

```

        </district>
        <!-- another "district" tags -->
    </data>
</answer>
</opendata>

```

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

Paramètres

- mode** (*string*) – Mode parmi ‘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+02:00</departure>
          <departure accurate="0" headsign="Champs Blancs">2012-08-25T15:16:00+02:00</departure>
        </departures>
      </stopline>
      <!-- another "stopline" tags -->
    </data>
  </answer>
</opendata>

```

Utilitaires

`starapi.parse_command_response` (*response*)

Paramètres*response* – Un objet response fourni par le module *requests*.

Type retournéUn noeud xml représentant le noeud ‘/opendata/answer/data’.

Lève`StarException` – En cas d’erreur, avec le code d’erreur fourni par la réponse.

Retourne le noeud xml */opendata/answer/data* après transformation en noeuds XML de la réponse texte.

`starapi.node_to_dict` (*node*)

Paramètres*node* (un noeud xml tel qu’il est représenté par *lxml* ou par *ElementTree*.) – Le noeud xml à transformer en dictionnaire.

Retourne l’équivalent en dictionnaire d’un noeud xml.

Exemple avec le XML suivant :

```
<district>
  <id>34</id>
  <name>Sud-Gare</name>
</district>
```

Le résultat d'un appel à cette méthode sera :

```
{'id': '34', 'name': 'Sud-Gare'}
```

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.

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.

Installation

Première étape : récupérer les sources du projet. C'est facile, c'est disponible sur le [dépot Bitbucket](https://bitbucket.org/exirel/star_api), soit un tag soit la dernière version de développement, par exemple comme ceci :

```
hg clone https://bitbucket.org/exirel/star_api
```

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

```
cd star_api
python setup.py install
```

Ceci doit installer le module *starapi* ainsi que sa dépendance principale, le module *requests*. Il est conseillé d'avoir le module *lxml* d'installé pour des questions de performance, mais le client fonctionne aussi bien sans (quoi que peut-être un peu moins véloce sur le traitement du XML).

Tables d'index

- genindex
- modindex
- search

S

starapi, 5

Symbols

`__init__()` (méthode `starapi.StarClient`), 6

A

`api_command()` (dans le module `starapi`), 11

C

`call_api()` (méthode `starapi.Handler`), 11

G

`get_alert_for_line()` (méthode `starapi.NetworkStar`), 8

`get_cities()` (méthode `starapi.NetworkStar`), 7

`get_districts()` (méthode `starapi.LeveloStar`), 7

`get_districts_in_city()` (méthode `starapi.NetworkStar`), 7

`get_equipment()` (méthode `starapi.NetworkStar`), 10

`get_equipment_status()` (méthode `starapi.NetworkStar`), 10

`get equipments()` (méthode `starapi.NetworkStar`), 9

`get equipments_station()` (méthode `starapi.NetworkStar`), 10

`get equipments_status()` (méthode `starapi.NetworkStar`), 10

`get equipments_status_station()` (méthode `starapi.NetworkStar`), 10

`get_lines()` (méthode `starapi.NetworkStar`), 7

`get_lines_alerts()` (méthode `starapi.NetworkStar`), 8

`get_lines_big()` (méthode `starapi.NetworkStar`), 8

`get_lines_medium()` (méthode `starapi.NetworkStar`), 8

`get_lines_normal()` (méthode `starapi.NetworkStar`), 8

`get_lines_small()` (méthode `starapi.NetworkStar`), 8

`get_station()` (méthode `starapi.LeveloStar`), 7

`get_stations()` (méthode `starapi.LeveloStar`), 7

`get_stations_in_district()` (méthode `starapi.LeveloStar`), 6

`get_stations_near_position()` (méthode `starapi.LeveloStar`), 6

`get_stations_near_station()` (méthode `starapi.LeveloStar`), 6

`get_subway_station()` (méthode `starapi.NetworkStar`), 9

`get_subway_stations()` (méthode `starapi.NetworkStar`), 8

`get_subway_stations_near_position()` (méthode `starapi.NetworkStar`), 9

`get_subway_stations_near_station()` (méthode `starapi.NetworkStar`), 9

`getbikedistricts()` (méthode `starapi.Handler`), 13

`getbikestations()` (méthode `starapi.Handler`), 13

`getbusnextdepartures()` (méthode `starapi.Handler`), 19

`getcities()` (méthode `starapi.Handler`), 18

`getcitydistricts()` (méthode `starapi.Handler`), 18

`getdistrict()` (méthode `starapi.Handler`), 12

`getequipments()` (méthode `starapi.Handler`), 15

`getequipmentsstatus()` (méthode `starapi.Handler`), 15

`getlines()` (méthode `starapi.Handler`), 14

`getlinesalerts()` (méthode `starapi.Handler`), 14

`getmetrostations()` (méthode `starapi.Handler`), 16

`getmetrostationsstatus()` (méthode `starapi.Handler`), 16

`getpos()` (méthode `starapi.Handler`), 17

`getrelayparks()` (méthode `starapi.Handler`), 17

`getstation()` (méthode `starapi.Handler`), 12

H

`Handler` (classe dans `starapi`), 11

K

`KEOLIS_VERSION_1` (dans le module `starapi`), 11

`KEOLIS_VERSION_2` (dans le module `starapi`), 11

`KEOLIS_VERSION_2_1` (dans le module `starapi`), 11

L

`LeveloStar` (classe dans `starapi`), 6

`levelostar()` (dans le module `starapi`), 5

N

`NetworkStar` (classe dans `starapi`), 7

`networkstar()` (dans le module `starapi`), 5

`node_to_dict()` (dans le module `starapi`), 19

P

`parse_command_response()` (dans le module `starapi`), 19

S

[set_handler\(\)](#) (méthode `starapi.StarClient`), 6

[starapi](#) (module), 5

[StarClient](#) (classe dans `starapi`), 6