Montréal

Plateform integration

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

Montréal

Plateform integration

TABLE OF CONTENTS

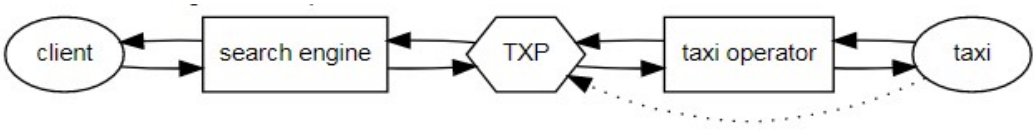
Montréal

Plateform integration

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 include 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 acceptation 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 acceptation 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@ville.montreal.qc.ca

Here are the links to communicate with the TXP services:
Acceptation : 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

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 is JSON. Note that data is isolated for each operator, no operator can see other operator's data.

2. CONTEXTUAL DATA

2.1 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.

```
POST /api/vehicles
```

Parameters

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

```
{
  "data": [
    {
      "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,
      "licence_plate": "COO-011",
      "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


```
{
  "data": [
    {
      "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,
      "licence_plate": "COO-011",
      "luxury": true,
      "model": "a4",
      "model_year": 1995,
      "nb_seats": 0,
      "nfc_cc_accepted": true,
      "pet_accepted": true,
      "private": false,
      "relais": true,
      "special_need_vehicle": true,
      "tablet": true,
      "taximetre": "aa",
      "type_": "sedan",
      "wifi": true
    }
  ]
}
```


Key	Value Type	Description
licence_plate	String	*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 Quebec, the licence plate is an alphanumeric combination of 6 characters.***
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 vehicle 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).* *Vehicle 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 purpose 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*

		purpose 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 purpose 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 purpose 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 purpose 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 purpose 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 purpose 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 purpose 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 purpose 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 purpose 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 Engine 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.*

2.2 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.

```
POST /api/drivers
```

Parameters

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

```
  "data": [
    {
      "birth_date": "1950-12-22",
      "departement": {
        "nom": "Montréal",
        "numero": "660"
      },
      "first_name": "Jon",
      "last_name": "Doe",
      "professional_licence": "XT0001"
    }
  ]
}
```


Response (JSON) status 200


```
{
  "data": [
    {
      "birth_date": "1950-12-22",
      "departement": {
        "nom": "Montréal",
        "numero": "660"
      },
      "first_name": "Jon",
      "last_name": "Doe",
      "professional_licence": "XT0001"
    }
  ]
}
```


Key	Value Type	Description
departement	department object	*The departement object is constituted of the identifier numero and the name (nom) of the local*

		authority. *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.* ***For Quebec, departement should always be set to 660-Montréal as shown in the above example.***
professional_licence	string	*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.* *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).* *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.* ***For Quebec, the professional_licence is the 'pocket number'. Proposed structure: XXXXX (a five digit code 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.***

2.3 Registering a 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 license (ADS) simply by submitting the updated ads object with the post method. If the insee or numero is different, a new license (ADS) will be created; if the insee and numero are unchanged, the license (ADS) will be updated.

```
POST /api/ads
```

Parameters *** Send only one item at a time***

Body (JSON)

```
{
  "data": [
    {
      "category": "",
      "vehicle_id": 36,
      "insee": "102005",
      "numero": "00011",
      "owner_name": "Co-op",
      "owner_type": "company",
      "doublage": false,
      "vdm_vignette": "string"
    }
  ]
}
```

Response (JSON) status 200

```
{
  "data": [
    {
      "category": "",
```

```
      "doublage": false,
      "insee": "102005",
      "numero": "00011",
      "owner_name": "Co-op",
      "owner_type": "company",
      "vehicle_id": 36,
      "vdm_vignette": "string"
    }
  ]
}
```


Key	Value Type	Description
insee	string	*Identifier of the local authority who attributed the license (ADS).* *See to the Taxi Zone section of this document for reference.* *It is composed of 5 characters (usually 5 digits, except for Corsica where the second character can be "A" or "B") and uniquely identify each municipality (or just two digits for the rare special case of license attributed by departments usually for airports). There are three special cases for Paris (insee code: 75056, the codes of the subdivisions -aka arrondissements- are not used), Lyon (insee code: 69123, idem) and Marseille (insee code: 13055, idem).* *Warning: the INSEE code is different from the zipcode (there are very often several municipalities in a single zipcode and in some cases several zipcodes in a single municipality). The zipcode/INSEE code correspondence can be found in the Base officielle des codes postaux. The usual way to get an INSEE code from a user is to ask for a zipcode, then offer to choose the municipality in a list of the municipalities of the zipcode.* *For Quebec, similarly to the zupc_id, this field*

		represents the six digits number assigned by the CTQ to the agglomeration that a taxi is allowed to operate. ***Three agglomerations exist for Montreal as follow:*** ***102005 : A5 – Eastern part of the island of Montreal*** ***102011 : A11 – Downtown/center Montreal*** ***102012 : A12- West part of the island of Montreal*** ***See section 5.1 for more details.***
numero	string	*This is the taxi license number (ADS number).* *It is often a string of digits but it might for some municipalities contain letters or other characters like dash or slashes.* *Warning: this identifier is not unique at the national level: two local authorities can each assign the same number to different licensees.* *The couple of this license number (numero) and the licensing authority (insee) is used as the license identifier when declaring a taxi as a vehicle/driver/license triplet.* ***For Quebec: number represents the taxi license number assigned by the CTQ (registration certificate). Proposed structure: (12 alphanumeric characters).***
owner_name	string	*Name of the holder of the license.* *Warning: It might be either an individual or a company.*
owner_type	string	*The two possible values are company or individual.*
category	string	*This field is used for administrative purpose.* *When a new license (aka ADS) is created by an Operator, an empty string has to be passed (not a null value).*
doublage	boolean	*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).*
vehicle_id	integer	*This field is used for administrative purpose.* *When a new license (aka ADS) is created by an Operator, a "0" or "nul" has to be passed.*
vdm_vignette	string	***This field represent the "Vignette" number given by the BTM (Bureau Taxi Montreal). Mandatory.***

2.4 Declaring a taxi

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.

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.

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

```
POST /api/taxis
```

Parameters

```
Body (JSON) ** Send only one item at a time
{
  "data": [
    {
      "vehicle": {
         "licence_plate": "WM-220-VP"
```

```
      },
      "driver": {
        "departement": "660",
        "professional_licence": "XT0001"
      },
      "ads": {
        "insee": "102005",
        "numero": "00011"
      },
      "status": "free"
    }
  ]
}
```


Response (JSON) status 200


```
{
  "data": [
    {
      "ads": {
        "insee": "102005",
        "numero": "00011"
      },
        "driver": {
            "departement": "660",
        "professional_licence": "XT0001"
      },
      "id": "ueXs7TR",
      "last_update": null,
      "operator": null,
      "position": {
        "lat": null,
        "lon": null
      },
      "private": false,
```

```
      "rating": 4.5,
      "status": "free",
      "vehicle": {
        "characteristics": null,
        "color": null,
        "constructor": null,
        "licence_plate": "COO-011",
        "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`.
`driver`	driver	*A partial* **driver** *object with only the fields:* `departement`, `professionnal_licence`.
`id`	string	*A long-lived 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	*The name of the 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 batch every 5 seconds to keep to 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
`timestamp`	string	*Exact time at which the **location was determined by the taxi**, formatted as a Unix time (IEEE 1003.1-2008 POSIX).* ***Warning: as per the POSIX specification, this should be UTC time without any timezone information.***

		Warning: Do not send locations in the future (or older than 1 minute) as they will return a http 400 error.
`operator`	string	*Login of the certified operator.*
`taxi`	string	*The `id` of the taxi is the `id` that was sent back when the taxi was declared (see Declaring a taxi).*
`lat`	string	*Latitude of the taxi.* *This should be in JavaScript double precision floating-point format, with decimal separator ".".* *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).*
`lon`	string	*Longitude of the taxi.* *This should be in JavaScript double precision floating-point format, with decimal separator ".".* *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).*
`device`	string	*`phone`, `tablet`, `taximeter` or `otherdevice`.*
`status`	string	*Possible values: answering, free, occupied, off, oncoming or unavailable.* *Mandatory.* *For more details, see the table below.*
`version`	string	*"2" for now (geolocation version 2 of the API).*
`speed`	string	*The actual speed of the taxi (in km/h) (optional).*
`azimuth`	string	*The current orientation of the taxi (360°) (optional).*

3.2 Taxis Status

Value	Description	Mandatory
`answering`	*The taxi is currently answering to a hail.*	*No, use unavailable if the information is not available.*

`free`	*The taxi can be hailed.*	*yes*
`occupied`	*The taxi has a customer on board.*	*yes*
`off`	*The taxi is not logged in or did not update its location recently enough.*	*yes*
`oncoming`	*The taxi is on its way to meet a customer.*	*No, use unavailable if the information is not available.*
`unavailable`	*The taxi is logged in, but cannot be hailed.*	*yes*

3.3 Updating the status of a taxi

The status of the taxi should be sent to the TXP whenever there is a change of status from the operator. The possible status are 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",
            "private": "false"
        }
    ]
}
```

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 known the status, ads, driver and vehicle of a taxi.

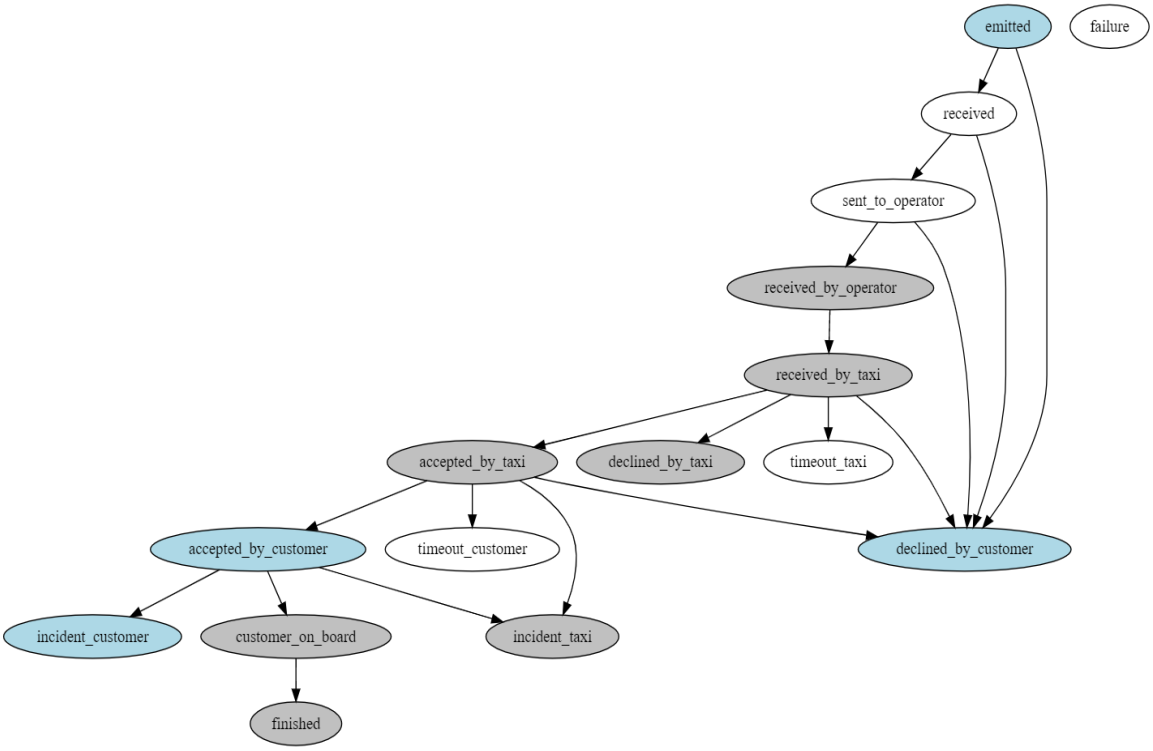
```
GET /api/taxis/{taxi_id}

Parameters

Path

taxi_id (string)(required)

Response (JSON) status 200

{
    "data": [
        {
            "ads": {
                "insee": "102011",
                "numero": "test-coop-z1"
            },
            "crowfly_distance": null,
            "driver": {
                "departement": "660",
                "professional_licence": "test-ops-z"
            },
            "id": "VsLwptA",
            "last_update": 1502819736,
            "operator": "coop",
            "position": {
                "lat": null,
                "lon": null
            },
            "private": false,
```

Plateform integration


```
            "rating": 4.42332039594968,
            "status": "answering",
            "vehicle": {
                "characteristics": [
                    "every_destination",
                    "gps",
                    "pet_accepted",
                    "bike_accepted",
                    "credit_card_accepted",
                    "luxury"
                ],
                "color": "GRISE",
                "constructor": "TOYOTA",
                "licence_plate": "test-ops-z2",
                "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 4.4 “Test Scenario” describe the scenario that the operator must support in order to receive hail from the TXP.

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 /hails/ API when a **search engine** creates a new hail.*
`received`	*The hail is received from the **search engine** by the TXP*	*The TXP changes the status to `received` after having sent back the complete hail (with the newly generated id) to the **search engine** and before forwarding the hail to the*

		***operator**. 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 `sent_to_operator` after forwarding the hail to the **operator** endpoint. Time before failure 10 seconds.*
`received_by_operator`	*The **operator** has acknowladged receiving the hail from the TXP*	*The TXP changes the status to `received_by_operator` 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 `received_by_taxi` by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been presented to the taxi driver. Time before timeout_taxi 30 seconds.*
`accepted_by_taxi`	*The hail has been accepted by the taxi driver*	*The "**operator**" should set the status of the hail to "`accepted_by_taxi`" by doing a "`PUT`" request on the /hails/{hail_id} API when the hail has been accepted by the taxi driver. Time before timeout_customer 10 minutes.*
`declined_by_taxi`	*The hail has been declined by the taxi driver*	*The "**operator**" should set the status of the hail to "`declined_by_taxi`" by doing a "`PUT`" request on the /hails/{hail_id} 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 `timeout_taxi` automatically after 30s have passed since the status was set to `received_by_taxi`.*
`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 20s*	*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:* • *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	*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.*

4.2 Faking a Search Engine

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

4.2.1 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"
      "operateur": "coop",
      "customer_id": "vdm_tester1"
    }
  ]
```

```
}
```

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": "vdm_tester1",
      "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	*Latitude of the position of the client.* *This should be in JavaScript double precision floating-point format, with decimal separator "."*
`customer_lon`	float	*Longitude of the position of the client.* *This should be in JavaScript double precision floating-point format, with decimal separator "."*
`customer_address`	string	*Address of the position of the client.* *This address will be used by the taxi driver to find the client.* *It should be displayed and validated by the client.* *Warning: In some cases, a POI might be more meaningful than a postal address.*
`customer_id`	string	*Identifier of the customer.* *This identifier is an identifier the Search Engine uses to uniquely identify the client. It can be a database id, an IMEI, a cookie identifier, a hash of a phone number, or any long lived identifier which will stay the same when the same client comes back.* *It does not need to carry any signification for the TXP: the TXP does not need the identity of the*

		client.
`customer_phone_number`	string	*Phone number of the client.* *This phone number might be used by the Operator of the taxi in case it proves difficult to find the client.*
`taxi_id`	string	*Identifier of the taxi the client is hailing.* *This identifier was returned by the TXP in the* Taxi *object.* *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.* *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`.*
`operateur`	string	*Identifier of the Operator of the taxi the client is hailing.* *This identifier was returned by the "TXP" in the "*Taxi*" object.*
`status`	status	*Status of the hail.* *All possible values are described* here
`id`	string	*Identifier of the hail.* *This identifier should be `null` 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.*
`taxi`	taxi	*Details of the taxi selected for the hail.*

`taxi_phone_number`	string	*Phone number the client should call in case of problem.* *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.*
`incident_customer_reason`	string	*Reason of the incident that prompted the client to cancel the ride.* *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 `incident_customer`.*
`incident_taxi_reason`	string	*Reason of the incident that prompted the taxi to cancel the ride.* *This field should be used by "**operators**" when setting the status of the ride to `incident_taxi`.* *The possible reasons are: "`no_show`" (when the client cannot be found), "`address`" (when the address cannot be found), "`traffic`" (when a traffic jam prevents the taxi to arrive at the location in a reasonable time) and "`breakdown`" (in case of a mechanical problem on the vehicle preventing the taxi to continue operating). This information will be visible too the search engine querying the `hail`.*
`rating_ride`	Integer (from 1 to 5)	*Rating of the ride by the client.* *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.*
`rating_ride_reason`	string	*Explanation of the rating of the ride*

		by the client. *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 "`rating_ride_reason`" when their `rating_ride` is 3 stars or less. This information will not be individually transmitted to the taxi driver.* *The possible values are "`ko`" (the taxi never showed and/or the ride did not happen), `payment` (credit card refused, etc), `courtesy`, (general attitude problems, loud radio, etc), `route` (subpar itinerary, etc) and `cleanliness` (dirty car, cigarette smell, etc).*
`reporting_customer`	boolean	*Reporting of a problem encountered with a customer by a driver.* *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 update with a `reporting_customer` set to `True` and a `reporting_customer_reason` should be provided.*
`reporting_customer_reason`	string	*Explanation of the problem encountered with a customer by a driver.* *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 "`reporting_customer`" is set to `True`.* *The possible values are "`ko`" (the client was nowhere to be seen and/or the ride did not happen), "`payment`" (unpaid ride, bargaining, etc), `courtesy` (general attitude problems, etc), `route` (non existing*

		destination address, etc) and `cleanliness` *(dirty luggages, cigarettes, etc).*

4.2.2 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 4.4 Test Scenario 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 be considered cancelled and a new hail transaction must be restarted.

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

Parameters

Path

Haild_id (string)(required)

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

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

4.3 Operator

4.3.1 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 Hailling 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.

4.3.2 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 reach the "customer_on_board" status.

```
GET /api/hails/{hail_id}

Parameters

Path

Hail_id (string)(required)
```

4.3.3 *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 4.4 Test Scenario, 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 be considered cancelled and a new hail transaction must be restarted.

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

Parameters

Path

Haild_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"
    }
  ]
}
```

4.4 Test scenario

All the scenario that the operator must support in order to receive hails from the Taxi Exchange Point (TXP) are described in this section. **Make sure you test them thoroughly.**

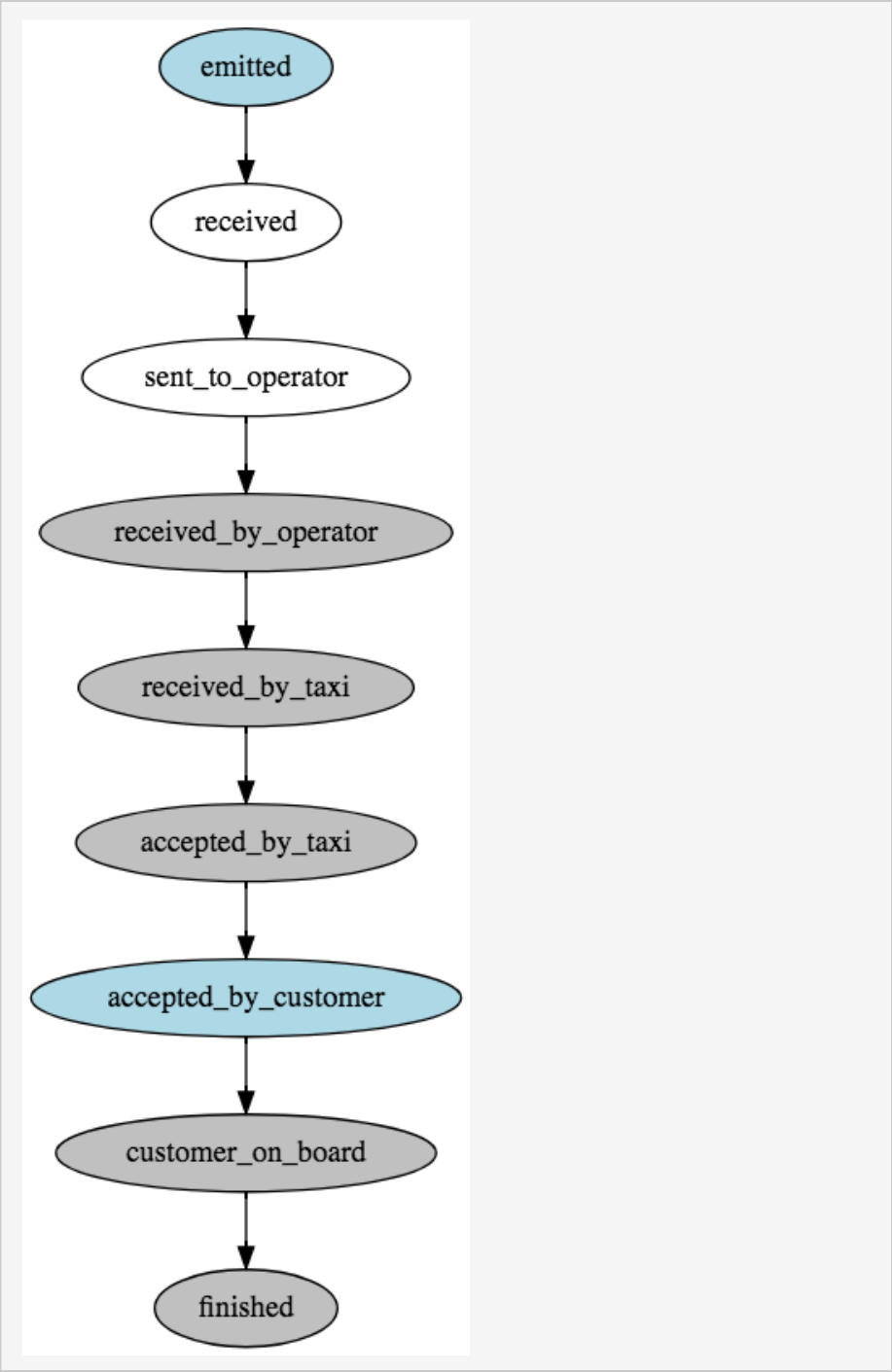
SCENARIO

4.4.1 Happy Path

4.4.1.1 Graphical representation

(See next page…)

4.4.1.1 Graphical representation (suite)



4.4.1.2 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",
        "operateur": "coop",
       "customer_id": "coop_user1"
      }
    ]
  }
```

- 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.

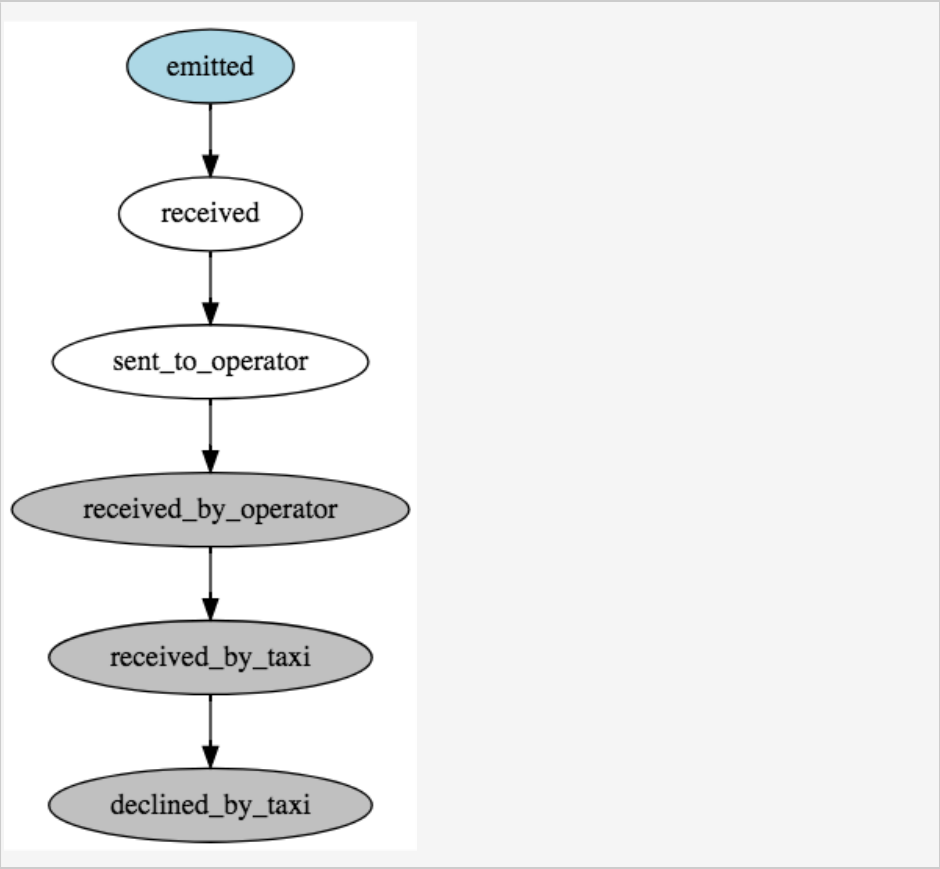
```
PUT /api/hails/{hailId} ** Send only one item at a time
{"data":[{"status":"customer_on_board"}]}
```

- 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"}]}
```

4.4.2 Declined by taxi

4.4.2.1 Graphical representation



4.4.2.2 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",
        "operateur": "coop",
        "customer_id": "coop_user1"
      }
    ]
  }
```

- 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.

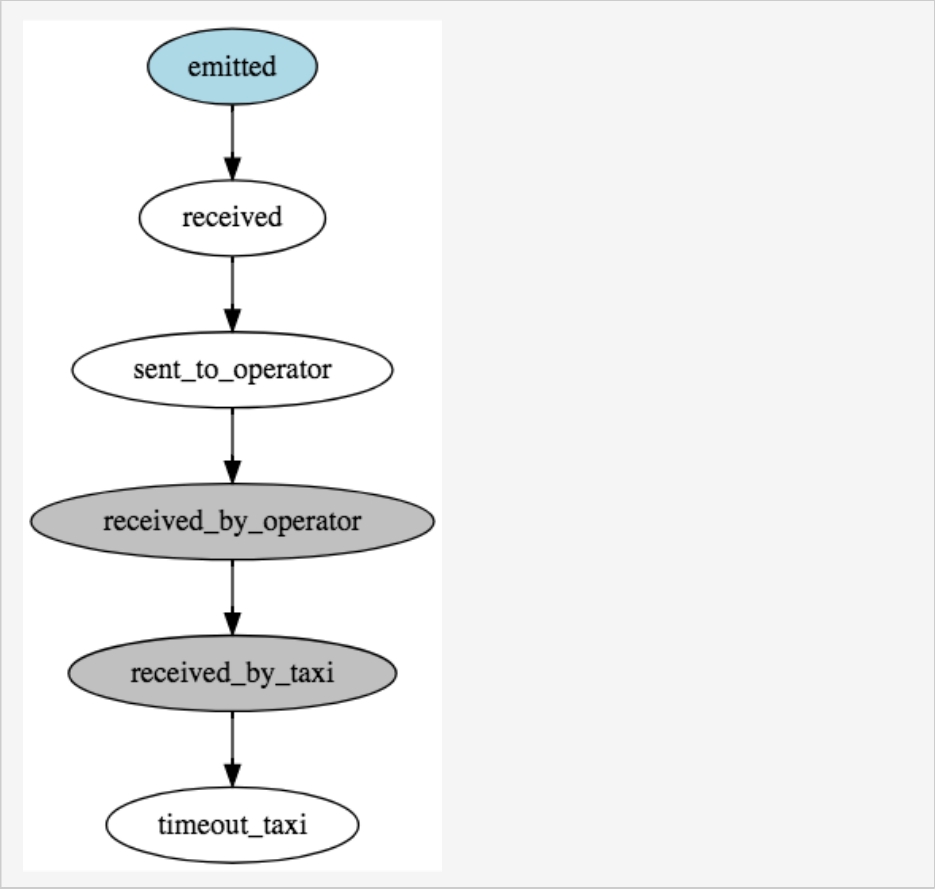
```
PUT /api/hails/{hailId} ** Send only one item at a time
{"data":[{"status":"received_by_taxi"}]}
```

- When the driver declined 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"}]}
```

4.4.3 Accepted by taxi after timeout

4.4.3.1 Graphical representation



4.4.3.2 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": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "coop",
        "customer_id": "coop_user1"
      }
    ]
  }
```

- 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"}]}
```

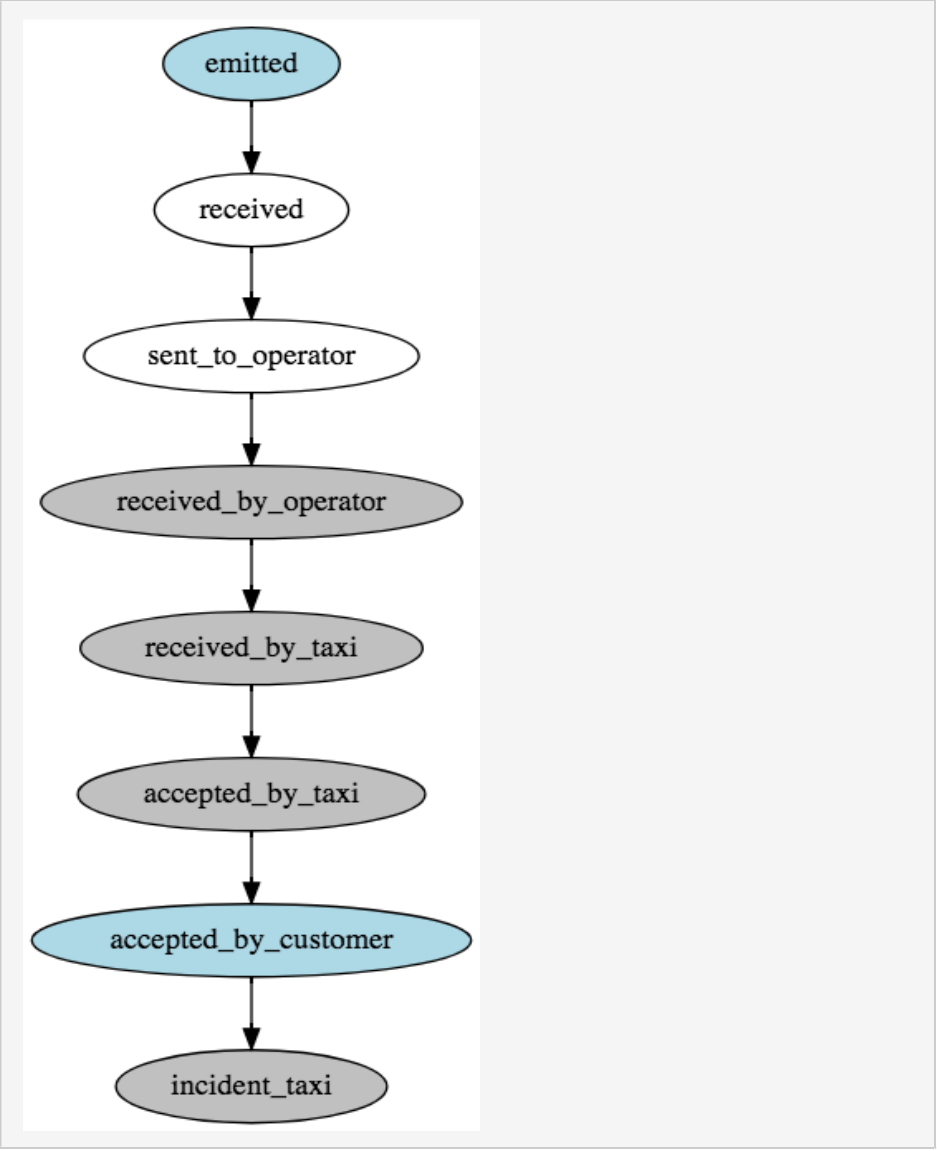
- When the driver accepts the hail after 30 seconds, the operator notifies the TXP.

```
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.

4.4.4 Canceled by taxi

4.4.4.1 Graphical representation



4.4.4.2 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": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "coop",
        "customer_id": "coop_user1"
      }
    ]
  }
```

- 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"}]}
```

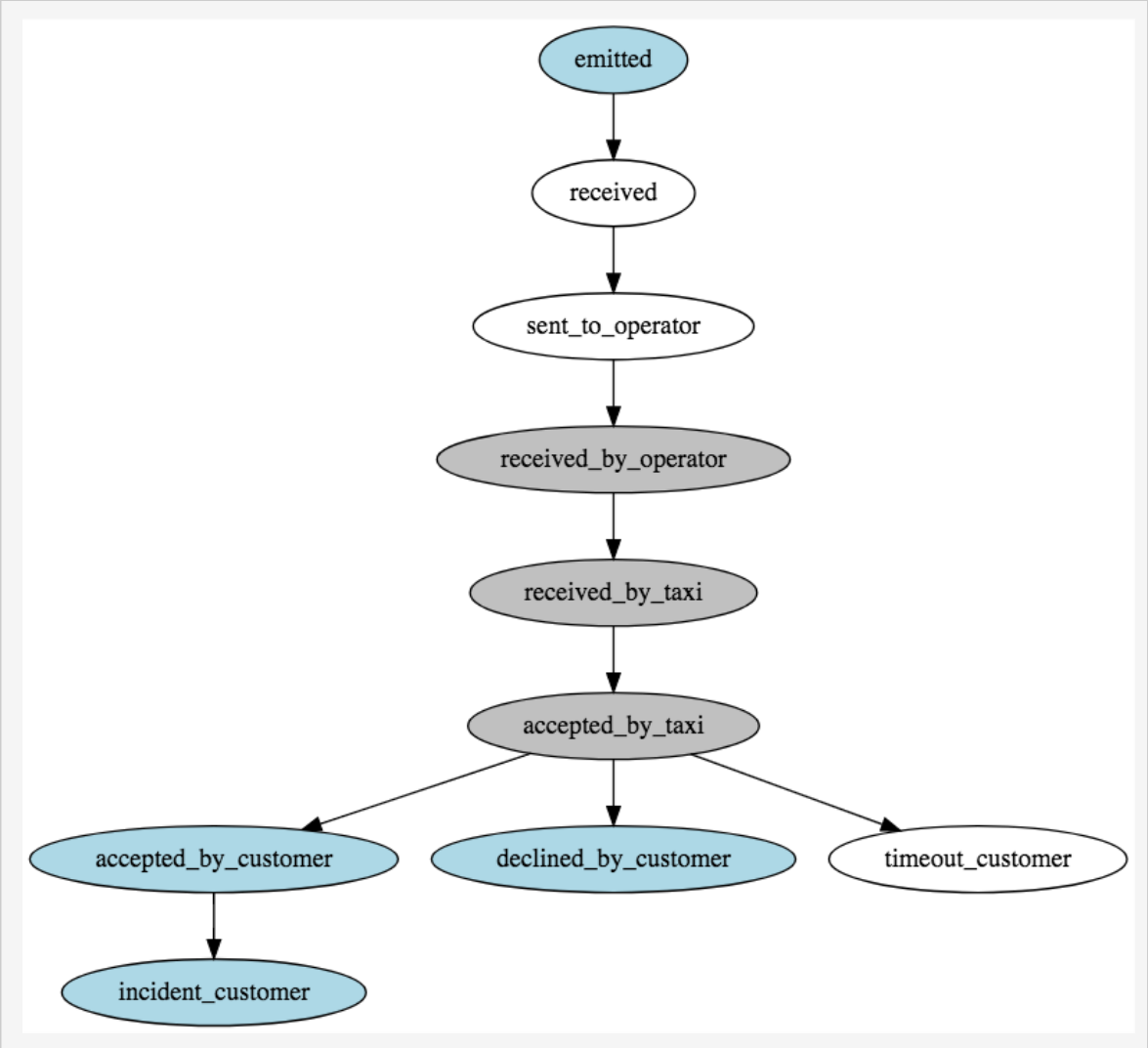
- When an incident occurs that prompted the driver to cancel the hail before having the client on board, the operator notifies the TXP and specifies the reason why the hail was cancel. See section 4.1 “Hail Status” from the Integration documentation for the list of possible reasons.

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

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

4.4.5 Canceled by client

4.4.5.1 Graphical representation



4.4.5.2 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",
        "operateur": "coop",
        "customer_id": "coop_user1"
      }
    ]
  }
```

- 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 reach 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 prompted the client to cancel the hail before being on board, the search engine notifies the TXP.

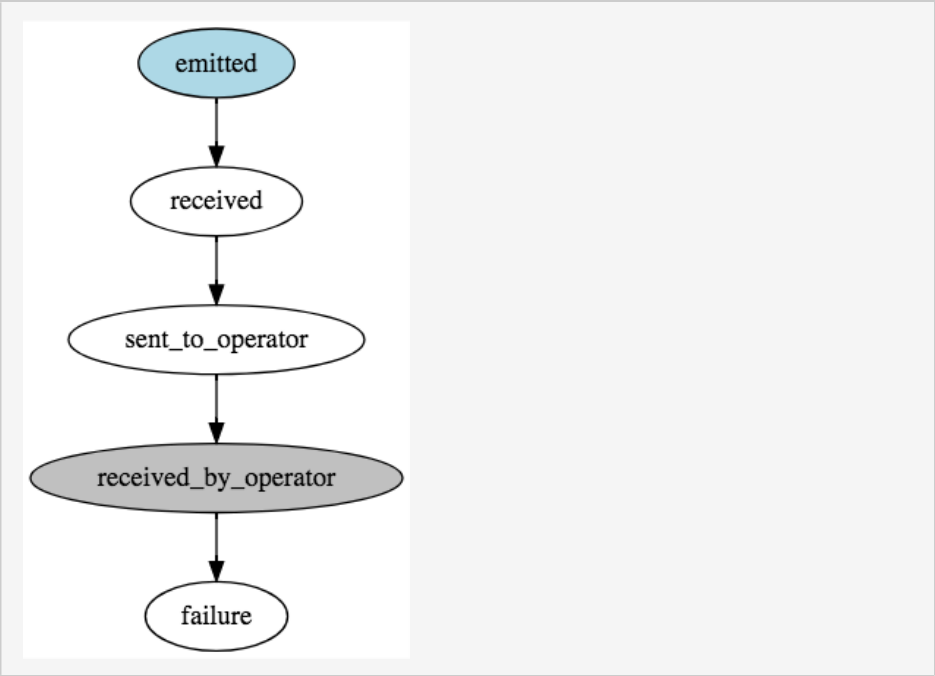
```
PUT /api/motor/hails/{hailId} ** Send only one item at a time
{"data":[{"status":"incident_customer"}]}
```

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

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

4.4.6 Failure example

4.4.6.1 Graphical representation



4.4.6.2 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": "{taxiId}",
        "customer_phone_number": "514 201-4454",
        "operateur": "coop",
        "customer_id": "coop_user1"
      }
    ]
  }
```

- 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}
```

4.4.7 Unexpected Exceptions

A state transition to the state failure may occurs 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 occurs 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. Other reference

5.1 Taxi zones (ZUPC)

- 102005 : A5 – Eastern part of the island of Montreal
- 102011 : A11 – Downtown/center Montreal
- 102012 : A12- West part of the island of MontrealA.5
- Montréal Pierre Elliott Trudeau Airport (Excluded)