tpg Open data

Real time information API

User Guide

Version 1.2

Version	Date	Comments
1.0	Aug. 27th, 2013	Creation
1.1	Sept. 9th, 2013	Update by GFI
1.2	Sept. 18th, 2015	Update by tpg

Table of contents

1. Purpose of the document

This document described the functions offered by tpg real time information API. It indicates query expected format to call web services, and details responses' format.

2. General notes

2.1 Terms of Use

To access real time information API, you need to agree to the Terms & Conditions.
Those Terms of Use are available for consultation on: http://data.tpg.ch.

2.2 Access to tpg API

To consume real time information API, you'll need a valid API key to be sent each time you call the web service. To ask for a key please consult tpg website: http://data.tpg.ch.

You will also need to specify the API version in each call to the web service.

Real time information is served by the server: http://prod.ivtr-od.tpg.ch.

Please find some examples of queries hereafter:
Commercial stop list:
http://prod.ivtr-od.tpg.ch/v1/GetStops?key=xxxx

Traffic information: http://prod.ivtr-od.tpg.ch/v1/GetDisruptions?key=xxxx

Next departures:
http://prod.ivtr-od.tpg.ch/v1/GetNextDepartures?key=xxxx&stopCode=xxxx
where xxxx are input parameters.

2.3 Output format

Responses are available in JSON and XML formats.

http://prod.ivtr-od.tpg.ch/v1/GetDisruptions.json?key=xxxxx
http://prod.ivtr-od.tpg.ch/v1/GetDisruptions.xml?key=xxxxx

Note: If no format is specified in the query, response is XML-formatted by default.

Each response is time and date stamped; the timestamp is in extended ISO format indicating the date, hour, minutes and time zone.
Example: <timestamp>2013-08-16T11:58:24+0200</timestamp>: response is dated on August 16th, 2013 at 11:58:24, time zone GMT+2.

Encoding is UTF-8.

2.4 Data source

Data comes from tpg SAEIV system (passenger information & operational support system).

2.5 Useful vocabulary

Please find some vocabulary that you'll find in functions' detailed description:

2.5.1 Difference between Physical stop and Commercial stop

A commercial stops groups one to several physical stops of same name together.
For example, « Gare Cornavin » commercial stop includes several physical stops located around the railway station. Each of these physical stops can be served by different bus lines.

2.5.2 Operating day

An operating day corresponds to one day for tpg operational teams. It starts at 4 o' clock AM and ends at 3.30 AM the calendar day after.

2.6 Data availability time

Between 3:30 AM and 4:15 PM, API data is not available, because it corresponds to data initialization in SAEIV system.

3. Getting stops

3.1 Getting commercial stops: GetStops

3.1.1 Description

This function returns the characteristics (code, name, lines, and directions) of one or several commercial stops. These stops are selected thanks to various parameters.
Information remains valid all the current operating day long.
Please note that the stops you get are all stops from tpg network for which there is at least one departure time in current operating day. As a

consequence, if no line serves a stop on Sunday, GetStops API called on Sunday will not return this stop.

3.1.2 Input

Name	Description	Mandatory?	Expected value
stopCode	commercial stop codes	no	String : List of comma (,)-separated commercial stop codes
stopName	Part of commercial stop name	no	String : Substring from the name
line	name of the line for which we want to know served stops	no	String : Line name
latitude	User coordinates	no	Float : Latitude in WGS84 coordinate system
longitude	User coordinates	no	Float : Longitude in WGS84 coordinate system

3.1.3 Possible uses

- **no input parameter**: the functions returns commercial stops for current operating day sorted by increasing stop code (stopCode).
e.g. : GetStops

- **stop codes only (stopCode):** the functions returns the list of all commercial stops for which stopCode is in the input list, sorted by increasing stopCode.

e.g.: GetStops?stopCode=CVIN,BHET

- **stopName only:** the function returns the list of all commercial stops whose name contains the substring indicated in input, sorted by increasing stopCode.

e.g.: GetStops?stopName=gare

- line only : the function returns the list of commercial stops served by the line indicated in input, sorted by increasing stopCode.
  e.g.: GetStops?lineCode=12

- latitude and longitude : the function returns the commercial stops located in a range of 500 meters from the spot specified in input, sorted by increasing distance to the spot.

  e.g.: GetStops?latitude=46.218176&longitude=6.146445

Those filters are exclusive. If too many parameters are sent in input, you will get Error #12: Too many parameters (please find error details in Error codes chapter later in this document).

3.1.4 Output

XML :

```
<stops>
 <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>

 <stops>
        <stop>
                <stopCode>CVIN</stopCode>
                <stopName>Gare Cornavin</name>

                <connections>
                        <connection>
                                <lineCode>1</lineCode>
                                <destinationName>Petit-Bel-Air</destinationName>
                                <destinationCode>PETIT-BEL-AIR</destinationCode>
                        </connection>
                        <connection>
                                <lineCode>1</lineCode>
                                <destinationName>Jar.-Botanique</destinationName>
                                <destinationCode>JAR. BOTANIQUE</destinationCode>
                        </connection>
                        …
        </connections>

        ...
        <distance>103</distance>
        </stop>
        <stop>
                …
```

```
        </stop>
        …
  </stops>

  ...
</stops>
```

JSON :

```
{
    stops:{
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
        stops:[
            {
                stopCode: "CVIN",
                stopName:"Gare Cornavin",
                distance:103
                connections:[
                    {
                        lineCode:1,
                        destinationName:"Petit-Bel-Air",
                        destinationCode:"PETIT-BEL-AIR"
                    },
                {
                        lineCode:1,
                        destinationName:"Jar.-Botanique",
                        destinationCode:"JAR. BOTANIQUE"
                    },
                    ...
                ]
            },
              ...
        ]
    }
}
```

Note : *distance* property is available only if coordinates (latitude, longitude) are specified in input.

If stopCode or lineCode specified in input is unknown, response is empty (only contains the timestamp) :

XML :

```
<stops>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
</stops>
```

JSON:

```
{ stops:{
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
            }
    }
```

3.2 Getting physical stops : GetPhysicalStops

3.2.1 Description

This function returns the characteristics (code, coordinates, name, lines, and directions) of one or several physical stops selected thanks to various parameters.

3.2.2 Input

Nom	Description	Mandatory ?	Expected value
stopCode	commercial stop code(s)	no	String : List of comma (,)-separated commercial stop codes
stopName	Part of commercial stop name	non	String : Substring from the name

3.2.3 Possible uses

- **no input parameter** : the function returns the list of all physical stops for the current operating day, sorted by increasing stopCode .
  e.g.: GetPhysicalStops

- **stopCode only** : the functions returns physical stops whose commercial code is included in the input list, sorted by increasing stopCode.
  e.g.: GetPhysicalStops?stopCode=CVIN,BHET

- **name only** : the functions returns physical stops whose name contains the substring indicated in input, sorted by increasing stopCode.
  e.g.: GetPhysicalStops?name=gare

Those filters are exclusive : if stopCode and stopName are specified in input, you will get error #12 : Too many parameters.

3.2.4 Output

XML :

```
<stops>
  <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>

  <stops>
        <stop>
              <stopCode>CVIN</stopCode>
              <stopName>Gare Cornavin</name>

              <physicalStops>
                    <physicalStop>
                          <physicalStopCode>CVIN01</physicalStopCode>
                          <stopName>Gare Cornavin</name>

                          <connections>
                                <connection>
                                      <lineCode>1</lineCode>
                                      <destinationName>Petit-Bel-
Air</destinationName>
                                      <destinationCode>PETIT-BEL-
AIR</destinationCode>
                                </connection>
                                …
                          </connections>

                          <coordinates>
                                <referential>WGS84</referential>
                                <latitude>46.21260795937985</latitude>
                                <longitude>6.142041223117598</longitude>
                          </coordinates>

                    </physicalStop>
                    <physicalStop>
                          <physicalStopCode>CVIN02</physicalStopCode>
                          <stopName>Gare Cornavin</name>

                          <coordinates>
                                <referential>WGS84</referential>
                                <latitude>46.21260795937985</latitude>
                                <longitude>6.142041223117598</longitude>
                          </coordinates>
```

```
                        <connections>
                            <connection>
                                <lineCode>1</lineCode>
                                <destinationName>Jar.-
Botanique</destinationName>
                                <destinationCode>JAR.
BOTANIQUE</destinationCode>
                            </connection>
                            …
                        </connections>
                    </physicalStop>
                    …
                </physicalStops>
            </stop>
            <stop>
                ...
            </stop>
            ...
    </stops>
</stops>
```

JSON :

```
{
    stops:{
        timestamp:”YYYY-MM-DDThh:mm:ssTZD”,
        stops:[
            {
                stopCode:“CVIN”,
                stopName:”Gare Cornavin”,
                physicalStops :[
                    {
                        physicalStopCode:"CVIN01",
                        stopName:”Gare Cornavin”,

                        coordinates:{
                            referential:”WGS84”,
                            latitude: 46.21260795937985,
                            longitude: 6.142041223117598
                        },
                        connections:[
                        {
                            lineCode:1,
                            destinationName:”Petit-Bel-Air”,
                            destinationCode:”PETIT-BEL-AIR”
                        },
                    {
                            lineCode:1,
                            destinationName:”Jar.-Botanique”,
                            destinationCode:”JAR. BOTANIQUE”
                        },
                        …
                        ]

                    },
                    …
                ]
            },
              …
        ]
    }
}
```

If a stopCode specified in input is unknown, or if the string in input matches no stop name, response is empty (contains only the timestamp) :

XML :

```
<stops>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
</stops>
```

JSON:

```
{ stops:{
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
          }
    }
```

4. Getting next departures

4.1 Getting departures within the hour: GetNextDepartures

4.1.1 Description

This function returns next departures' characteristics depending on several parameters. Departures are sorted by increasing departure time.

4.1.2 Precisions

4.1.2.1 Waiting time

The function returns next departures within the current hour, meaning departures with waiting times no more than 59 minutes 59 seconds.
A departure can be considered reliable (<reliability>F</reliability>, F for "*fiable*" meaning Reliable in French) or approximate (<reliability>T</reliability>, T for "*Theoretical*").
If for a given line and direction, next departure is in more than one hour, the function returns « >1h » : <waitingTime>>1h</waitingTime>.
If for a given line and direction, there were some departures in current operating day but there is no departure anymore, the function returns « no more» : <waitingTime>no more</waitingTime>.
If waitingTime's value is zero , the vehicle is said to be « approaching ». A waitingTime of zero can be associated to a waitingTimeMillis of zero, strictly positive or strictly negative.

4.1.2.2 PRM equipment

Stop attribute <characteristics> indicates if the vehicle is accessible to Persons with Reduced Mobility (PRM).

4.1.2.3 Disruptions and deviations

If a deviation concerns a stop usually served by the vehicle, the response indicates that deviation (details later in the document).
If disruptions impact the trip, the response includes a Disruption object (details later in the document).

4.1.3 Input

Name	Description	Mandatory ?	Expected value
stopCode	commercial stop code	yes	String : stop code
departureCode	departure code to identify which	no	Integer : departure code

	connections are possible (details later in the document)		
linesCode	filter by line(s)	no	String : List of comma (,)-separated lines
destinationsCode	filter by direction(s)	no	String : List of comma (,)-separated directions

4.1.4 Possible uses

- **stopCode only** : the function returns the list of next within the hour for the commercial stop specified in input.
  e.g.: GetNextDepartures?stopCode=CVIN

- **stopCode and departureCode** : the function returns the list of next departures within the hour for the commercial stop specified in input, and for each of these departures we compute the waiting time to the connection specified in input thanks to its departureCode.
  e.g.: GetNextDepartures?stopCode=WTC0&departureCode=15468 :

Illustration : A passenger wants to join *Blandonnet* stop from *De Joinville* stop. To do this, he wants to :

- take line number 10 *De Joinville* to *WTC* (stopCode WTC0, departureCode=15468 that day)
- then change bus at *WTC* to take line number Y going to *Val-Thoiry* direction
- and leaves the bus at *Blandonnet* stop.

His bus is expected in *WTC* in 28 minutes' time. All departures from *WTC* expected before that time have therefore strictly negative connectionWaitingTime, meaning that those connexions are *not* possible for our passenger.

- **linesCode and destinationsCode** : the function returns the departures corresponding to the chosen filters

4.1.5 Output

XML :

```
<nextDepartures>
  <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>

  <stop> (reminder of stop characteristics ; same stop object as  GetStops function…)
        ...
  </stop>

  <departures>
        <departure>
              <departureCode>27439</departureCode>
              <waitingTime>0</waitingTime>
              <waitingTimeMillis>-20000</waitingMillis>

              <connectionWaitingTime>12</connectionWaitingTime>

              <connection>
                    <lineCode>12</lineCode>
                    <destinationName>Palettes</destinationName>
                    <destinationCode>PALETTES</destinationCode>
              </connection>

              <reliability>F</reliability>
              <characteristics>PMR</characteristics>

              <disruptions>
                    <disruption>
                          <disruptionCode>13306</disruptionCode>
                    <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
                    <place>77 Route de Chêne</lieu>
                    <nature>Suite accident</nature>
                    <consequence>Retard de 15 minutes</consequence>
                          <stopName>Grange Canal</stopName>
                    </disruption>
                    …
              </disruptions>

              <deviation>
                          <deviationCode>O</deviationCode>
              </deviation>
       <departure>
       …
```

```
  </departures>
</nextDepartures>
```

JSON :

```
{
    nextDepartures:{
        timestamp:”YYYY-MM-DDThh:mm:ssTZD”,
        stop:{ (same stop object as GetStops function) },
        departures:[
            {
                departureCode:27439,
                waitingTime:0,
                waitingTimeMillis:-20000,
                connectionWaitingTime:12,
                connection:{
                    lineCode:12,
                    destinationName:”Palettes”,
                    destinationCode:”PALETTES”
                },
                reliability:”F”,
                characteristics:”PMR”,
                deviation:{
            deviationCode:"O"
            }
                disruptions:{
                    disruptionCode:”137907”,
             timestamp:”YYYY-MM-DDThh:mm:ssTZD”,
                    place:”77 Route de Chêne”,
                    nature:”Suite accident”,
                    consequence:”Retard de 15 minutes”
            stopName:”Grange Canal”,
                }
            },
            ...
        ]
    }
}
```

Notes :

- *connectionWaitingTime* property is sent only if a departureCode is specified in input.
- *deviation* property of a departure is sent only if there is a deviation.

- *disruptions* property of a departure is sent only if a disruption concerns its line.

If the stopCode specified in input is unknown, the response is empty (only contains a timestamp):

XML :

```
<nextDepartures>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
</ nextDepartures >
```

JSON:

```
{ nextDepartures:{
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
          }
    }
```

If the departureCode specified in input is unknown, it is simply ignored and ConnectionWaitingTime attributes are absent from the API answer.

If triplet (stopCode, lineCode, destinationCode) does not exist, the answer provides stop's characteristics but no departure :

XML :

```
        <nextDepartures>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <stop> … </stop> (stop object, described earlier in the document)
        </ nextDepartures >
```

JSON:

```
{ nextDepartures:{
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
          stop { (stop object, described earlier in the document)
          }
          }
    }
```

4.2 Getting next departures : GetAllNextDepartures

4.2.1 Description

This function returns all next departures until the end of current operating day, for a given line and direction. Departures are sorted by increasing departure time.
Note : Deviations, disruptions and reliability flag are not provided.

4.2.2 Input

Nom	Description	Mandatory ?	Expected value
stopCode	commercial stop code	yes	String : stop code
lineCode	line code	yes	String : line code
destinationCode	direction code	yes	String : direction code

4.2.3 Possible uses

- **stopCode, lineCode and destinationCode** : the function returns the list of all next departures until the end of the current operating day, for the line and direction provided in input.
  e.g.:
  GetAllNextDepartures?stopCode=CVIN&lineCode=12&destination=BHET

4.2.4 Output

XML :

```
<nextDepartures>
  <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>

  <stop> (same stop object as GetStops function)
        ...
  </stop>

  <departures>
        <departure>
              <departureCode>27439</departureCode>
              <timestamp>2013-08-16T13:58:49+0200</timestamp>
              <waitingTimeMillis>-20000</waitingMillis>
              <waitingTime>0</waitingTime>

              <connection>
                    <lineCode>12</lineCode>
                    <destinationName>Palettes</destinationName>
                    <destinationCode>PALETTES</destinationCode>
              </connection>

              <reliability>F</reliability>
        <departure>
        …
  </departures>
  ...
</nextDepartures>
```

JSON :

```
{
    nextDepartures:{
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
        stop:{ (same stop object as GetStops function) },
        departures:[
              {
                    departureCode:27439,
                    timestamp":"2013-08-16T13:57:31+0200"
                    waitingTimeMillis:-20000,
                waitingTime:0,
                    connection:{
                          lineCode:12,
                          destinationName:"Palettes",
                          destinationCode:"PALETTES"
```

```
                    },
                    reliability:”F”,
                },
                ...
        ]
    }
}
```

If triplet (stopCode, lineCode, destinationCode) does not exist, the answer provides stop’s characteristics but no departure:

XML :

```
<nextDepartures>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <stop> … </stop> (stop object, see earlier in the document)
</ nextDepartures >
```

JSON:

```
{ nextDepartures:{
        timestamp:”YYYY-MM-DDThh:mm:ssTZD”,
        stop { (stop object, see earlier in the document)
          }
          }
    }
```

5. Getting a Thermometer

5.1 Commercial stops thermometer : GetThermometer

5.1.1 Description

This function returns the sequence of commercial stops representing the trip whose departureCode is specified in input. This ordered sequence of couples (stop, waiting time) is called « commercial stops thermometer».
The response also indicates deviations and disruptions if any.

Example : Line number 8 corresponds to the trip *Veyrier-Douane / OMS*. One wants to know the commercial stops sequence from stop *IUT* to final stop *OMS*.
Input is therefore the departure code corresponding to line 8 trip from *IUT* to *OMS* we are interested in (and obtained thanks to GetNextDepartures web service).
The response provides all commercial stops and departure times from *Veyrier-Douane* to *OMS*.
The tag <visible>True</visible> indicates that a commercial stop is located between *IUT* and *OMS*.

5.1.2 Input

Nom	Description	Mandatory ?	Expected value
departureCode	departure code indicating which trip we are interested in	yes	Integer : departure code of a vehicle

Please note that input parameter is a departure code, offering precise location on a thermometer even if path is circular or of a stop is served twice on a same trip.

5.1.3 Possible use

- **departureCode** : the function returns the list of commercial stops with corresponding waiting times. It also provides deviation and disruption information if any.
  e.g.: GetThermometer?departureCode=156879

5.1.4 Output

XML :

```
<thermometer>
  <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>

  <stop> (same stop object as function GetStops)
        ...
  </stop>

  <lineCode>12</lineCode>
  <destinationName>Palettes</destinationName>
  <destinationCode>PALETTES</destinationCode>

  <steps>
        <step>
              <departureCode>123324</departureCode>
              <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
              <stop>
                    (same stop object as function GetStops)
              </stop>
              <reliability>F</reliability>
              <arrivalTime>12</arrivalTime>
              <deviation>false</deviation>
              <visible>false</visible>
        </step>
        ...
        <step>
              <departureCode>123339</departureCode>
              <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
              <stop>
                    (same stop object as GetStops function)
              </stop>
              <reliability>F</reliability>
              <arrivalTime>15</arrivalTime>
              <deviation>true</deviation>
              <deviationCode>475</deviationCode>
              <visible>true</visible>
        </step>
  </steps>

  <disruptions>
        … (same object as function GetNextDepartures)
  </disruptions>

  <deviations>
        <deviation>
              <deviationCode>475</deviationCode>
```

```
                <startStop>
                    (same stop object as function GetStops)
                </startStop>
                <endStop>
                    (same stop object as function GetStops)
                </endStop>
            </deviation>
            …
    </deviations>


</thermometer>
```

JSON :

```
{ thermometer :
  {
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
        stop:{ (same stop object as function GetStops) },

        lineCode : 3,
        destinationName : "Gardiol",
        destinationCode : "GARDIOL",

        steps : [
                {
                        stop" : { (same stop object as function GetStops) },
                        departureCode : 79934,
                        timestamp : "2013-08-14T16:34:00+0200",
                        arrivalTime : 284,
                        reliability : "T",
                        deviation : true,
                        deviationCode : 475
                        visible : false
                },
                {
                        stop : { (same stop object as function GetStops ) },
                        departureCode : 24568,
                        timestamp : "2013-08-14T16:34:00+0200",
                        arrivalTime : 284,
                        reliability : "T",
                        deviation : false,
                        visible : true
                },
                ...
      ]
```

```
        disruptions : { (same object as function GetNextDepartures)
            deviations : [
                {
                        deviationCode:475,
                        startStop:{( same stop object as function GetStops) },
                        endStop:{ (same stop object as function GetStops) }
                }
        ],

    }
}
```

Note :

- *deviationCode* attribute is present only if a deviation exists for that departure.
  Deviation detail (start stop, end stop) is provided in *deviations* object.
- *disruptions* attribute is present only if a disruption concerns the line.

5.2 Getting physical stops thermometer : GetThermometerPhysicalStops

5.2.1 Description

This function returns the sequence of physical stops representing the trip whose departureCode is specified in input. This ordered sequence of couples (physical stop, waiting time) is called “physical stops thermometer”. The response also indicates deviations and disruptions if any.

5.2.2 Input

Nom	Description	Mandatory ?	Expected value
departureCode	departure code indicating which trip we are interested in	yes	Integer : departure code of a vehicle

5.2.3 Possible use

- **departureCode** : the function returns the list of physical stops with corresponding waiting times. It also provides deviation and disruption information if any.
  e.g.: GetThermometerPhysicalStops?horaireRef=156879

5.2.4 Output

XML :

```
<thermometer>
  <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
  <stop> (same object as GetStops function)
        ...
  </stop>
  <lineCode>12</lineCode>
  <destinationCode>PALETTES</destinationCode>
  <destinationName>Palettes</destinationName>

  <steps>
        <step>
              <departureCode>123324</departureCode>
              <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
```

```
            <stop> (same object as GetStops function)
                ...
            </stop>
            <physicalStop>
                (same object as GetPhysicalStops function)
            </physicalStop>

            <deviation>true</deviation>
            <deviationCode>150</deviationCode>

            <reliability>F</reliability>
            <arrivalTime>15</arrivalTime>
            <visible>false</visible>
        </step>
        ...
        <step>
            <departureCode>123324</departureCode>
            <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
            <stop> (same object as GetStops function)
                ...
            </stop>
            <physicalStop>
                (same object as GetPhysicalStops function)
            </physicalStop>

            <deviation>false</deviation>
            <reliability>F</reliability>
            <arrivalTime>15</arrivalTime>
            <visible>true</visible>
        </step>
</steps>


<disruptions>
        … (same object as GetNextDepartures function)
</disruptions>

<deviations>
        <deviation>
            <deviationCode>475</deviationCode>
            <startStop>
                (same object as GetStops function)
            </startStop>
            <endStop>
                (same object as GetStops function)
```

```
            </endStop>
        </deviation>
        …
  </deviations>

</thermometer>
```

JSON :

```
{ thermometer :
  {
        timestamp:"YYYY-MM-DDThh:mm:ssTZD",
        stop:{ (same object as GetStops function) },
        lineCode : 3,
        destinationName : "Gardiol",
        destinationCode : "GARDIOL",

        steps : [
              {
                    physicalStop " : { (same object as GetPhysicalStops function) },
                                departureCode : 79934,
                    timestamp : "2013-08-14T16:34:00+0200",
                    arrivalTime : 284,
                    reliability : "T",
                    deviation : true,
                    deviationCode : 150
                    visible : false
              },
              {
                    physicalStop " : { (same object as GetPhysicalStops function) },
                    departureCode : 24568,
                    timestamp : "2013-08-14T16:34:00+0200",
                    arrivalTime : 284,
                    reliability : "T",
                    deviation : false,
                    visible : true
              },
              ...
      ]
        disruptions : { (same object as GetNextDepartures function)

            deviations : [
              {
                    deviationCode:150,
                    startStop:{(same object as GetStops function)                    },
                    endStop:{ (same object as GetStops function) }
                    }
```

```
        ],

    }
}
```

6. Getting line colors

6.1 Getting line colors : GetLinesColors

6.1.1 Description

This function returns tpg line colors. Information is valid all the current operating day long.

6.1.2 Input

No input parameter is expected.

6.1.3 Possible use

- no input parameter : the function returns a list of lines with their color information in hexadecimal format.
  e.g.: GetLinesColors

6.1.4 Output

XML :

```
<colors>
  <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>

  <colors>
        <color>
              <lineCode>12</lineCode>
              <hexa>56FF42</hexa>
        </color>
        <color>
              <lineCode>13</lineCode>
              <hexa>46EF41</hexa>
        </color>
        …
  </colors>
</colors>
```

JSON :

```
{ colors :
  {
```

```
        timestamp:”YYYY-MM-DDThh:mm:ssTZD”,
        colors : [
                {
                        lineCode : "12",
                        hexa :"56FF42"
                },
                {
                        lineCode : "13",
                        hexa :"46EF41"
                },
                ...
        ]
        }
}
```

7. Getting traffic information

7.1 Getting disruptions : GetDisruptions

7.1.1 Description

This function returns real time disruptions currently impacting tpg bus network.
Please note that planned disruptions (public works, demonstrations) are not provided.

7.1.2 Input

No input parameter is expected.

7.1.3 Possible uses

- no parameter : the function returns the list of current disruptions.

e.g.: GetDisruptions

7.1.4 Output

XML :

```
<disruptions>
  <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>

  <disruptions>
        <disruption>
              <disruptionCode>150</disruptionCode>
              <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
              <place>77 Route de Chêne</lieu>
              <nature>Suite accident</nature>
              <stopName>Grange Canal</stopName>
              <lineCode>12</lineCode>
              <consequence>Retard de 15 minutes entre Grange Canal et
Carouge</consequence>
        </disruption>
        …
  </disruptions>

</disruptions>
```

JSON :

```
{ disruptions :
```

```
 {
        timestamp:”YYYY-MM-DDThh:mm:ssTZD”,
        disruptions : [
                        {
                                timestamp:”YYYY-MM-DDThh:mm:ssTZD”,
                                place:”77 Route de Chêne”,
                                nature:”Suite accident”,
                                stopName:”Grange Canal”,
                                lineCode :"12",
                                consequence:”Retard de 15 minutes entre Grange Canal et
Carouge”
                        },
                ...
        ]
        }
}
```

Please note disruption content is not localized, it is in French.

8. Error codes

Error messages you might get are listed below:

8.1 Error #400 : Bad request

This error message is sent when the request built by the client is syntactically incorrect. That might happen is the following cases:

8.1.1 A mandatory parameter is missing

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>10</errorCode>
        <errorMessage>Parameter [param] is missing</errorMessage>
</error>
```

where [param] corresponds to the first missing mandatory parameter of the list of expected parameters.

8.1.2 Bad parameter format

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>11</errorCode>
        <errorMessage>Parameter [param] format is incorrect </errorMessage>
</Error>
```

where [param] corresponds to the first incorrectly formatted parameter of the list of expected parameters.

8.1.3 Too many parameters

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>12</errorCode>
        <errorMessage>Too many parameters </errorMessage>
</error>
```

8.2 Error #403 : Forbidden

This error message is sent when API key used in the request is not valid :

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>20</errorCode>
```

```
        <errorMessage>invalid API key </errorMessage>
</error>
```

8.3 Error #404 : Not found

This error message is sent when the URL asked for is unknown, for example of you ask for a web service that does not exist :

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>30</errorCode>
        <errorMessage>Page not found</errorMessage>
</error>
```

8.4 Error #410 : Gone

This error message is sent when API version provided in input is not available on the server. It can happen in two cases: either the API version does not exist and never existed, or the API version is obsolete.

8.4.1 Obsolete API version

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>40</errorCode>
        <errorMessage>The requested API version is no more available. Please
upgrade.</errorMessage>
</error>
```

8.4.2 Unknown API version

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>41</errorCode>
        <errorMessage>The requested API version is incorrect<\errorMessage>
</error>
```

8.5 Error #503 : Service unavailable

This error message is sent when Real time information API server is not available. Technically, it might correspond to two different cases:

- it is the beginning of the operating day and data is being initialized (stop and line graph, stop sequences, line colors)
- data obtained from real time system looks obsolete (real time information “chain” is interrupted)

In both cases, error message is the following:

```
<error>
        <timestamp>YYYY-MM-DDThh:mm:ssTZD</timestamp>
        <errorCode>50</errorCode>
        <errorMessage>Service unavailable</errorMessage>
</error>
```