

The Taxi Exchange Point

Operator's Guide

Version	Description	Author	Date
1.0	Initial	David Beaudoin	16/08/2017
1.1	Révision	Stéphane Leblanc	07/09/2017
1.2	Révision	David Beaudoin	25/01/2018
1.3	Révision	Sébastien Blais	22/11/2018
1.4	Taxi position and status Mandatory column	Gaston-Andres Bellei	06/12/2019
1.5	Sections 2.4 and 3.3 updated	Gaston-Andres Bellei	15/06/2020
1.6	Sections 2.4 and 4.1 updated	Gaston-Andres Bellei	21/07/2020
1.7	Sections 2,1, 2.2, 2.3, 2.4 and 4.1 updated	Gaston-Andres Bellei	29/07/2020
1.8	Updated example column in section 4.1	Brian Di Croce	19/08/2020
1.9	Various text corrections	Gaston-Andres Bellei	10/11/2020
2.0	Bill 17 compliance added and general formatting	Gaston-Andres Bellei	20/11/2020

Table of contents

1. Introduction	1
1.1 Overview	1
1.2 Process of integration	1
1.3 Mandatory HTTP Headers	2
1.4 Authentication	2
1.5 APIs implementation	2
2. Contextual Data	2
2.1 Registering a Driver	2
2.2 Registering a Vehicle	5
2.3 Registering a Owner/license (ADS)	10
2.4 Declaring a Taxi	14
3. Taxi Positions and Status	18
3.1 Updating the location and status of a taxi	18
3.2 Taxis Status	20
3.3 Updating the status of a taxi	20
3.4 Querying a taxi	21
4. Hails	23
4.1 Hail Status	23
4.2 Receiving a Hail	26
4.3 Querying the Status of a Hail	27
4.4 Updating a Hail	27
5. Tests	28
5.1 Contextual Data Tests	29
5.1.1 The license plate of a vehicle changes	29
5.2 Hail Tests	32
5.2.1 Overview	32
5.2.2 Faking a Search Engine	32
5.2.3 Happy Path	39
5.2.4 Declined by taxi	41
5.2.5 Accepted by taxi after timeout	42
5.2.6 Canceled by taxi	44
5.2.7 Canceled by client	46
5.2.8 Failure example	48
5.3 Bill 17 Tests	49
5.3.1 Migrate a driver when the vehicles he drives have not been migrated yet	51
5.3.2 Migrate a vehicle when the drivers who drive it have been migrated	55
5.3.3 Migrate a vehicle and the drivers who drive it at the same time	59
5.3.4 Unallowed migration paths	63
5.3.5 Many vehicles can have the same owner	64

1. Introduction

1.1 Overview

The Taxi Exchange Point (TXP) aim is to connect taxis and their clients. Clients can use taxis search engines to hail taxis geolocated by taxis operators. The TXP mediates between search engines and operators.

All interactions with the TXP can be made from the central infrastructure of taxi operators, this also includes communication of the location and availability of taxis (using on-board equipment in the taxi).

1.2 Process of integration

To integrate with the TXP, the operator must send its contextual data (section 2) and the position and status of its taxis (section 3). Once the development is done in the acceptance environment, the operator must contact the TXP administrator. When the TXP administrator has verified that the operator is properly integrated, an API key will be sent to the operator for the production environment.

Sending the positions and status of the taxis is the first milestone for the operator. The law will eventually state that the taxi owners must send the position and the status of their taxis to the TXP via an authorized taxi operator. Operators will be given an API key for the production environment even if they cannot receive hails from the TXP (section 4).

Once the operator has shown that its system can receive hails from the TXP in the acceptance environment, the TXP administrator must be contacted. The TXP administrator will verify that the operator can receive hails properly and will configure the TXP in order to send hails to the operator in the production environment.

The TXP administrator can be contacted at: support.taxi.exchange.point@montreal.ca

Here are the links to communicate with the TXP services:

Acceptance : <https://taximtl.accept.ville.montreal.qc.ca>

Production : <https://taximtl.ville.montreal.qc.ca>

1.3 Mandatory HTTP Headers

The following HTTP Headers are mandatory for all requests to the TXP REST APIs:

Name	Value	Description
Accept	Application/json	Media types which are acceptable for the response
X-VERSION	2	Version of the API
X-API-KEY	token	API Key

1.4 Authentication

Authentication of your application is done for each query to the TXP by including a HTTP header X-API-KEY.

API keys are available for accredited developers and will be distributed by the BTM (Bureau Taxi Montréal) upon demand and validation.

1.5 APIs implementation

This documentation provides an overview of the TXP REST APIs. REST APIs provide access to resources (data entities) via URL paths. To use a REST API, your application will make an HTTPS request and parse the response. Your methods will be the standard HTTP methods like GET, PUT and POST. REST APIs operate over HTTPS making it easy to use with any programming language or framework. The input and output formats for the TXP REST APIs are JSON.

Note that data is isolated for each operator. No operator can see other operator's data.

2. Contextual Data

2.1 Registering a Driver

The structure of the required driver object is described below. You should push this information on a daily basis to keep to the data up to date.

Calls to this API are idempotent: you can update a driver simply by submitting the updated driver object with the same post method. If the department or professional licence is different, a new driver will be created; if the department and professional licence are unchanged, the driver will be updated.

Status on create	Status on update	Unique identifier(s)
201	200	departement and professional_licence

POST /api/drivers

Parameters

Body (JSON) **** Send only one item at a time**

```
{
  "data": [
    {
      "birth_date": "1950-12-22",
      "departement": {
        "nom": "Québec",
        "numero": "1000"
      },
      "first_name": "Jon",
      "last_name": "Doe",
      "professional_licence": "L1531-171274-08"
    }
  ]
}
```

Response (JSON) status 200 / 201

```
{
  "data": [
    {
      "birth_date": "1950-12-22",
      "departement": {
        "nom": "Québec",
        "numero": "1000"
      },
      "first_name": "Jon",
      "last_name": "Doe",
      "professional_licence": "L1531-171274-08"
    }
  ]
}
```

Key	Value Type	Description
departement	department object	<p>The departement object is constituted of the identifier numero and the name (nom) of the local authority.</p> <p>When a new driver is created by an Operator, an empty string or null can be passed instead of the name nom: only the identifier numero is used by the TXP.</p> <p>For Quebec, Since the adoption of Bill 17, the department should always be: departement.nom: "Québec" and departement.numero: 1000. When departement.numero is 1000 (Québec), the driver is identified by it's SAAQ driver's license number.</p> <p>Before Bill 17, drivers were part of the departement 660(Montreal) and were identified by their 'pocket number'.</p>
professional_licence	string	<p>Professional license number of the driver. It is often a string of digits but it might for some departments contain letters or other characters like dash or slashes.</p> <p>Warning: this identifier is not unique at the national level: two local authorities can each assign the same number to different drivers. Warning: the typo "licence" (French writing) instead of "license" (English writing) is still in the API (as of version 2).</p> <p>The couple of this professional license number (professional_licence) and the licensing local authority (departement) is used as the driver identifier when declaring a taxi as a vehicle/driver/license triplet.</p> <p>For Quebec, Since the adoption of Bill 17, the SAAQ driver's license number is used as the professional_licence.</p> <p>Before Bill 17, drivers were part of the</p>

		departement 660(Montreal) and were identified by their 'pocket number'.
last_name	string	Last name of the driver.
first_name	string	First name of the driver.
birth_date	string, RFC3339	Birth date of the driver in "YYYY-MM-DD" format. For Quebec, the birth date is ignored for privacy reasons.

2.2 Registering a Vehicle

The structure of the required vehicle object is described below. You should push this information on a daily basis to keep to the data up to date.

Calls to this API are idempotent: you can update a vehicle simply by submitting the updated vehicle object with the same post method. If the licence plate is different, a new vehicle will be created; if the licence plate is unchanged, the vehicle will be updated.

Response on create	Response on update	Unique identifier(s)
201	200	licence_plate

POST /api/vehicles

Parameters

Body (JSON) **** Send only one item at a time**

```
{
  "data": [
    {
      "licence_plate": "FAB1234",
      "vehicle_identification_number": "1FTFW1R6XBFD08251",
      "air_con": true,
      "horodateur": "aa",
      "color": "gris",
      "date_dernier_ct": "2016-12-22",
      "date_validite_ct": "2016-12-22",
      "credit_card_accepted": true,
      "electronic_toll": true,
      "fresh_drink": true,
      "pet_accepted": true,
      "tablet": true,
      "dvd_player": true,
      "taximetre": "aa",
      "every_destination": true,
      "nfc_cc_accepted": true,
      "baby_seat": true,
      "special_need_vehicle": true,
      "amex_accepted": true,
      "gps": true,
      "engine": "GO",
      "cpam_conventionne": true,
      "relais": true,
      "bank_check_accepted": true,
      "luxury": true,
      "horse_power": 2.0,
      "model_year": 1995,
      "wifi": true,
      "type_": "sedan",
      "nb_seats": 0,
      "constructor": "audi",
      "bike_accepted": true,
      "model": "a4"
    }
  ]
}
```


Response (JSON) status 200 / 201

```
{
  "data": [
    {
      "licence_plate": "FAB1234",
      "vehicle_identification_number": "1FTFW1R6XBFD08251",
      "air_con": true,
      "amex_accepted": true,
      "baby_seat": true,
      "bank_check_accepted": true,
      "bike_accepted": true,
      "color": "gris",
      "constructor": "audi",
      "cpam_conventionne": true,
      "credit_card_accepted": true,
      "date_dernier_ct": "2016-12-22",
      "date_validite_ct": "2016-12-22",
      "dvd_player": true,
      "electronic_toll": true,
      "engine": "GO",
      "every_destination": true,
      "fresh_drink": true,
      "gps": true,
      "horodateur": "aa",
      "horse_power": 2,
      "id": 36,
      "luxury": true,
      "model": "a4",
      "model_year": 1995,
      "nb_seats": 0,
      "nfc_cc_accepted": true,
      "pet_accepted": true,
      "private": false,
      "(obsolete)"relais": true,
      "special_need_vehicle": true,
      "tablet": true,
      "taximetre": "aa",
      "type_": "sedan",
      "wifi": true
    }
  ]
}
```

Key	Value Type	Description
licence_plate	String	Mandatory - License plate of the vehicle. Warning: the typo "licence" (French writing) instead of "license" (English writing) is still in the API (as of version 2). The licence_plate is used as the vehicle identifier to declare a taxi as a vehicle/driver/license triplet. For historical reasons, values of licence_plate are case sensitive . Even though, in reality, license plates should not have lower case letters.
vehicle_identification_number	String	Optional - The licence_plate is the only mandatory identifier for vehicles in the taxi registry. Even though it is an optional attribute, the vehicle identification number must be transmitted when available.
constructor	String	Constructor of the vehicle.
model	String	Model of the vehicle.
color	String	Color of the vehicle.
type_	String	Type of the vehicle. The possible values are sedan, station_wagon, normal or mpv. Warning: the name of this key is type_ with the final underscore. If your type is not listed use "type_": null.
nb_seats	Integer	Number of seating positions available for passengers in the vehicle (not counting the seat of the driver). As per European Regulation EU/678/2011 the following requirements apply for the counting of the seating positions: (a) each individual seat shall be counted as one seating position; (b) in the case of a bench seat, any space having a width of at least 400 mm measured at the seat cushion level shall be counted as one seating position. (c) however, a space as referred to in point (b) shall not be counted as one seating position where: (i) the bench seat includes features that prevent the bottom of the manikin from sitting in a natural way - for example: the presence of a fixed console box, an unpadded area or an interior trim interrupting the nominal seating surface; (ii) the design of the floor pan located immediately in front of a presumed seating position (for example the presence of a tunnel) prevents the feet of the manikin from being positioned in a natural way. When available, the area intended for an occupied wheelchair shall be regarded as one seating position.
air_con	Boolean	This vehicle is equipped with air conditioning.
amex_accepted	Boolean	This vehicle accepts American Express card for any amount (no minimum).
baby_seat	Boolean	This vehicle is equipped with a baby seat.

bank_check_accepted	Boolean	This vehicle accepts national bank checks (foreign bank checks might still be refused).
bike_accepted	Boolean	This vehicle can transport a bicycle.
credit_card_accepted	Boolean	This vehicle accepts credit card payments for any amount (no minimum). This should be true for vehicles accepting at least Visa and MasterCard. There is a different Boolean amex_accepted for American Express.
dvd_player	Boolean	This vehicle has a DVD player at the disposal of clients during the ride.
electronic_toll	Boolean	This vehicle is equipped with an electronic device letting them use express toll booths on toll roads.
every_destination	Boolean	As per the French regulation, taxis can refuse service to clients whose destination is not within their zone. Some taxis do accept any destination outside of their zone. The every_destination boolean should be false by default, and true for taxis who renounce their right to refuse service to clients depending on their destination.
fresh_drink	Boolean	This taxi offers refreshments.
gps	Boolean	This vehicle is equipped with GPS navigation.
luxury	Boolean	This is a luxury vehicle.
nfc_cc_accepted	Boolean	This vehicle accepts NFC credit card payments.
pet_accepted	Boolean	This vehicle can accommodate pets (understood as cats or small dogs; other large or unusual pets might still be refused).
special_need_vehicle	Boolean	Wheelchair accessible vehicle as defined in “ EU/678/2011 ” (which amends 2007/46/EC). Vehicles constructed or converted specifically so that they accommodate one or more persons seated in their wheelchairs when travelling on the road.
tablet	Boolean	This vehicle has a digital tablet at the disposal of the clients during the ride.
wifi	Boolean	This vehicle has complimentary Wi-Fi aboard.
cpam_conventionne	Boolean	This vehicle has a convention with social security to transport patients. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
date_dernier_ct	string, RFC3339	Date of the latest compulsory roadworthiness tests in "YYYY-MM-DD" format. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.

date_validite_ct	String, RFC3339	Expiration date of the latest compulsory roadworthiness tests in "YYYY-MM-DD" format. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
engine	String	Engine type of the vehicle. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
horse_power	Integer	Fiscal power of the vehicle. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
model_year	Integer	Model year of the vehicle. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
relais	Boolean	True if this vehicle is a temporary replacement vehicle for a fully licensed one. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
taximetre	String	Brand and model of the taximeter. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
horodateur	String	Brand and model of the time clock. This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value.
id	Integer	This field is used for administrative purposes only. When a new vehicle is created by an Operator, this field can be omitted or passed with a null value. There is no need for Operators or Search Engines to store the value returned by the TXP: the field used to uniquely identify vehicles in all transactions with the TXP is the licence_plate.

2.3 Registering a Owner/license (ADS)

The structure of the required ads object is described below. You should push this information on a daily basis to keep to the data up to date.

Calls to this API are idempotent: you can update a owner (ADS) simply by submitting the updated ads object with the post method. If the insee or numero is different, a new owner (ADS) will be created; if the insee and numero are unchanged, the owner (ADS) will be updated.

Owner (ADS) vs license (ADS)

Following adoption of Bill 17, the meaning of ADS has changed from license to owner.

The owner of a vehicle used as a taxi requires a license for each vehicle he owns. Before Bill 17, the taxi registry was keeping track of each individual license. Since the adoption of Bill 17, the taxi registry does not keep track of each individual license anymore. The taxi registration does now only keep track of the owner and the vehicles he owns. An owner may own many vehicles.

The ADS with the meaning of owner can be distinguished from the ADS with the meaning of license by the value of the taxi zone (insee). If the taxi zone is 1000 (Québec), it means owner, otherwise it means license.

Zone	ADS.insee	ADS.numero	Name	ADS.vdm_vignette
Before Bill 17				
Montréal Est	102005	4M000000011A	John Doe	8811
Montréal Est	102005	4M000000022B	John Doe	8822
After Bill 17				
Québec	1000	161555777	John Doe	Unused

Response on create	Response on update	Unique identifier(s)
201	200	insee and numero

POST /api/ads

Parameters ** Send only one item at a time

Body (JSON)

```
{
  "data": [
    {
      "category": "",
      "vehicle_id": 36,
      "insee": "1000",
      "numero": "161555777",
      "owner_name": "Co-op",
      "owner_type": "company",
      "doublage": false,
      "vdm_vignette": "string"
    }
  ]
}
```

Response (JSON) status 200 / 201

```
{
  "data": [
    {
      "category": "",
      "doublage": false,
      "insee": "1000",
      "numero": "161555777",
      "owner_name": "Co-op",
      "owner_type": "company",
      "vehicle_id": 36,
      "vdm_vignette": "string"
    }
  ]
}
```

Key	Value Type	Description
insee	string	<p>Bill 17 abolishes the existing taxi zones, Montreal West (A12), Montreal downtown (A11) and Montreal East (A5), with the exception of the YUL airport zone which is under federal jurisdiction.</p> <p>Since the adoption of Bill 17, all taxis, in Quebec province, belong to the 1000(Québec) taxi zone.</p> <p>Before the adoption of Bill 17, ADS were identified by their CTQ license number. Three agglomerations exist for Montreal as follow:</p> <p>402005 : A5 — Eastern part of the island of Montreal</p> <p>402011 : A11 — Downtown/center Montreal</p> <p>402012 : A12— West part of the island of Montreal</p>

numero	string	<p>After Bill 17, numero represents: The SAAQ file number identifying a company or an individual that owns a vehicle. The SAAQ file number format varies depending on the owner being a company or an individual. Ex: company = 161902393 Ex: individual = L1531-171274-08 For individuals, the SAAQ file number is the same as the driver license number.</p> <p>See section <i>Owner (ADS) vs license (ADS)</i> above for more details.</p> <p>The couple ADS.insee and ADS.numero is used when declaring a taxi as a driver/vehicle/license triplet.</p> <p>Before the adoption of Bill 17, ADS were identified by their CTQ license number (12 alphanumeric characters).</p>
owner_name	string	<p>Name of the holder of the license. Warning: It might be either an individual or a company.</p>
owner_type	string	<p>The two possible values are company or individual.</p>
category	string	<p>This field is used for administrative purposes. When a new license (aka ADS) is created by an Operator, an empty string has to be passed (not a null value).</p>
doublage	boolean	<p>Some regulation specific to the Paris area limits the working hours of the driver to 10 hours a day. Some licenses (ADS) can be used for 2 shifts a day (by two different drivers) and this field should then be set to true. Others can only be operated 10 hours a day and this field should be set to false. When a new license (aka ADS) is created by an Operator, this field should always be set to false if the local authority in the insee field is not 75056 (i.e. Paris).</p>
vehicle_id	integer	obsolete

vdm_vignette	string	This field represents the "Vignette" number given by the BTM (Bureau Taxi Montreal). Mandatory. This is ignored when <i>insee</i> (see above) is Québec-1000 zone.
--------------	--------	---

2.4 Declaring a Taxi

Status attribute is OBSOLETE and will be ignored.

The structure of the required taxi object is a minimalist version containing only the identifiers of the vehicle, driver and ads and the initial status of the taxi. The vehicle, driver and ads used to compose a taxi need to have been registered first through their respective API.

As per drivers, ads, vehicles etc, information can be updated daily, but this request should be used on new taxi creation or when private attribute changes.

If successful, the API returns the complete taxi object as described including the characteristics of the vehicle and most importantly the unique identifier id of the taxi that will be used for subsequent communications.

Calls to this API are idempotent: if you resubmit the same triplet of vehicle, driver and ads, the taxi returned will have the same id.

Private parameters can be updated via this POST request.

Warning: Please make sure to save the returned Id, it will be required to update the taxi later on.

Response on create	Response on update	Unique identifier(s)
201	200	licence_plate (vehicle) and departement and professional_licence (driver) and insee and numero (ads)

POST /api/taxis

Parameters

*Body (JSON) ** Send only one item at a time*

```
{
  "data": [
    {
      "private": true,
      "vehicle": {
        "licence_plate": "FAB1234"
      },
      "driver": {
        "departement": "1000",
        "professional_licence": "L1531-171274-08"
      },
      "ads": {
        "insee": "1000",
        "numero": "161555777"
      }
    }
  ]
}
```

Response (JSON) status 200 / 201

```
{
  "data": [
    {
      "ads": {
        "insee": "1000",
        "numero": "161555777"
      },
      "driver": {
        "departement": "1000",
        "professional_licence": "L1531-171274-08"
      },
      "id": "ueXs7TR",
      "last_update": null,
      "operator": null,
      "position": {
        "lat": null,
        "lon": null
      },
      "private": true,
      "rating": 4.5,
      "vehicle": {
        "licence_plate": "FAB1234",
        "characteristics": null,
        "color": null,
        "constructor": null,
        "model": "a4",
        "nb_seats": null
      }
    }
  ]
}
```

Key	Value Type	Description
vehicle	vehicle	A partial vehicle object with only the fields: characteristics, color, constructor, licence_plate, model, nb_seats. Warning: some of those fields might not be returned (or be returned with a null value) if they were not provided by the taxi operator.
ads	ADS	A partial ADS object with only the fields: insee, numero. When <i>ADS.insee</i> is Québec-1000 then: - <i>driver.departement</i> must be 1000 (Québec). - <i>vehicle.licence_plate</i> cannot be a 'T' license plate. For more detail, see section 5.3.4.
driver	driver	A partial driver object with only the fields: departement, professionnel_licence.
id	string	A long-lived 7 characters long identifier generated for this vehicle/ads/driver triplet by the TXP. This field should be omitted by operators when declaring a new taxi through a POST request; the newly generated id will be returned in the taxi object sent back as the response.
operator	string	Login of the certified operator.
private	boolean	As per VDM and BTM's requirements, as an option, you can set the taxi's private field to true or false. By default, the taxi's private field is set to false. A private taxi will never receive hails from the TXP.
rating	float	The mean of the ratings of last rides of the taxi. It is calculated by the TXP and falls between 0 and 5.
status	status	Status of the taxi. The possible values are described in the section 4.1 Hail Status.

position	{lat, lon}	The latitude and longitude of the taxi. Warning: those values are only returned by the TXP in the response to a GET request on the /taxis/ API looking for taxis around a client. They will be nulled when returned in the response to a GET request on the /taxis/{taxi_id}/ API looking for information on a specific taxi.
last_update	integer	Timestamp of the last geolocation update of the taxi. The format is the usual Unix time (IEEE P1003.1 POSIX) and as such is UTC (no timezone).

3. Taxi Positions and Status

3.1 Updating the location and status of a taxi

You should push this information in batches every 5 seconds to keep the data up to date.

The JSON payload should be as follows.

POST /api/taxi-position-snapshots

Parameters

Body (JSON) **items should contain all your taxis

```
{
  "items": [
    {
      "timestamp": "1430076493",
      "operator": "coop",
      "taxi": "tPc79rW",
      "lat": "45.38852053",
      "lon": "-73.84394873",
      "device": "phone",
      "status": "free",
      "version": "2",
      "speed": "50",
      "azimuth": "180"
    }
  ]
}
```

Key	Value Type	Description	Mandatory
timestamp	string	<p>Exact time at which the location was determined by the taxi, formatted as a Unix time (IEEE 1003.1-2008 POSIX).</p> <p>Warning: as per the POSIX specification, this should be UTC time without any timezone information.</p> <p>Warning: Do not send locations in the future (or older than 1 minute) as they will return a http 400 error. Timestamp must be in second.</p>	Yes
operator	string	Login of the certified operator.	Yes
taxi	string	The id of the taxi is the id that was sent back when the taxi was declared (see <i>Declaring a taxi</i>).	Yes
lat	string	<p>Latitude of the taxi.</p> <p>This should be in JavaScript double precision floating-point format, with decimal separator ".".</p> <p>You can truncate the values to 6 decimal places if you want to keep the payload as short as possible (6 decimal places is worth up to 10 cm).</p>	Yes
lon	string	<p>Longitude of the taxi.</p> <p>This should be in JavaScript double precision floating-point format, with decimal separator ".".</p> <p>You can truncate the values to 6 decimal places if you want to keep the payload as short as possible (6 decimal places is worth up to 10 cm).</p>	Yes
device	string	phone, tablet, taximeter or otherdevice .	Yes

status	string	<i>Possible values: answering, free, occupied, off, oncoming or unavailable.</i> <i>Mandatory.</i> <i>For more details, see the table below.</i>	Yes
version	string	<i>"2" for now (geolocation version 2 of the API).</i>	Yes
speed	string	<i>The actual speed of the taxi (in km/h).</i>	Yes
azimuth	string	<i>The current orientation of the taxi (360°).</i>	Yes

3.2 Taxis Status

Value	Description	Mandatory
answering	<i>The taxi is currently answering to a hail.</i>	<i>No, use unavailable if the information is not available.</i>
free	<i>The taxi can be hailed.</i>	yes
occupied	<i>The taxi has a customer on board.</i>	yes
off	<i>The taxi is not logged in or did not update its location recently enough.</i>	yes
oncoming	<i>The taxi is on its way to meet a customer.</i>	<i>No, use unavailable if the information is not available.</i>
unavailable	<i>The taxi is logged in, but cannot be hailed.</i>	yes

3.3 Updating the status of a taxi

This section is OBSOLETE. Please notice the following:

1. It is now recommended to use POST /api/taxis (section 2.4) to modify private attribute.
2. Even if it is discouraged to use PUT /api/taxis/{taxi_id}, it is still a valid request.

3. Only status change submitted thru POST /api/taxi-position-snapshots (section 3.1) will be considered. Sending status value thru PUT /api/taxis/{taxi_id}, is still possible but status value it will be ignored.

The status of the taxi should be sent to the TXP whenever there is a change of status from the operator. The possible status is free or occupied or off or answering or oncoming. This is done through a “HTTPS PUT request to the /taxis/{taxi_id}/ API”.

You can only update the following attributes: status and private. For more details, see the attribute description in section 2.4 “Declaring a taxi”.

```
PUT /api/taxis/{taxi_id}
```

Parameters

Taxi_id (string)

*Body (JSON) ** Send only one item at a time*

```
{
  "data": [
    {
      "status": "free",
      (mandatrory)"private": "false" (string or boolean)
    }
  ]
}
```

Response

Return the taxi's details in JSON (see below 3.4 Querying a taxi)

3.4 Querying a taxi

In order to check that the updating of the status or location of the taxi worked properly, you can use a “HTTPS GET request to the /taxis/{taxi_id}/ API”.

Warning: the GET /taxis/{taxi_id}/ API will return the status and the last_update (in UNIX TIME STAMP) but the “lat” and “lon” will be nulled (for privacy reasons).

Warning: in production, you should almost never need the GET /taxis/{taxi_id}/ API. The endpoint is provided only to improve the developer experience by allowing them to know the status, ads, driver and vehicle of a taxi.

GET /api/taxis/{taxi_id}

Parameters

Path

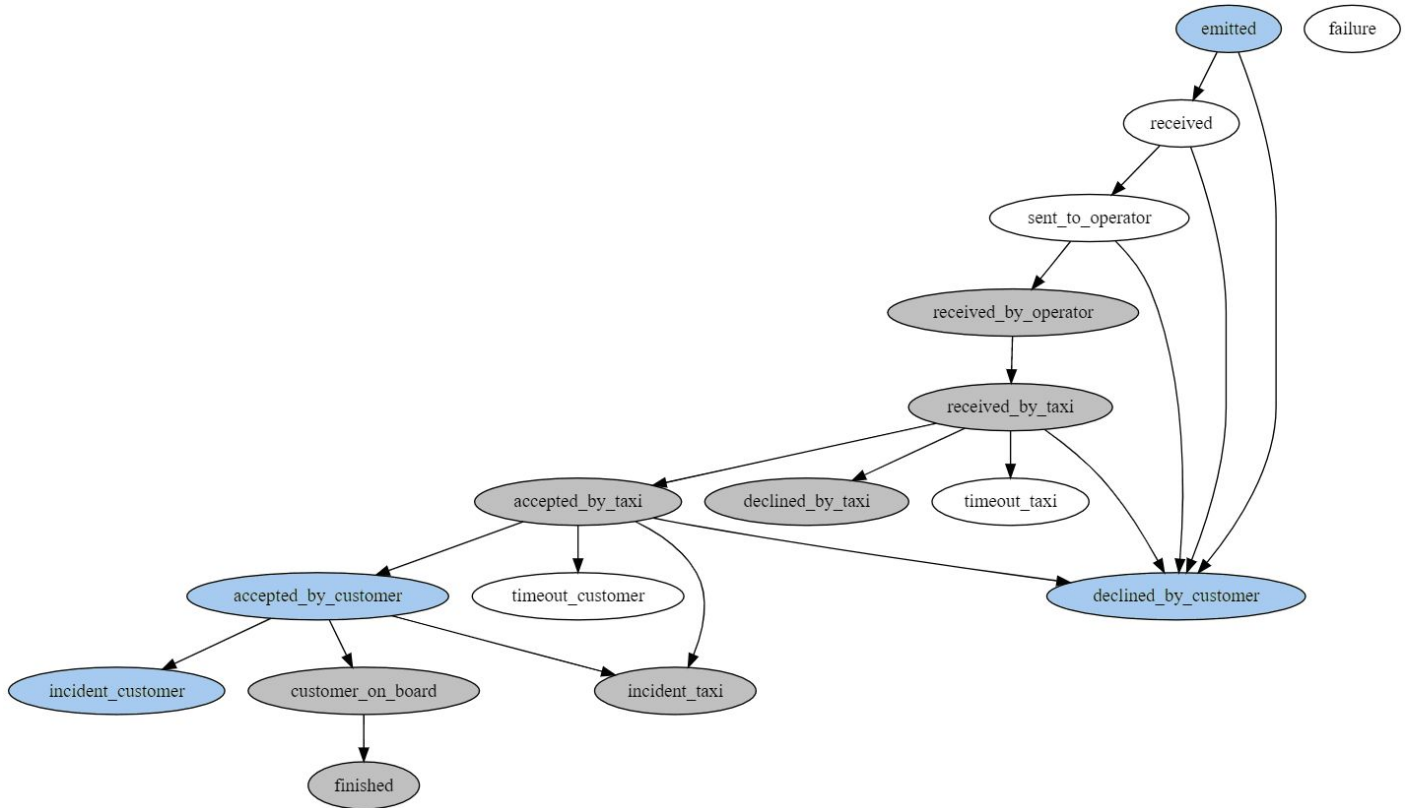
taxi_id (string)(required)

Response (JSON) status 200

```
{
  "data": [
    {
      "ads": {
        "insee": "1000",
        "numero": "161555777"
      },
      "crowfly_distance": null,
      "driver": {
        "departement": "1000",
        "professional_licence": "L1531-171274-08"
      },
      "id": "VsLwptA",
      "last_update": 1502819736,
      "operator": "coop",
      "position": {
        "lat": null,
        "lon": null
      },
      "private": false,
      "rating": 4.42332039594968,
      "status": "answering",
      "vehicle": {
        "licence_plate": "FAB1234",
        "characteristics": [
          "every_destination",
          "gps",
          "pet_accepted",
          "bike_accepted",
          "credit_card_accepted",
          "luxury"
        ],
        "color": "GRISE",
        "constructor": "TOYOTA",
        "model": "SIENNA",
        "nb_seats": 6
      }
    }
  ]
}
```


4. Hails

4.1 Hail Status



The states in gray can be reached after an interaction with the operator and are described in section 4.3 “Operator”. The ones in blue can be reached after an interaction with the search engine and are described in the section 4.2 “Faking a Search Engine”. The states in white are under the control of the TXP.

The section [5.2 Hail Tests](#) describes the scenario that the operator must support in order to receive hail from the TXP.

Status description

Value	Description	Interaction
emitted	The initial status of hail when created by a search engine	This is the status that should be used in the payload of the POST request on the <code>/hails/</code> API when a search engine creates a new hail.
received	The hail is received from the search engine by the TXP	Before forwarding the hail to the operator, the TXP changes the status to <code>received</code> and sends back the complete hail (with the newly generated id) to the search engine. Time before failure 15 seconds.
sent_to_operator	The hail has been sent from the TXP to the operator	The TXP changes the status to <code>sent_to_operator</code> after forwarding the hail to the operator endpoint. Time before failure 10 seconds.
received_by_operator	The operator has acknowledged receiving the hail from the TXP	The TXP changes the status to <code>received_by_operator</code> after receiving a HTTP 200 response from the “operator” endpoint. Time before failure 10 seconds.
received_by_taxi	The hail has been received by the taxi	The “operator” should set the status of the hail to <code>received_by_taxi</code> by doing a “PUT” request on the <code>/hails/{hail_id}</code> API when the hail has been presented to the taxi driver. Time before <code>timeout_taxi</code> 30 seconds.
accepted_by_taxi	The hail has been accepted by the taxi driver	The “operator” should set the status of the hail to <code>accepted_by_taxi</code> by doing a “PUT” request on the <code>/hails/{hail_id}</code> API when the hail has been accepted by the taxi driver. Time before <code>timeout_customer</code> 10 minutes.
declined_by_taxi	The hail has been declined by the taxi driver	The “operator” should set the status of the hail to <code>declined_by_taxi</code> by doing a “PUT” request on the <code>/hails/{hail_id}</code> API when the hail has been rejected by the taxi driver.
timeout_taxi	The taxi driver did not accept nor reject the hail after 30s	The TXP changes the status to <code>timeout_taxi</code> automatically after 30s have passed since the status was set to <code>received_by_taxi</code> .

accepted_by_customer	The hail has been confirmed by the client	The “ search engine ” should set the status of the hail to “ accepted_by_customer ” by doing a “ PUT ” request on the /hails/{hail_id} API when the hail has been confirmed by the search engine. Time before failure 1 hour. Warning: this confirmation can only happen “after” the status has been set to “ accepted_by_taxi ” by the “ operator ”.
declined_by_customer	The hail has been canceled by the client	The “ search engine ” should set the status of the hail to “ declined_by_customer ” by doing a “ PUT ” request on the /hails/{hail_id} API when the hail has been canceled by the client. Warning: this cancellation can happen at any moment (including before the taxi driver accepts the hail).
timeout_customer	The client did not confirm nor cancel the hail after 10 minutes	The “ TXP ” changes the status to “ timeout_customer ” automatically after 10 minutes have passed since the status was set to accepted_by_taxi .
incident_customer	An event of force majeure prevents the client to wait for the taxi	The “ search engine ” should set the status of the hail to “ incident_customer ” by doing a “ PUT ” request on the /hails/{hail_id} API when the client cancels the hail after having reconfirmed it.
incident_taxi	An event of force majeure prevents the taxi to serve the client	The operator should set the status of the hail to “ incident_taxi ” by doing a “ PUT ” request on the /hails/{hail_id} API when the taxi cancels the hail after having accepted it.
failure	A technical problem happened.	The “ TXP ” changes the status to “ failure ” when: <ul style="list-style-type: none"> • The operator endpoint is unreachable; • or when receiving a HTTP 4xx or 5xx response from the operator endpoint; • or if the operator endpoint does not return a hail JSON object containing a valid taxi_phone_number; • or if the operator does not set the status of the hail to received_by_taxi in the 10s after the status has been set to received_by_operator.
customer_on_board	The customer is on board.	The “ operator ” should set the status of the hail to customer_on_board by doing a “ PUT ” request on the /hails/{hail_id} API when the hail has been accepted by the taxi driver.

finished	The hail is finished	<i>The “operator” should set the status of the hail to finished by doing a “PUT” request on the /hails/{hail_id} API when the hail has been accepted by the taxi driver.</i>
-----------------	----------------------	---

Status changes based on timeout

Status before timeout	Status after timeout	Timeout delay (seconds)
accepted_by_customer	failure	3 600
accepted_by_taxi	timeout_customer	600
customer_on_board	failure	86 400
emitted	failure	10
received	failure	15
received_by_operator	failure	10
received_by_taxi	timeout_taxi	30
sent_to_operator	failure	10

4.2 Receiving a Hail

In order to receive hails, “operators” need to implement an endpoint on their servers. The endpoint has to be able to receive HTTPS “POST” requests from the TXP. This is the only endpoint that needs to be implemented on the “operator” side.

The certificate used by the operator for HTTPS requests must be recent and trusted by a well-known certification authority. Moreover, the endpoint must use an API key transmitted via an HTTP header in order to authenticate the client.

Accredited “operators” can configure the URL of the endpoint on the operator side, the API key HTTP header and the API key value on their profile page of the TXP website.

“hail” JSON object which will be transmitted as the payload of the HTTPS POST request has the same structure as the response described in section 4.2.1 Hailing a taxi_Hail_Structure.

The endpoint should return one of the valid HTTP status code in the “2xx Success” range in case of success, and in the “4xx Client Error or 5xx Server Error in case of error. If the status code is in the “2xx Success” range, the response payload can be a JSON hail object, in which case the hail will be

updated on the TXP. This response payload can for instance be used to transmit the “taxi_phone_number” to the TXP.

4.3 Querying the Status of a Hail

In order to keep track of the status of the hail, you can do a “HTTPS GET request to the/hails/{hail_id}/API0”. The JSON object has the same structure as the response described in section 4.2.1 Hailling a taxi. `_Hail_Status`.

The operator needs to fetch the status of the hail to check if the status changed. The polling delay for this check should be equal or superior to 120 seconds the moment the hail is received until it reaches the “customer_on_board” status.

```
GET /api/hails/{hail_id}
Parameters
Path
Hail_id (string)(required)
```

4.4 Updating a Hail

As the hail transaction progresses, the operator must update the hail status. This is done through a call of the “HTTPS PUT request to the hails API.

See section [5.2 Hail Tests](#), for more details on when and why the operator must update the hail status.

You do have to check the response of the PUT request to ensure that the update went as expected. For example, if the PUT request was not received in a timely fashion, the hail status may be “timeout_taxi” instead of the expected status. If the transaction reaches an unexpected end state (ex: timeout_taxi, failure, etc.), the whole hail transaction will be considered cancelled and a new hail transaction must be restarted.

```

PUT /api/hails/{hail_id}
Parameters
Path
Hail_id (string)(required)
Body (JSON) **all value inside the JSON are OPTIONAL, use as needed
** Send only one item at a time
{
  "data": [
    {
      "status": "emitted",
      "incident_taxi_reason": "no_show",
      "reporting_customer": true,
      "reporting_customer_reason": "ko"
    }
  ]
}

```

5. Tests

This section illustrates tests that operator's IT staff can perform to verify proper Taxi Exchange Point (TXP) integration and to ensure that changes, required by Bill 17, are properly implemented.

Steps grouping

Tests steps are grouped, when appropriate, in two sections:





- Initial State.
 - Indicates the situation before tests start. This situation may already exist or can be created in the acceptance environment by following proposed steps.
 - On test step's table this section starts with the title: **Initial State**
 - On pictograms this section is delimited by dashed lines.
- Test.
 - Indicates steps to follow in order to conform to Bill 17.
 - On test step's table this section starts with the title: Test

Pictograms

Pictograms are used to facilitate comprehension by clearly differentiating entities. (Different vehicles, drivers, taxis etc.) Also, at the end of test procedures, they help illustrate, not only the entities created by the test procedure, but also the link between different entities.

- A particular color or an uppercase letter/number combination identifies a single entity.
- Different entities of the same color are not necessarily linked, though they generally are.
- Arrows formally bind different entities.


Icons contain a textual reference, consisting of an uppercase letter: *D* for driver, *V* for vehicle, *P* for permit and *T* for taxi followed by and an index number: Uniquely identifying the entity.





Icon	Entity (generic)	Identification
	Driver	Drive number 1
	Vehicle	Vehicle number 1
	Permit (ADS)	Permit number 1
	Taxi	Taxi number 1

5.1 Contextual Data Tests

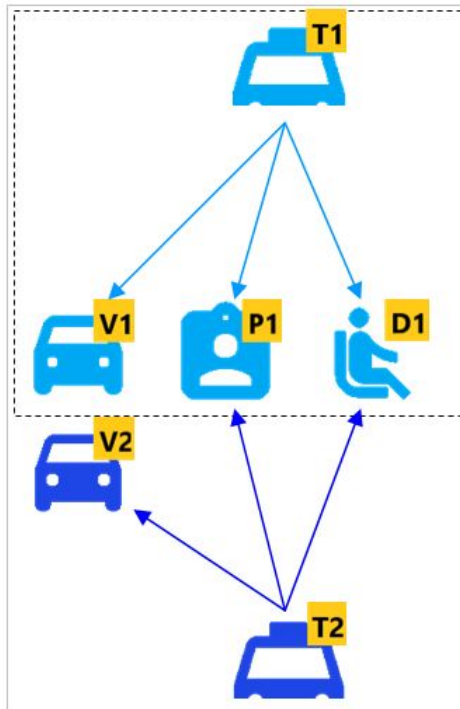
5.1.1 The license plate of a vehicle changes

Unlike the driver's license number and the SAAQ file number which are immutable, the license plate can change over time. An owner can change, at will, his vehicle's license plate.

Step	Action	Request	Response	Icon
Initial State				
1	Create a driver.	<pre>POST /api/drivers { "data": [{ "professional_licence": "L1006-221166-01", "departement": { "nom": "Québec", "numero": "1000" } }, ...] }</pre>	HTTP 201 Created	

2	Create a vehicle.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "FAA0011", "type_": "sedan", "constructor": "audi", "model": "a4" }] }</pre>	HTTP 201 Created	
3	Create a permit/ADS.	<pre>POST /api/ads { "data": [{ "insee": "1000", "numero": "161000011", }] }</pre>	HTTP 200 OK	
4	Create a taxi using the driver D1, vehicle V1 and permit/ADS P1.	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "1000", "numero": "161000011" }, "vehicle": { "licence_plate": "FAA0011" }, "driver": { "departement": "1000", "professional_licence": "L1006-221166-01" }, }] }</pre>	HTTP 201 Created <pre>{ "data": [{ "id": "\$T1", }] }</pre>	
Test				
5	Create a vehicle identical to V1 but with a different licence plate.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "FBB0022", "type_": "sedan", "constructor": "audi", "model": "a4" }] }</pre>	HTTP 201 Created	

6	Create a taxi for vehicle with the new license plate	<pre> POST /api/taxis { "data": [{ "ads": { "insee": "1000", "numero": "161000011" }, "vehicle": { "licence_plate": "FBB0022" }, "driver": { "departement": "1000", "professional_licence": "L1006-221166-01" }, ... }] } </pre>	<p>HTTP 201 Created</p> <pre> { "data": [{ "id": "\$T2", ... }] } </pre>	
7	Start sending taxi positions and status with the taxi created (T2).	<pre> POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T2", ... }] } </pre>	<p>HTTP 200 OK</p>	



Entities present in the system after the license plate renewal

5.2 Hail Tests

5.2.1 Overview

For testing purposes, the operators will be allowed to emulate a search engine in the accept environment. Your API key will allow you to create and update hails for your taxis only. This will not be allowed in the production environment.

5.2.1.1 Scenarios

All the scenarios that the operator must support in order to receive hails from the Taxi Exchange Point (TXP) are described in sections 5.2.3 to 5.2.8. Make sure you test them thoroughly.

5.2.1.2 Unexpected Exceptions

A state transition to failure state may occur from any state. Moreover, technical problems may cause the TXP to be unreachable or to respond with a HTTP 500 status code (server error). These are edge cases that should not occur frequently, but the operator must be ready to deal with these situations. If an unexpected error occurs, the operator should assume that the state of the hail is failure and should stop interacting with the TXP for this hail.

The operator must notify the driver that a technical problem occurred and that the hail is canceled if the technical problem occurred before reaching the following states: `received_by_taxi`, `accepted_by_taxi`, `declined_by_taxi`. However, there is no need to notify the driver if the technical problem occurs before reaching the following states: `incident_taxi`, `customer_on_board`, `finished`

5.2.2 Faking a Search Engine

5.2.2.1 Hailing a taxi

Hailing a taxi is done through an HTTPS POST request to the fake hail API. The structure of the required "hail" object is described below.

In order to receive a hail, the state of the taxi must be free. For more details, see section 3.3 Updating the status of a taxi.

In order to receive a hail, the location of the taxi must have been updated recently. For more details, see section 3.1 Updating the location and status of a taxi.

POST/api/motor/hails/

Parameters

Body (JSON) **** Send only one item at a time**

```
{
  "data": [
    {
      "customer_lat": 45.58017,
      "customer_lon": -73.61479,
      "customer_address": "801 rue Brennan, Montreal QC H3C 0G4",
      "taxi_id": "tPc79rW",
      "customer_phone_number": "514 999-9999",
      "opérateur": "operPlus",
      "customer_id": "anonymous"
    }
  ]
}
```

Response (JSON) Status 200 OK

```
{
  "data": [
    {
      "creation_datetime": "Thu, 22 Dec 2016 11:24:53 -0000",
      "customer_address": "801 rue Brennan, Montreal QC H3C 0G4",
      "customer_id": "anonymous",
      "customer_lat": 45.58017,
      "customer_lon": -73.61479,
      "customer_phone_number": "514 999-9999",
      "id": "hvuJ45S",
      "incident_customer_reason": null,
      "incident_taxi_reason": null,
      "last_status_change": "Thu, 22 Dec 2016 11:24:53 -0000",
      "operateur": "coop",
      "rating_ride": null,
      "rating_ride_reason": null,
      "reporting_customer": null,
      "reporting_customer_reason": null,
      "status": "received",
      "taxi": {
        "id": "tPc79rW",
        "last_update": 1482423893,
        "position": {
          "lat": 45.6164341134,
          "lon": -73.6138161294
        }
      },
      "taxi_phone_number": null
    }
  ]
}
```

Key	Value Type	Description
customer_lat	float	<p>Latitude of the position of the client.</p> <p>This should be in JavaScript double precision floating-point format, with decimal separator "."</p>
customer_lon	float	<p>Longitude of the position of the client.</p> <p>This should be in JavaScript double precision floating-point format, with decimal separator "."</p>
customer_address	string	<p>Address of the position of the client.</p> <p>This address will be used by the taxi driver to find the client.</p> <p>It should be displayed and validated by the client.</p> <p>Warning: In some cases, a POI might be more meaningful than a postal address.</p>
customer_id	string	<p>Identifier of the customer.</p> <p>The only acceptable value is "anonymous" (case sensitive)</p>
customer_phone_number	string	<p>Phone number of the client.</p> <p>This phone number might be used by the Operator of the taxi in case it proves difficult to find the client.</p>
taxi_id	string	<p>Identifier of the taxi the client is hailing.</p> <p>This identifier was returned by the TXP in the Taxi object.</p> <p>Warning: for historical reasons, when a "search engine" sends a new hail to the TXP, the taxi id should be passed as a "taxi_id" field directly in the "ail" object.</p> <p>In all subsequent exchanges, including when the "TXP" forwards the "hail" to the "operator", the taxi id appears instead as a "id" field in an embedded "taxi" JSON object inside the hail.</p>
opérateur	string	<p>Identifier of the Operator of the taxi the client is hailing.</p> <p>This identifier was returned by the "TXP" in the "Taxi" object.</p>

status	status	<p>Status of the hail.</p> <p>All possible values are described here</p>
id	string	<p>Identifier of the hail.</p> <p>This identifier should be <i>null</i> or omitted when a “search engine” sends a new hail to the TXP. The newly generated “id” will be in the “hail” object returned by the TXP as a response.</p>
taxi	taxi	<p>Details of the taxi selected for the hail.</p>
taxi_phone_number	string	<p>Phone number the client should call in case of problem.</p> <p>This phone number can be either the number of the call center of the operator or the mobile phone number of the taxi driver. It has to be reachable at the time of the ride: call center numbers should only be transmitted during opening times.</p>
incident_customer_reason	string	<p>Reason of the incident that prompted the client to cancel the ride.</p> <p>This is reserved for future use. The only accepted value as per version 2 of the API is an empty string. This field should be used by “search engines” when setting the status of the ride to <i>incident_customer</i>.</p>
incident_taxi_reason	string	<p>Reason of the incident that prompted the taxi to cancel the ride.</p> <p>This field should be used by “operators” when setting the status of the ride to <i>incident_taxi</i>.</p> <p>The possible reasons are: “<i>no_show</i>” (when the client cannot be found), “<i>address</i>” (when the address cannot be found), “<i>traffic</i>” (when a traffic jam prevents the taxi to arrive at the location in a reasonable time) and “<i>breakdown</i>” (in case of a mechanical problem on the vehicle preventing the taxi to continue operating). This information will be visible to the search engine querying the <i>hail</i>.</p>
rating_ride	Integer (from 1 to 5)	<p>Rating of the ride by the client.</p> <p>This field should be used by “search engines” during or after the ride to let customers rate the ride between 1 and 5 stars. A ride can be rated multiple times, but only the latest rating will be considered.</p>

<code>rating_ride_reason</code>	string	<p>Explanation of the rating of the ride by the client.</p> <p>This field should be used by “search engines” during or after the ride to let the client explain low ratings of the ride. It is recommended to ask customers for their “<code>rating_ride_reason</code>” when their <code>rating_ride</code> is 3 stars or less. This information will not be individually transmitted to the taxi driver.</p> <p>The possible values are “<code>ko</code>” (the taxi never showed and/or the ride did not happen), <code>payment</code> (credit card refused, etc), <code>courtesy</code>, (general attitude problems, loud radio, etc), <code>route</code> (subpar itinerary, etc) and <code>cleanliness</code> (dirty car, cigarette smell, etc).</p>
<code>reporting_customer</code>	boolean	<p>Reporting of a problem encountered with a customer by a driver.</p> <p>This field should be used by “operators” during or after the ride to let the taxi driver inform the search engine that a problem happened with the client. In that case, the hail should be updated with a <code>reporting_customer</code> set to <code>True</code> and a <code>reporting_customer_reason</code> should be provided.</p>
<code>reporting_customer_reason</code>	string	<p>Explanation of the problem encountered with a customer by a driver.</p> <p>This field should be used by “operators” during or after the ride to let taxi drivers explain the type of problem they encountered with a client. It is only required if “<code>reporting_customer</code>” is set to <code>True</code>.</p> <p>The possible values are “<code>ko</code>” (the client was nowhere to be seen and/or the ride did not happen), “<code>payment</code>” (unpaid ride, bargaining, etc), <code>courtesy</code> (general attitude problems, etc), <code>route</code> (non existing destination address, etc) and <code>cleanliness</code> (dirty luggages, cigarettes, etc).</p>

5.2.2.1 Updating a Hail

As the hail transaction progresses, the search engine must update the hail status. This is done through a call of the HTTPS PUT request to the “hails” API.

See section [5.2 Hail Tests](#) for more details on when and why the search engine must update the hail status.

You do have to check the response of the PUT request to ensure that the update went as expected. For example, if the PUT request was not received in a timely fashion, the hail status may be "timeout_customer" instead of the expected status. If the transaction reaches an unexpected end state (ex: timeout_customer, failure, etc.), the whole hail transaction will be considered cancelled and a new hail transaction must be restarted.

```
PUT /api/motor/hails/{hail_id}
```

Parameters

Path

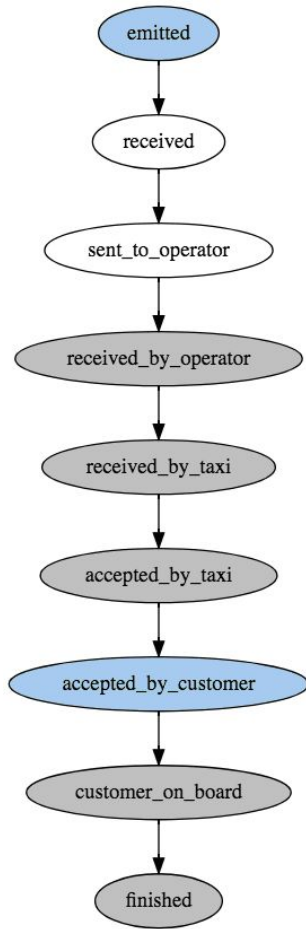
Hail_id (string)(required)

*Body (JSON) ** Send only one item at a time*

```
{  
  "data": [  
    {  
      "status": "accepted_by_customer"  
    }  
  ]  
}
```


5.2.3 Happy Path

5.2.3.1 Graphical representation



5.2.3.2 Details

The client hails a taxi through the search engine.

```
POST /api/motor/hails/ ** Send only one item at a time
{
  "data": [
    {
      "customer_lat": 45.495,
      "customer_lon": -73.554,
      "customer_address": "70 Jarry",
      "taxi_id": "{taxild}",
      "customer_phone_number": "514 201-4454",
      "opérateur": "coop",
      "customer_id": "anonymous"
    }
  ]
}
```

The operator receives a hail request from the TXP.

Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"received_by_taxi"}]}
```

When the driver accepts the hail within 30 seconds, the operator notifies the TXP.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"accepted_by_taxi "}]}
```

When the client accepts the hail within 20 seconds, the search engine notifies the TXP.

```
PUT /api/motor/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"accepted_by_customer"}]}
```

When the operator receives the information that the client is on board, the operator notifies the TXP.

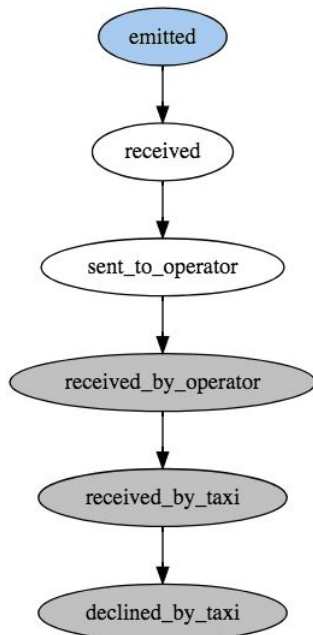
```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"customer_on_board"}]}
```

When the client leaves the taxi, the operator notifies the TXP.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"finished"}]}
```

5.2.4 Declined by taxi

5.2.4.1 Graphical representation



5.2.4.2 Details

The client hails a taxi through the search engine.

*POST /api/motor/hails/ ** Send only one item at a time*

```
{
  "data": [
    {
      "customer_lat": 45.495,
      "customer_lon": -73.554,
      "customer_address": "70 Jarry",
      "taxi_id": "{taxid}",
      "customer_phone_number": "514 201-4454",
      "opérateur": "coop",
      "customer_id": "anonymous"
    }
  ]
}
```

The operator receives a hail request from the TXP.

Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail.

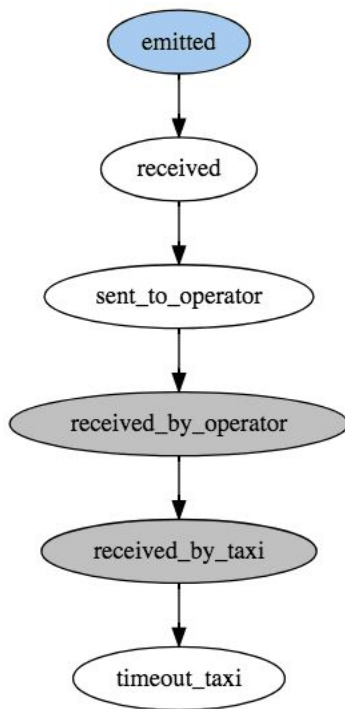
```
PUT /api/hails/{hailId} ** Send only one item at a time
{"data":[{"status":"received_by_taxi"}]}
```

When the driver declines the hail within 30 seconds, the operator notifies the TXP.

```
PUT /api/hails/{hailId} ** Send only one item at a time
{"data":[{"status":"declined_by_taxi"}]}
```

5.2.5 Accepted by taxi after timeout

5.2.5.1 Graphical representation



5.2.5.2 Details

The client hails a taxi through the search engine.

```
POST /api/motor/hails/ ** Send only one item at a time
{
  "data": [
    {
      "customer_lat": 45.495,
      "customer_lon": -73.554,
      "customer_address": "70 Jarry",
      "taxi_id": "{taxiid}",
      "customer_phone_number": "514 201-4454",
      "opérateur": "coop",
      "customer_id": "anonymous"
    }
  ]
}
```

The operator receives a hail request from the TXP.

Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"received_by_taxi"}]}
```

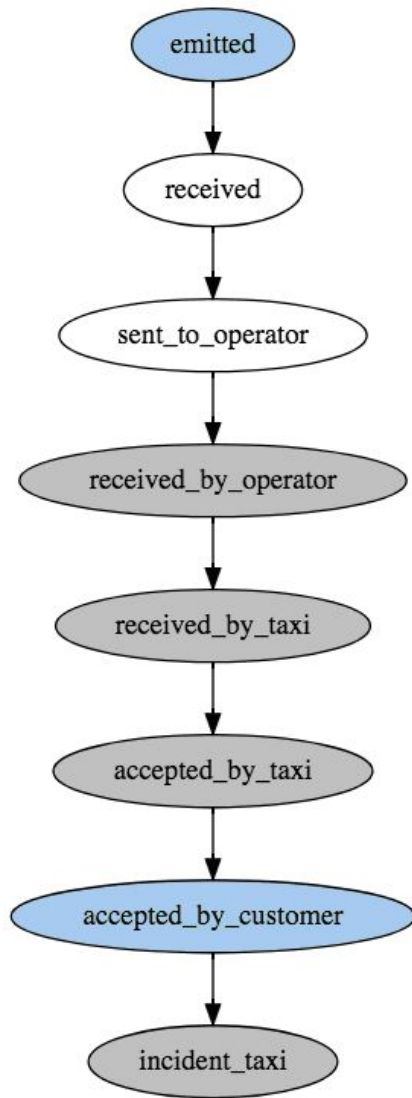
When the driver accepts the hail after 30 seconds, the operator notifies the TXP.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"accepted_by_taxi"}]}
```

The operator notifies the driver that he/she had not answered in a timely fashion and that the hail has been canceled. As explained in section 4.2.2 “Updating a Hail” from the Integration documentation, the operator must always check the value of the status attribute in the response to ensure that the state transition completed as expected.

5.2.6 *Canceled by taxi*

5.2.6.1 Graphical representation



5.2.6.2 Details

The client hails a taxi through the search engine.

```
POST /api/motor/hails/ ** Send only one item at a time
{
  "data": [
    {
      "customer_lat": 45.495,
      "customer_lon": -73.554,
      "customer_address": "70 Jarry",
      "taxi_id": "{taxid}",
      "customer_phone_number": "514 201-4454",
      "opérateur": "coop",
      "customer_id": "anonymous"
    }
  ]
}
```

The operator receives a hail request from the TXP.

Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"received_by_taxi"}]}
```

When the driver accepts the hail within 30 seconds, the operator notifies the TXP.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"accepted_by_taxi"}]}
```

When the client accepts the hail within 20 seconds, the search engine notifies the TXP.

```
PUT /api/motor/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"accepted_by_customer"}]}
```

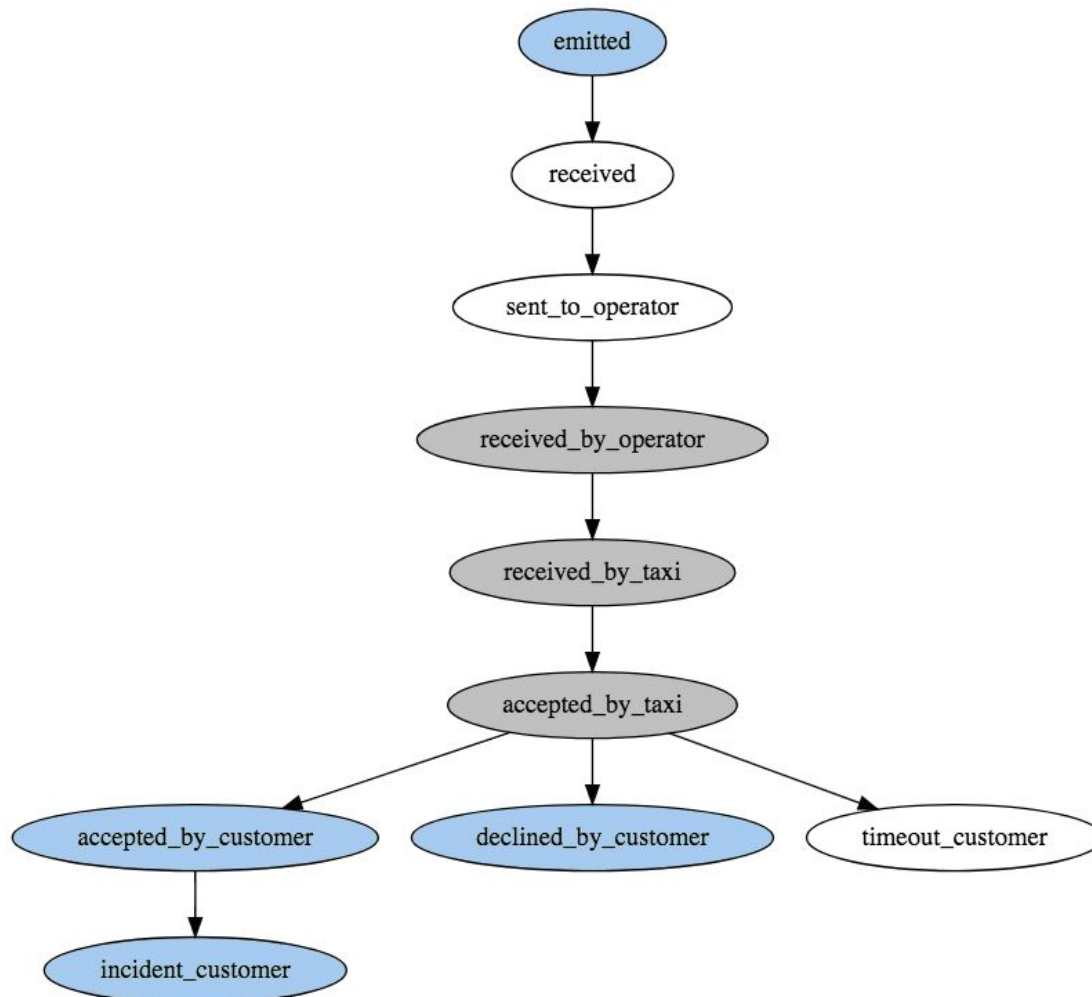
When an incident occurs that prompts the driver to cancel the hail before having the client on board, the operator notifies the TXP and specifies the reason why the hail was cancel. See section 4.1 “Hail Status” from the Integration documentation for the list of possible reasons.

*PUT /api/hails/{hailId} ** Send only one item at a time*

```
{  
  "data": [  
    {  
      "status": "incident_taxi",  
      "incident_taxi_reason": "breakdown",  
    }  
  ]  
}
```

5.2.7 Canceled by client

5.2.7.1 Graphical representation



5.2.7.2 Details

The client hails a taxi through the search engine.

```
POST /api/motor/hails/ ** Send only one item at a time
{
  "data": [
    {
      "customer_lat": 45.495,
      "customer_lon": -73.554,
      "customer_address": "70 Jarry",
      "taxi_id": "{taxild}",
      "customer_phone_number": "514 201-4454",
      "opérateur": "coop",
      "customer_id": "anonymous"
    }
  ]
}
```

The operator receives a hail request from the TXP.

Within 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"received_by_taxi"}]}
```

When the driver accepts the hail within 30 seconds, the operator notifies the TXP.

```
PUT /api/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"accepted_by_taxi"}]}
```

Once the hail is in the state `accepted_by_taxi`, the operator must poll the TXP each 30 seconds in order to ensure that the hail is not in the state `incident_customer`, `declined_by_customer` or `timeout_customer`. The operator can stop polling once the hail reaches one of these states: `customer_on_board`, `incident_taxi`, `incident_customer`, `declined_by_customer` or `timeout_customer`.

```
GET /api/hails/{hailid}
```

When the client accepts the hail within 20 seconds, the search engine notifies the TXP.

```
PUT /api/motor/hails/{hailid} ** Send only one item at a time
{"data":[{"status":"accepted_by_customer"}]}
```

When an incident occurs that prompts the client to cancel the hail before being on board, the search engine notifies the TXP.

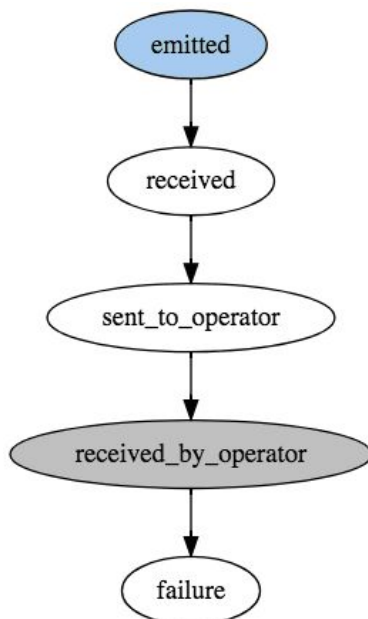
```
PUT /api/motor/hails/{hailId} ** Send only one item at a time
{"data":[{"status":"incident_customer"}]}
```

When the operator gets the information that the hail was canceled by the client, the operator notifies the driver.

```
GET /api/hails/{hailId}
```

5.2.8 Failure example

5.2.8.1 Graphical representation



5.2.8.2 Details

The client hails a taxi through the search engine.

```
POST /api/motor/hails/ ** Send only one item at a time
```

```
{
  "data": [
    {
      "customer_lat": 45.495,
      "customer_lon": -73.554,
      "customer_address": "70 Jarry",
      "taxi_id": "{taxild}",
      "customer_phone_number": "514 201-4454",
      "opérateur": "coop",
      "customer_id": "anonymous"
    }
  ]
}
```

The operator receives a hail request from the TXP.

After 10 seconds, the operator notifies the TXP that they asked if the driver wanted to accept the hail.

```
PUT /api/hails/{hailid}
{"data":[{"status":"received_by_taxi"}]}
```

The operator notifies the driver that a technical problem occurred and that the hail is canceled. As explained in section 4.2.2 “Updating a Hail” from the Integration documentation, the operator must always check the value of the status attribute in the response to ensure that the state transition completed as expected.

```
GET /api/hails/{hailid}
```

5.3 Bill 17 Tests

This section is intended for the operator’s IT staff responsible for implementing the changes, required by Bill 17, in the operator’s IT system. This section presents all the information and the links required to implement these changes.

To understand how these changes impact the operator at the business level, see the document [Guide d’accompagnement loi 17](#).

For information on how to read the tests presented in this section, see *Steps grouping* and *Pictograms* in section 5.

The operator’s IT staff is responsible for verifying that the changes required by Bill 17 are correctly implemented in the operator’s IT system.

The operator's IT staff is responsible for deciding when to deploy these changes to the production environment. To ensure proper migration, tests can be performed, at will, in the acceptance environment at: <https://taximtl.accept.ville.montreal.qc.ca>.

BTM's staff remains available for answering any related questions at: support.taxi.exchange.point@montreal.ca.

If needed, The operator's IT staff can ask BTM's staff to help make sure that changes, required by Bill 17, are properly implemented in operators' IT system by verifying tests results in the acceptance environment.

When available, vehicle_identification_number must be transmitted




See section 2.2 for more information.





Obligation to continue transmitting vehicle's positions




In accordance with the Act regulating the remunerated transport of persons by automobile, vehicles must remain connected to the Taxi register of the Bureau du taxi de Montréal (BTM). During the transition period (October 10, 2020 to March 31, 2021), positions of non-migrated vehicles **must continue** to be transmitted as long as the position's transmission of migrated vehicles is not active.



5.3.1 Migrate a driver when the vehicles he drives have not been migrated yet

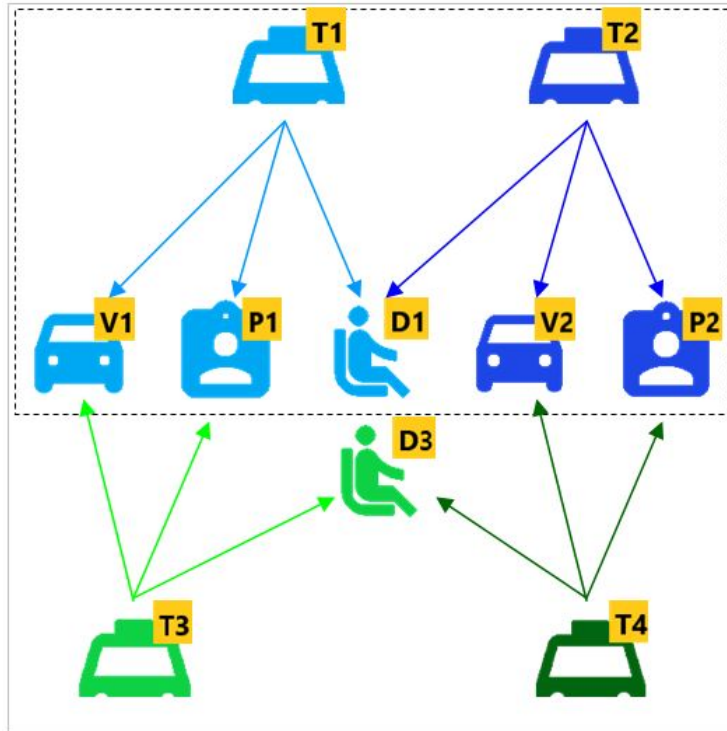
This migration can be performed as soon as possible. There are no prerequisites.

Step	Action	Request	Response	Icon
Initial State				
1	Create a driver with Montréal-600 department.	<pre>POST /api/drivers { "data": [{ "first_name": "John", "last_name": "Doe", "professional_licence": "00011", "departement": { "nom": "Montréal", "numero": "660" } }] }</pre>	HTTP 201 Created	
2	Create a vehicle with a 'T' licence plate.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "T00011A", ... }] }</pre>	HTTP 201 Created	
3	Create a permit (ADS)	<pre>POST /api/ads { "data": [{ "insee": "102005", "numero": "4M000000011A", "vdm_vignette": "5511", ... }] }</pre>	HTTP 201 Created	

4	Create the first taxi driven by D1 and link it to V1 and P1.	<pre> POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000011A" }, "vehicle": { "licence_plate": "T00011A" }, "driver": { "departement": "660", "professional_licence": "00011" }, ... }] } </pre>	HTTP 201 Created	
5	Create a second vehicle with 'T' licence plate.	<pre> POST /api/vehicles { "data": [{ "licence_plate": "T00012B", ... }] } </pre>	HTTP 201 Created	
6	Create a second permit (ADS)	<pre> POST /api/ads { "data": [{ "insee": "102005", "numero": "4M000000012B", "vdm_vignette": "5512", ... }] } </pre>	HTTP 200 OK	
7	Create the second taxi driven by D1 and link it to V2 and P2.	<pre> POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000012B" }, "vehicle": { "licence_plate": "T00012B" }, "driver": { "departement": "660", "professional_licence": "00011" }, ... }] } </pre>	HTTP 201 Created	

Test				
8	<p>In accordance with Bill 17, create a new driver using his driving license number and Québec-1000 department.</p> <p>For more details on how to submit a driver in accordance with Bill 17, see the description of department and professional licence in section 2.1.</p>	<pre>POST /api/drivers { "data": [{ "departement": { "nom": "Québec", "numero": "1000" }, "first_name": "John", "last_name": "Doe", "professional_licence": "L0006-221166-01" }] }</pre>	HTTP 201 Created	
9	<p>Create a taxi using the driver in accordance with Bill 17 (D3) and the first vehicle that have not been migrated yet (V2 with P2).</p>	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000011A" }, "vehicle": { "licence_plate": "T00011A" }, "driver": { "departement": "1000", "professional_licence": "L0006-221166-01" }, ... }] }</pre>	<pre>HTTP 201 Created { "data": [{ "id": "\$T3", ... }] }</pre>	
10	<p>Start sending taxi positions and status with the taxi created (T3).</p>	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T3", ... }] }</pre>	HTTP 200 OK	

<p>11</p>	<p>A driver may drive many vehicles. Make sure to create a new taxi for each vehicle driven by the driver.</p> <p>Create a second taxi using the driver in accordance with Bill 17 (D3) and the second vehicle that have not been migrated yet (V2 with P2)</p>	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000012B" }, "vehicle": { "licence_plate": "T00012B" }, "driver": { "departement": "1000", "professional_licence": "L0006-221166-01" }, ... }] }</pre>	<p>HTTP 201 Created</p> <pre>{ "data": [{ "id": "\$T4", ... }] }</pre>	
<p>12</p>	<p>Start sending taxi positions and status with the taxi created (T4).</p>	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T4", ... }] }</pre>	<p>HTTP 200 OK</p>	






Entities present in the system after the migration




5.3.2 Migrate a vehicle when the drivers who drive it have been migrated





This migration can be performed when an owner transmits the new license plate in conformity with Bill 17 (No T license plate).

Before performing this migration, the migration 5.3.1 must have been performed for all the drivers who drive the vehicle.

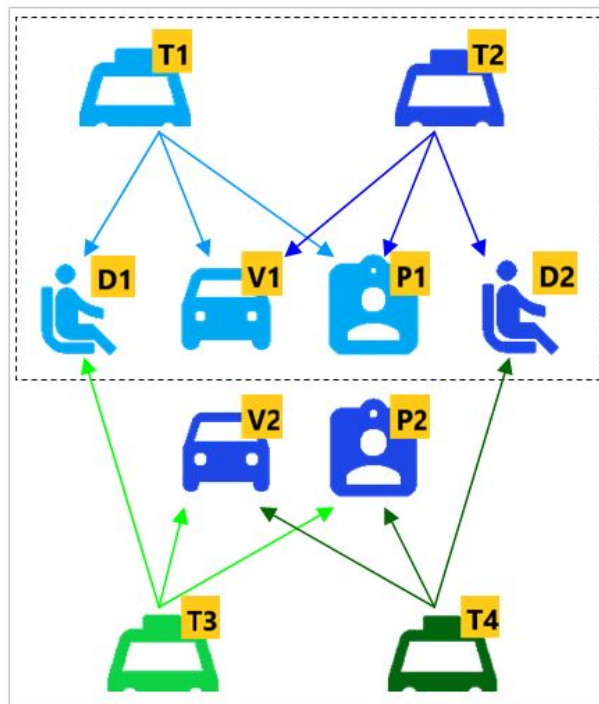
Note that an owner can change the license plate for reasons unrelated to Bill 17. See section 5.1.1 for more information.

Step	Action	Request	Response	Icon
Initial State				
1	To emulate the results of the migration described section 5.3.1, create a new driver using his driving license number and Québec-1000 department. For more details on how to submit a driver in accordance with Bill 17, see the description of department and professional licence in section 2.1.	<pre>POST /api/drivers { "data": [{ "departement": { "nom": "Québec", "numero": "1000" }, "first_name": "John", "last_name": "Doe", "professional_licence": "L1006-221166-11" }] }</pre>	HTTP 201 Created	
2	Create a vehicle with a 'T' licence plate.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "T00011A", "type_": "sedan", "constructor": "audi", "model": "a4" }] }</pre>	HTTP 201 Created	
3	Create a permit (ADS)	<pre>POST /api/ads { "data": [{ "insee": "102005", "numero": "4M000000011A", "vdm_vignette": "5511", "owner_name": "Taxi-Pro", }] }</pre>	HTTP 201 Created	

4	Create the first taxi driven by D1 and link it to V1 and P1.	<pre> POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000011A" }, "vehicle": { "licence_plate": "T00011A" }, "driver": { "departement": "1000", "professional_licence": "L1006-221166-11" }, ... }] } </pre>	HTTP 201 Created	
5	<p>To emulate the results of the migration described section 5.3.1, create a second driver using his driving license number and Québec-1000 department.</p> <p>For more details on how to submit a driver in accordance with Bill 17, see the description of department and professional licence in section 2.1</p>	<pre> POST /api/drivers { "data": [{ "departement": { "nom": "Québec", "numero": "1000" }, "first_name": "Jane", "last_name": "Din", "professional_licence": "L2006-221166-22" }] } </pre>	HTTP 201 Created	
6	Create the second taxi driven by D2 and link it to V1 and P1.	<pre> POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000011A" }, "vehicle": { "licence_plate": "T00011A" }, "driver": { "departement": "1000", "professional_licence": "L2006-221166-22" }, ... }] } </pre>	HTTP 201 Created	
Test				

7	<p>In accordance with Bill 17, create a vehicle identical to V1 but with the new licence plate.</p> <p>For more details on how to submit a vehicle in accordance with Bill 17, see the description of licence_plate in section 2.2</p>	<pre>POST /api/vehicles { "data": [{ "licence_plate": "FAA0012", "type_": "sedan", "constructor": "audi", "model": "a4" }] }</pre>	HTTP 201 Created	
8	<p>In accordance with Bill 17, create a permit/ADS identical to P1 but with zone Québec-1000 and SAAQ file number.</p> <p>For more details on how to submit a permit/ADS in accordance with Bill 17, see the description of insee and numero in section 2.3</p>	<pre>POST /api/ads { "data": [{ "insee": "1000", "numero": "161000012", "owner_name": "Taxi-Pro", }] }</pre>	HTTP 200 OK	
9	<p>Create a taxi using driver D1, vehicle V2 and permit/ADS P2.</p>	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "1000", "numero": "161000012" }, "vehicle": { "licence_plate": "FAA0012" }, "driver": { "departement": "1000", "professional_licence": "L1006-221166-11" }, }] }</pre>	<pre>HTTP 201 Created { "data": [{ "id": "\$T3", }] }</pre>	
10	<p>Start sending taxi positions and status with the taxi created (T3).</p>	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T3", }] }</pre>	HTTP 200 OK	

11	Create a taxi using driver D2, vehicle V2 and permit/ADS P2.	<pre> POST /api/taxis { "data": [{ "ads": { "insee": "1000", "numero": "161000012" }, "vehicle": { "licence_plate": "FAA0012" }, "driver": { "departement": "1000", "professional_licence": "L2006-221166-22" }, ... }] } </pre>	HTTP 201 Created <pre> { "data": [{ "id": "\$T4", ... }] } </pre>	
12	Start sending taxi positions and status with the taxi created (T4).	<pre> POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T4", ... }] } </pre>	HTTP 200 OK	



Entities present in the system after the migration



5.3.3 Migrate a vehicle and the drivers who drive it at the same time





This migration can be performed as soon as an owner transmits the new license plate in conformity with Bill 17 (No T license plate).





This scenario is an alternative to scenarios described in sections 5.3.1 and 5.3.2. If this scenario does not simplify the changes required by Bill 17 in the operator's IT system, then just ignore it and use the two-step migration as described in sections 5.3.1 and 5.3.2 instead.





This scenario is more complex, because during the transition period, a driver can drive the migrated vehicle and the non-migrated vehicle. For this scenario to succeed, the operator's IT system must be able to continue to identify the driver by the pocket number when he is driving the non-migrated vehicle and identify the same driver by his driving license number when he drives the migrated vehicle.

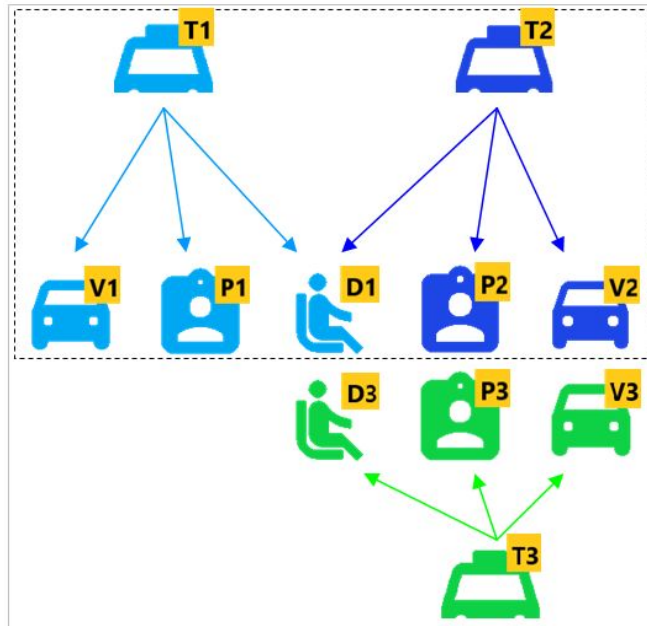
Note that in order to keep this scenario simple, it presents the case where the migrated vehicle is driven by a single driver. **However, the operator's IT system must also support the scenario where the migrated vehicle is driven by multiple drivers.**

Step	Action	Request	Response	Icon
Initial State				
1	Create a driver with Montréal-600 department.	<pre>POST /api/drivers { "data": [{ "first_name": "John", "last_name": "Doe", "professional_licence": "00011", "departement": { "nom": "Montréal", "numero": "660" } }] }</pre>	HTTP 201 Created	
2	Create a vehicle with a 'T' licence plate.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "T00011A", "type_": "sedan", "constructor": "audi", "model": "a4" }] }</pre>	HTTP 201 Created	

3	Create a permit (ADS)	<pre>POST /api/ads { "data": [{ "insee": "102005", "numero": "4M000000011A", "vdm_vignette": "5511", ... }] }</pre>	HTTP 201 Created	
4	<p>Create the first taxi driven by D1 and link it to V1 and P1.</p> <p>This taxi will remain unmigrated at the end of this test.</p>	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000011A" }, "vehicle": { "licence_plate": "T00011A" }, "driver": { "departement": "660", "professional_licence": "00011" }, ... }] }</pre>	<pre>HTTP 201 Created { "data": [{ "id": "\$T1", ... }] }</pre>	
5	Start sending taxi positions and status with the taxi that will not be migrated. (T1)	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T1", ... }] }</pre>	HTTP 200 OK	
6	Create a second vehicle with a 'T' licence plate.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "T00022B", "type_": "mpv", "constructor": "toyota", "model": "camry" ... }] }</pre>	HTTP 201 Created	

7	Create a second permit (ADS)	<pre>POST /api/ads { "data": [{ "insee": "102005", "numero": "4M000000022B", "vdm_vignette": "5522", ... }] }</pre>	HTTP 201 Created	
8	<p>Create a second taxi driven by driver D1 and link it to V2 and P2.</p> <p>This taxi will be migrated in the Test section.</p>	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "102005", "numero": "4M000000022B" }, "vehicle": { "licence_plate": "T00022B" }, "driver": { "departement": "660", "professional_licence": "00011" }, ... }] }</pre>	<pre>HTTP 201 Created { "data": [{ "id": "\$T2", ... }] }</pre>	
Test				
9	<p>In accordance with Bill 17, create a driver, identical to D1 but using his driving license number and Québec-1000 department.</p> <p>For more details on how to submit a driver in accordance with Bill 17, see the description of department and professional licence in section 2.1</p>	<pre>POST /api/drivers { "data": [{ "departement": { "nom": "Québec", "numero": "1000" }, "first_name": "John", "last_name": "Doe", "professional_licence": "L3006-221166-33" }] }</pre>	HTTP 201 Created	
10	In accordance with Bill 17, create a vehicle, identical to V2 but using a license plate without T prefix.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "FCC0013", "type_": "mpv", "constructor": "toyota", "model": "camry" ... }] }</pre>	HTTP 201 Created	

<p>11</p>	<p>In accordance with Bill 17, create a permit/ADS identical to P2 but with zone Québec-1000 and SAAQ file number.</p> <p>For more details on how to submit a permit/ADS in accordance with Bill 17, see the description of insee and numero in section 2.3</p>	<pre>POST /api/ads { "data": [{ "insee": "1000", "numero": "163000013", "owner_name": "Taxi-Pro", ... }] }</pre>	<p>HTTP 200 OK</p>	
<p>12</p>	<p>Create a taxi driven by driver D3 and link it to V3 and P3.</p>	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "1000", "numero": "163000013" }, "vehicle": { "licence_plate": "T00013C" }, "driver": { "departement": "1000", "professional_licence": "L3006-221166-33" }, ... }] }</pre>	<p>HTTP 201 Created</p> <pre>{ "data": [{ "id": "\$T3", ... }] }</pre>	
<p>13</p>	<p>Start sending taxi positions and status with the taxi created (T3).</p>	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T3", ... }] }</pre>	<p>HTTP 200 OK</p>	
<p>14</p>	<p>Taxi T1 continues to send positions and status. T1 remains unmigrated.</p> <p>At this point John Doe drives a migrated vehicle (V3) as driver D3 and an unmigrated vehicle (V1) as driver D1.</p>	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T1", ... }] }</pre>	<p>HTTP 200 OK</p>	



Entities present in the system after the migration

5.3.4 Unallowed migration paths

Owners will not all regularize at the same time their situation with SAAQ. During the transition period, certain vehicles will be migrated and others will not. However, when a taxi is linked to an owner (ADS) in the Québec-1000 zone, that taxi must be fully migrated. (Driver, vehicle and owner/license/ADS)

1. It is not possible to migrate an owner (ADS) without migrating the drivers that drive the vehicle belonging to that owner.

As soon as possible, drivers must send their driver's license number to the operator.

If the owner (ADS) is in the Québec-1000 zone, then a linked driver must be in the Québec-1000 department. Otherwise a http 400 error will occur.

2. It is not possible to migrate the owner without migrating the vehicle.





Following the license plate change, the owner must communicate his SAAQ file number and his new license plate number to the operator.





If the owner is in the Québec-1000 zone, then the license plate must not start with a T; otherwise a http 400 error will occur.

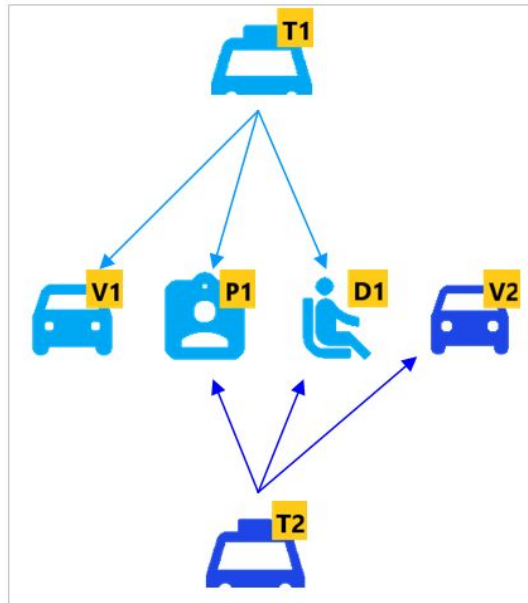
5.3.5 Many vehicles can have the same owner

As described in section 2.3, following adoption of Bill 17, the meaning of ADS has changed from permit to owner's license. This example illustrates this change. Please make sure this change is well supported by the operator's IT system.

To simplify, all vehicles will be driven by the same driver.

Test				
1	Create a driver in the Québec-1000 department.	<pre>POST /api/drivers { "data": [{ "departement": { "nom": "Québec", "numero": "1000" }, "first_name": "John", "last_name": "Doe", "professional_licence": "L1006-221166-11" }] }</pre>	HTTP 201 Created	
2	Create a vehicle.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "FAA0011", ... }] }</pre>	HTTP 201 Created	
3	Create a second vehicle.	<pre>POST /api/vehicles { "data": [{ "licence_plate": "FBB0022", ... }] }</pre>	HTTP 201 Created	
4	<p>Create a permit/ADS in the zone Québec-1000 and SAAQ file number.</p> <p>For more details on how to submit an ADS in accordance with Bill 17, see the description of insee and numero in section 2.3</p>	<pre>POST /api/ads { "data": [{ "insee": "1000", "numero": "161000011", "owner_name": "Taxi-Pro", ... }] }</pre>	HTTP 200 OK	

5	Create a taxi driven by driver D1 and link it to V1 and P1.	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "1000", "numero": "161000011" }, "vehicle": { "licence_plate": "FAA0011" }, "driver": { "departement": "1000", "professional_licence": "L1006-221166-11" }, ... }] }</pre>	<pre>HTTP 201 Created { "data": [{ "id": "\$T1", ... }] }</pre>	
6	Create a second taxi driven by driver D1 and link it to V2 and P1.	<pre>POST /api/taxis { "data": [{ "ads": { "insee": "1000", "numero": "161000011" }, "vehicle": { "licence_plate": "FBB0022" }, "driver": { "departement": "1000", "professional_licence": "L1006-221166-11" }, ... }] }</pre>	<pre>HTTP 201 Created { "data": [{ "id": "\$T2", ... }] }</pre>	
7	Start sending taxi positions and status with the taxi T1.	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T1", ... }] }</pre>	<pre>HTTP 200 OK</pre>	
8	Start sending taxi positions and status with the taxi T2.	<pre>POST /api/taxi-position-snapshots { "items": [{ "taxi": "\$T2", ... }] }</pre>	<pre>HTTP 200 OK</pre>	



Entities present in the system after the test