

Accéder au service Vélib en REST

Stéphane Bortzmeyer

<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 Septembre 2007. Dernière mise à jour le 3 Octobre 2007

<http://www.bortzmeyer.org/velib-rest.html>

Le service Vélib' de location de vélos à Paris est un grand succès sur l'Internet. D'innombrables sites le discutent et proposent des "mashups" à partir des données Vélib'. En effet, le service Vélib' a une API REST qui permet facilement de trouver des informations.

Les grands succès du Web sont souvent les services qui exposent une API, permettant ainsi au reste du monde de bâtir des services à valeur ajoutée. Vélib' a une API qui permet de connaître l'état des stations (combien de vélos libres ? combien d'emplacements libres ?). On peut alors assez facilement construire des services indiquant cet état, ou faisant des statistiques sur le long terme, par exemple avec des outils comme RRDtool. Voir par exemple <http://www.roulib.fr/>, <http://v.mat.cc/> ou bien <http://myvelib.free.fr/>. On peut aussi trouver un amusant gadget Netvibes (qui marche aussi avec le service personnalisé de Google) en <http://www.ghusse.com/informatique/widget-velib-11-fonctionne-so-309/>.

Ainsi, pour récupérer l'état d'une station, il faut interroger <http://www.velib.paris.fr/service/stationdetails/NNN> où NNN est le numéro de la station. Essayons avec un client REST classique, curl :

```
% curl http://www.velib.paris.fr/service/stationdetails/14024
<?xml version="1.0" encoding="UTF-8"?>
<station>
  <available>16</available>
  <free>8</free>
  <total>24</total>
  <ticket>1</ticket>
```

16 vélos sont donc disponibles.

Quant à <http://www.velib.paris.fr/service/carto>, il permet de récupérer la liste des stations.

Où est documentée cette API? Eh bien justement, il semble qu'elle ne l'est nulle part. Il faut apparemment l'extraire par rétro-ingénierie du code source Javascript des pages de <http://www.velib.paris.fr/>. Par exemple, la page http://www.velib.paris.fr/les_stations/trouver_une_station appelle le script Javascript `gmaps_search_station.js` qui contient :

```
request.open("GET", stationDetailsUrl + "/" + this.number,
  true);
```

et, un peu plus loin, l'explication des éléments XML retournés :

```
instance.availableBikes = xmlDoc.getElementsByTagName('available')[0].firstChild.nodeValue;
instance.freeCapacity = xmlDoc.getElementsByTagName('free')[0].firstChild.nodeValue;
instance.totalCapacity = xmlDoc.getElementsByTagName('total')[0].firstChild.nodeValue;
```

Merci à Mathieu Arnold pour ses explications sur ce point.

Voyons maintenant un client pour accéder à ce service. Je l'écris en C pour montrer que C ne signifie pas forcément manipulations de bas niveau des prises et travail XML difficile. Pour le réseau, j'utilise l'excellente libcurl (qui gère tous les détails, y compris IPv6, SSL, etc) et pour le XML la non moins excellente libxml2. Le résultat est disponible en (en ligne sur <http://www.bortzmeyer.org/files/get-station.c>). Il se compile ainsi :

```
% gcc -Wall `curl-config --cflags` `xml2-config --cflags` -c get-station.c
% gcc -o get-station get-station.o `curl-config --libs` `xml2-config --libs`
```

et s'utilise ainsi (la station 15068 est située boulevard Victor près de la gare du RER) :

```
% ./get-station 15068
Available bikes - Free slots - Total slots  at station 15068
                14             4             20
```

(Le total ne correspond pas car un emplacement peut être en maintenance.)

Un autre programme équivalent a été développé en Haskell, en utilisant la bibliothèque standard `Network.HTTP` et le paquetage `HaXML` (<http://www.cs.york.ac.uk/fp/HaXml/>) pour traiter l'XML. Ce programme (en ligne sur <http://www.bortzmeyer.org/files/get-station.hs>) prend les mêmes arguments et affiche les mêmes résultats que le programme en C.

On peut appliquer la même méthodologie au service Vélov de Lyon. Si on regarde le source HTML des pages Web de <http://velov.grandlyon.com/>, on trouve la mention du script Javascript <http://velov.grandlyon.com/velov/zhp/js/velov.js> qui, une fois téléchargé et étudié, nous indique que l'API est très proche de celle de Paris (ce qui n'est pas étonnant, c'est la même société). Développons donc un client REST, cette fois en Python. Ce programme (en ligne sur <http://www.bortzmeyer.org/files/get-station.py>) s'utilise de la même façon :

<http://www.bortzmeyer.org/velib-rest.html>

```
% python get-station.py 1023
Available bikes - Free slots - Total slots  at station 1023 - CROIX ROUSSE / PERFETTI
              7             11             20
```

Le service déployé à Marseille, Le vélo (<http://www.levelo-mpm.fr/>), est techniquement identique à celui de Paris. On peut utiliser les mêmes programmes, avec l'URL <http://www.levelo-mpm.fr/service/stationdetails/NNNN>.

Testons un dernier langage sur ce service, celui du shell Unix. Comme le shell n'a pas de mécanisme incorporé pour analyser le XML, nous utiliserons un script XSLT :

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:output method="text"/>

  <xsl:param name="num"/>

  <xsl:template match="/">
    <xsl:text>Statistiques pour la station </xsl:text>
    <xsl:value-of select="$num"/>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="available">
    <xsl:text>Vélos libres : </xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="free">
    <xsl:text>Bornes libres : </xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="total">
    <xsl:text>Nombre total de bornes : </xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="ticket">
    <xsl:text>Bornes en panne : </xsl:text>
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```

et le script contient :

```
#!/bin/sh

# Recuperer des informations sur une station du service de vélos en
# libre-service Le Velo a Marseille

URL="http://www.levelo-mpm.fr/service/stationdetails"
STYLESHEET="format.xml"
set -e

if [ "$1" = "" ]; then
  _____
  http://www.bortzmeyer.org/velib-rest.html
```

```
    echo "Usage: $0 station-number" > /dev/stderr
    exit 1
fi

num=$1
output=`mktemp /tmp/levelo-$num.XXXXX`

curl --silent --output $output $URL/$num
xsltproc --stringparam num $num $STYLESHEET $output
rm -f $output
```

et s'utilise ainsi (sur la station la plus proche du Vieux Port) :

```
% ./get-station.sh 1156
Statistiques pour la station 1156
Vélos libres : 3
Bornes libres : 4
Nombre total de bornes : 8
Bornes en panne : 1
```

Ce programme marcherait probablement sans problème avec le service Vélô à Toulouse, qui utilise encore le même logiciel (on trouve même /marseille dans les URL de http://www.velo.toulouse.fr/les_stations/trouver_une_station).

D'autres systèmes de location de vélos en libre-service utilisent apparemment une interface bien différente. Ainsi, le service BIXI de Montréal fonctionne en publiant un seul fichier, <https://profil.bixi.ca/data/bikeStations.xml>, qui contient à la fois l'information sur les stations et l'état actuel de leur occupation. (Merci à Gérôme Sauve pour avoir attiré mon attention sur ce service et à Pascal Courtois pour son analyse de l'application avec Firebug.)

Le programme « montréalais » (en ligne sur <http://www.bortzmeyer.org/files/bixi.py>) est également en Python mais avec une autre bibliothèque (d'API très semblable à ElementTree), lxml (<http://codespeak.net/lxml/>). Voici des exemples d'utilisation :

```
% python bixi.py 'Cartier / des Carrières'
Cartier / des Carrières: 0 bikes, 7 empty docks

% python bixi.py 'Mozart / St-Laurent'
Mozart / St-Laurent: 4 bikes, 3 empty docks

% python bixi.py 'Holt / Fullum'
Holt / Fullum: not yet installed
```