com.dropoff.service.brawndo.client

This is the 3rd party dropoff ruby client for creating and viewing orders.

Table of Contents

Using the client

Copy the ruby client into your application. Let's assume you copied it into a folder called dropoff. Brawndo is accessed via a require_relative call.

```
require_relative 'dropoff/brawndo'
```

Configuration

You will then have to configure the brawndo instance via the constructor.

```
config = {}
config['public_key'] = 'b123bebbce97f1b06382095c3580d1be4417dbe376956ae9'
config['private_key'] = '87150f36c5de06fdf9bf775e8a7a1d0248de9af3d8930da0'
config['api_url'] = 'https://sandbox-brawndo.dropoff.com/v1'
config['host'] = 'sandbox-brawndo.dropoff.com'

brawndo = Brawndo.new(config)
```

- **api_url** - the url of the brawndo api. This field is required.
- **host** - the api host. This field is required.
- **public_key** - the public key of the user that will be using the client. This field is required.
- **private_key** - the private key of the user that will be using the client. This field is required.

Getting Your Client Information

If you want to know your client id and name you can access this information via the info call.

If you are an enterprise client user, then this call will return all of the accounts that you are allowed to manage with your current account.

```
info_data = brawndo.info()
```

A response will look like this:

```
{
  success: true
  timestamp: "2017-01-25T16:51:36Z",
  data: {
    client: {
      company_name: "EnterpriseCo Global",
      id: "1111111111110"
    },
    user: {
      first_name: "Algis",
      last_name: "Woss",
      id: "2222222222222"
    },
    managed_clients: {
```

```
level: 0,
company_name: "EnterpriseCo Global",
id: "1111111111110"
children : [
  {
    level: 1,
    company_name: "EnterpriseCo Europe",
    id: "1111111111112"
    children : [
      {
        level: 2,
        company_name: "EnterpriseCo Paris",
        id: "1111111111111"
        children : []
      },
      {
        level: 2,
        company_name: "EnterpriseCo London",
        id: "1111111111113"
        children : []
      },
      {
        level: 2,
        company_name: "EnterpriseCo Milan",
        id: "1111111111114"
        children : []
      }
    ]
  },
  {
    level: 1,
    company_name: "EnterpriseCo NA",
    id: "1111111111115"
    children : [
      {
        level: 2,
        company_name: "EnterpriseCo Chicago",
        id: "1111111111116"
        children : []
      },
      {
        level: 2,
        company_name: "EnterpriseCo New York",
        id: "1111111111117"
        children : []
```

```
            },
            {
              level: 2,
              company_name: "EnterpriseCo Los Angeles",
              id: "1111111111118"
              children : []
            }
          ]
        }
      ]
    }
  }
}
```

The main sections in data are user, client, and managed_clients.

The user info shows basic information about the Dropoff user that the used keys represent.

The client info shows basic information about the Dropoff Client that the user belongs to who's keys are being used.

The managed_clients info shows a hierarchical structure of all clients that can be managed by the user who's keys are being used.

Enterprise Managed Clients

In the above info example you see that keys for a user in an enterprise client are being used. It has clients that can be managed as it's descendants.

The hierarchy looks something like this:

```
    EnterpriseCo Global (1111111111110)
    ├─ EnterpriseCo Europe (1111111111112)
    |  ├─ EnterpriseCo Paris (1111111111111)
    |  ├─ EnterpriseCo London (1111111111113)
    |  └─ EnterpriseCo Milan (1111111111114)
    └─ EnterpriseCo NA (1111111111115)
       ├─ EnterpriseCo Chicago (1111111111116)
       ├─ EnterpriseCo New York (1111111111117)
       └─ EnterpriseCo Los Angeles (1111111111118)
```

Let's say I was using keys for a user in **EnterpriseCo Europe**, then the returned hierarchy would be:

```
    EnterpriseCo Europe (1111111111112)
    ├─ EnterpriseCo Paris (1111111111111)
    ├─ EnterpriseCo London (1111111111113)
    └─ EnterpriseCo Milan (1111111111114)
```

Note that You can no longer see the **EnterpriseCo Global** ancestor and anything descending and including **EnterpriseCo NA**.

So what does it mean to manage an enterprise client? This means that you can:

- Get estimates for that client.
- Place an order for that client.
- Cancel an order for that client.
- View existing orders placed for that client.
- Create, update, and delete tips for orders placed for that client.

All you have to do is specify the id of the client that you want to act on. So if wanted to place orders for **EnterpriseCo Paris** I would make sure to include that clients id: "1111111111111".

The following api documentation will show how to do this.

Getting Pricing Estimates

Before you place an order you will first want to estimate the distance, eta, and cost for the delivery. The client provides a **getEstimate** function for this operation.

```
estimate_params = {}
estimate_params['origin'] = '4729 Burnet Rd, Austin, TX 78756'        #required
estimate_params['destination'] = '2517 Thornton Rd, Austin, TX 78704' #required
estimate_params['utc_offset'] = Time.now.utc_offset                   #required
estimate_params['ready_timestamp'] = Time.now.to_i                    #optional
```

- **origin** - the origin (aka the pickup location) of the order. Required.
- **destination** - the destination (aka the delivery location) of the order. Required.
- **utc_offset** - the utc offset of the timezone where the order is taking place. Required.
- **ready_timestamp** - the unix timestamp (in seconds) representing when the order is ready to be picked up. If not set we assume immediate availability for pickup.
- **company_id** - if you are using brawndo as an enterprise client that manages other dropoff clients you can specify the managed client id who's estimate you want here. This is optional and only works for enterprise clients.

estimate*data = brawndo.order.estimate(estimate*params)

An example of a successful response will look like this:

```
{
    success => true,
    timestamp => '2015-03-05T14:51:14+00:00',
    service_type => 'standard',
    data => {
        ETA => '243.1',
        Distance => '0.62',
        From => '78701',
        To => '78701',
        asap => {
            Price => '19.00',
            ETA => '243.1',
            Distance => '0.62'
        },
        two_hr => {
            Price => '17.00',
            ETA => '243.1',
            Distance => '0.62'
        },
        four_hr => {
            Price => '15.00',
            ETA => '243.1',
            Distance => '0.62'
        },
        all_day => {
            Price => '19.00',
            ETA => '243.1',
            Distance => '0.62'
        }
    }
}
```

- **data** - contain the pricing information for the allowed delivery window based on the given ready time, so you will not always see every option.
- **Distance** - the distance from the origin to the destination.
- **ETA** - the estimated time (in seconds) it will take to go from the origin to the destination.
- **From** - the origin zip code. Only available if you have a zip to zip rate card configured.
- **To** - the destination zip code. Only available if you have a zip to zip rate card configured.
- **asap** - the pricing for an order that needs to delivered within an hour of the ready time.

- **two_hr** - the pricing for an order that needs to delivered within two hours of the ready time.
- **four_hr** - the pricing for an order that needs to delivered within four hours of the ready time.
- **all_day** - the pricing for an order that needs to delivered by end of business on a weekday.
- **service_type** - The service type for pricing, could be standard, holiday, or after_hr.

Placing an order

Given a successful estimate call, and a window that you like, then the order can be placed. An order requires origin information, destination information, and specifics about the order.

Origin and Destination data.

The origin and destination contain information regarding the addresses in the order.

```
origin = {
    address_line_1 => '117 San Jacinto Blvd',  # required
    company_name => 'Gus\'s Fried Chicken',    # required
    first_name => 'Napoleon',                  # required
    last_name => 'Bonner',                     # required
    phone => '5124744877',                     # required
    email => 'orders@gussfriedchicken.com',    # required
    city => 'Austin',                          # required
    state => 'TX',                             # required
    zip => '78701',                            # required
    lat => '30.263706',                        # required
    lng => '-97.741703',                       # required
    remarks => 'Be nice to napoleon'           # optional
}

destination = {
    address_line_1 => '800 Brazos Street',     # required
    address_line_2 => '250',                   # optional
    company_name => 'Dropoff Inc.',            # required
    first_name => 'Algis',                     # required
    last_name => 'Woss',                       # required
    phone => '8444376763',                     # required
    email => 'deliveries@dropoff.com',         # required
    city => 'Austin',                          # required
    state => 'TX',                             # required
    zip => '78701',                            # required
    lat => '30.269967',                        # required
    lng => '-97.740838'                        # required
}
```

- **address*line*1** - the street information for the origin or destination. Required.
- **address*line*2** - additional information for the address for the origin or destination (ie suite number). Optional.
- **company_name** - the name of the business for the origin or destination. Required.
- **first_name** - the first name of the contact at the origin or destination. Required.
- **last_name** - the last name of the contact at the origin or destination. Required.
- **phone_number** - the contact number at the origin or destination. Required.
- **email** - the email address for the origin or destination. Required.
- **city** - the city for the origin or destination. Required.
- **state** - the state for the origin or destination. Required.
- **zip** - the zip code for the origin or destination. Required.
- **lat** - the latitude for the origin or destination. Required.
- **lng** - the longitude for the origin or destination. Required.
- **remarks** - additional instructions for the origin or destination. Optional.

Order details data.

The details contain attributes about the order

```
details = {
    quantity => 1,                      # required
    weight => 5,                        # required
    eta => void(0),                     # required
    distance => nil,                    # required
    price => nil,                       # required
    ready_date => Time.now.utc_offset,  # required
    type => nil,                        # required
    reference_name => nil,              # optional
    reference_code => nil               # optional
}
```

- **quantity** - the number of packages in the order. Required.
- **weight** - the weight of the packages in the order. Required.
- **eta** - the eta from the origin to the destination. Should use the value retrieved in the getEstimate call. Required.
- **distance** - the distance from the origin to the destination. Should use the value retrieved in the getEstimate call. Required.
- **price** - the price for the order. Should use the value retrieved in the getEstimate call.. Required.
- **ready_date** - the unix timestamp (seconds) indicating when the order can be picked up. Can be up to 60 days into the future. Required.
- **type** - the order window. Can be asap, two*hr, four*hr, after*hr, or holiday depending on the ready*date.

Required.

- **reference_name** - a field for your internal referencing. Optional.
- **reference_code** - a field for your internal referencing. Optional.

Once this data is created, you can create the order.

```
order = {
  origin => origin,
  destination => destination,
  details => details
}

order_create_response = brawndo.order.create(order)
```

Note that if you want to create this order on behalf of a managed client as an enterprise client user you will need to specify the company_id.

```
order = {
  origin => origin,
  destination => destination,
  details => details,
  company_id => '1111111111111'
}

order_create_response = brawndo.order.create(order)
```

The data in the callback will contain the id of the new order as well as the url where you can track the order progress.

Cancelling an order

```
order_cancel_response = brawndo.order.cancel(order_id)
```

If you are trying to cancel an order for a manage client order as an enterprise client user, include the company_id in the argument parameters

```
order_cancel_data = {
  order_id => '61AE-Ozd7-L12',
  company_id => '1111111111111'
}

  order_cancel_response = brawndo.order.cancel(order_cancel_data)
```

- **order_id** - the id of the order to cancel.
- **company_id** - if you are using brawndo as an enterprise client that manages other dropoff clients you can specify the managed client id who you would like to cancel an order for. This is optional and only works for enterprise clients.

An order can be cancelled in these situations

- The order was placed less than **ten minutes** ago.
- The order ready time is more than **one hour** away.
- The order has not been picked up.
- The order has not been cancelled.

Getting a specific order

```
order_read_params = {
  order_id => 'zzzz-zzzz-zzz'
}

order_data = brawndo.order.read(order_read_params)
```

Example response

```
{
     data => {
         destination => {
             order_id => 'ac156e24a24484a382f66b8cadf6fa83',
             short_id => '06ex-r3zV-BMb',
             createdate => 1425653646,
             updatedate => 1425653646,
             order_status_code => 0,
             company_name => 'Dropoff Inc.',
             first_name => 'Algis',
             last_name => 'Woss',
             address_line_1 => '800 Brazos Street',
             address_line_2 => '250',
             city => 'Austin',
             state => 'TX',
             zip => '78701',
             phone_number => '8444376763',
             email_address => 'deliveries@dropoff.com',
             lng => -97.740838,
             lat => 30.269967
         },
```

```
      details => {
          order_id => 'ac156e24a24484a382f66b8cadf6fa83',
          short_id => '06ex-r3zV-BMb',
          createdate => 1425653646,
          customer_name => 'Algis Woss',
          type => 'ASAP',
          market => 'austin',
          timezone => 'America/Chicago',
          price => '15.00',
          signed => 'false',
          distance => '0.62',
          order_status_code => 0,
          wait_time => 0,
          order_status_name => 'Submitted',
          pickupETA => 'TBD',
          deliveryETA => '243.1',
          signature_exists => 'NO',
          quantity => 1,
          weight => 5,
          readyforpickupdate => 1425578400,
          updatedate => 1425653646
      },
      origin => {
          order_id => 'ac156e24a24484a382f66b8cadf6fa83',
          short_id => '06ex-r3zV-BMb',
          createdate => 1425653646,
          updatedate => 1425653646,
          order_status_code => 0,
          company_name => 'Gus's Fried Chicken',
          first_name => 'Napoleon',
          last_name => 'Bonner',
          address_line_1 => '117 San Jacinto Blvd',
          city => 'Austin',
          state => 'TX',
          zip => '78701',
          phone_number => '5124744877',
          email_address => 'orders@gussfriedchicken.com',
          lng => -97.741703,
          lat => 30.263706,
          market => 'austin',
          remarks => 'Be nice to napoleon'
      }
  },
  success => true,
  timestamp => '2015-03-09T18:42:15+00:00'
```

```
}
```

Getting a page of orders

Get the first page of orders

```
brawndo.order.read({})
```

Get a page of orders after the last_key from a previous response

```
order_read_params = {
  last_key => 'zhjklzvxchjladfshjklafdsknvjklfadjlhafdsjlkavdnjlvadslnjkdas'
}

order_data = brawndo.order.read(order_read_params)
```

Get the first page of orders as an enterprise client user for a managed client

```
order_read_params = {
  company_id => '1111111111111'
}

order_data = brawndo.order.read(order_read_params)
```

Get a page of orders after the last_key from a previous response as an enterprise client user for a managed client

```
order_read_params = {
  company_id => '1111111111111',
  last_key => 'zhjklzvxchjladfshjklafdsknvjklfadjlhafdsjlkavdnjlvadslnjkdas'
}

order_data = brawndo.order.read(order_read_params)
```

Example response

```
{
    data => [ ... ],
    count => 10,
    total => 248,
    last_key => 'zhjklzvxchjladfshjklafdsknvjklfadjlhafdsjlkavdnjlvadslnjkdas',
    success => true,
    timestamp => '2015-03-09T18:42:15+00:00'
}
```

Tips

You can create, delete, and read tips for individual orders. Please note that tips can only be created or deleted for orders that were delivered within the current billing period. Tips are paid out to our agents and will appear as an order adjustment charge on your invoice after the current billing period has expired. Tip amounts must not be zero or negative. You are limited to one tip per order.

Creating a tip

Tip creation requires two parameters, the order id **(order_id)** and the tip amount **(amount)**

```
tip_params = {
  order_id =>'61AE-Ozd7-L12',
  amount => 4.44
}

tip_create_response = brawndo.order.tip.create(tip_params)
```

Deleting a tip

Tip deletion only requires the order id **(order_id)**.

```
tip_delete_response = brawndo.order.tip.delete('61AE-Ozd7-L12')
```

If you are trying to delete a tip on a manage client order as an enterprise client user, include the company_id in the argument parameters

```
tip_delete_params = {
  order_id => '61AE-Ozd7-L12',
  company_id => '1111111111111'
}

tip_delete_response = brawndo.order.tip.delete(tip_delete_params)
```

Reading a tip

Tip reading only requires the order id **(order_id)**.

```
tip_read_response = brawndo.order.tip.read('61AE-Ozd7-L12')
```

If you are trying to read a tip on a manage client order as an enterprise client user, include the company_id in the argument parameters

```
tip_read_params = {
  order_id => '61AE-Ozd7-L12',
  company_id => '1111111111111'
}

tip_read_response = brawndo.order.tip.read(tip_read_params)
```

Example response:

```
{
  amount => "4.44"
  createdate => "2016-02-18T16:46:52+00:00"
  description => "Tip added by Dropoff(Algis Woss)"
  updatedate => "2016-02-18T16:46:52+00:00"
}
```

Webhooks

You may register a server route with Dropoff to receive real time updates related to your orders.

Your endpoint must handle a post, and should verify the X-Dropoff-Key with the client key given to you when registering the endpoint.

The body of the post should be signed using the HMAC-SHA-512 hashing algorithm combined with the client secret give to you when registering the endpoint.

The format of a post from Dropoff will be:

```
{
    count : 2,
    data : [ ]
}
```

- **count** contains the number of items in the data array.
- **data** is an array of events regarding orders and agents processing those orders.

Backoff algorithm

If your endpoint is unavailable Dropoff will try to resend the events in this manner:

- Retry 1 after 10 seconds
- Retry 2 after twenty seconds
- Retry 3 after thirty seconds
- Retry 4 after one minute
- Retry 5 after five minutes
- Retry 6 after ten minutes
- Retry 7 after fifteen minutes
- Retry 8 after twenty minutes
- Retry 9 after thirty minutes
- Retry 10 after forty five minutes
- All subsequent retries will be after one hour until 24 hours have passed

If all retries have failed then the cached events will be forever gone from this plane of existence.

Events

There are two types of events that your webhook will receive, order update events and agent location events.

All events follow this structure:

```
{
    event_name : <the name of the event ORDER_UPDATED or AGENT_LOCATION>
    data : { ... }
}
```

- **event_name** is either **ORDER_UPDATED** or **AGENT_LOCATION**
- **data** contains the event specific information

Order Update Event

This event will be triggered when the order is either:

- Accepted by an agent.
- Picked up by an agent.
- Delivered by an agent.
- Cancelled.

This is an example of an order update event

```
{
    event_name: 'ORDER_UPDATED',
    data: {
        order_status_code: 1000,
        company_id: '7df2b0bdb418157609c0d5766fb7fb12',
        timestamp: '2015-05-15T12:52:55+00:00',
        order_id: 'klAb-zwm8-mYz',
        agent_id: 'b7aa983243ccbfa43410888dd205c298'
    }
}
```

- **order*status*code** can be -1000 (cancelled), 1000 (accepted), 2000 (picked up), or 3000 (delivered)
- **company_id** is your company id.
- **timestamp** is a utc timestamp of when the order occured.
- **order_id** is the id of the order.
- **agent_id** is the id of the agent that is carrying out your order.

Agent Location Update Event

This event is triggered when the location of an agent that is carrying out your order has changed.

```
{
    event_name: 'AGENT_LOCATION',
    data: {
        agent_avatar: 'https://s3.amazonaws.com/com.dropoff.alpha.app.workerphoto/b7a
a983243ccbfa43410888dd205c298/worker_photo.png?AWSAccessKeyId=AKIAJN2ULWKTZXXEOQDA&Ex
pires=1431695270&Signature=AFKNQdT33lhlEddrGp0kINAR4uw%3D',
        latitude: 30.2640713,
        longitude: -97.7469492,
        order_id: 'klAb-zwm8-mYz',
        timestamp: '2015-05-15T12:52:50+00:00',
        agent_id: 'b7aa983243ccbfa43410888dd205c298'
    }
}
```

- **agent_avatar** is an image url you can use to show the agent. It expires in 15 minutes.
- **latitude** and **longitude** reflect the new coordinates of the agent.
- **timestamp** is a utc timestamp of when the order occured.
- **order_id** is the id of the order.
- **agent_id** is the id of the agent that is carrying out your order.

Managed Client Events

If you have registered a webhook with an enterprise client that can manager other clients, then the webhook will also receive all events for any managed clients.

So in our hierarchical example at the start, if a webhook was registered for **EnterpriseCo Global**, it would receive all events for:

- EnterpriseCo Global
- EnterpriseCo Europe
- EnterpriseCo Paris
- EnterpriseCo London
- EnterpriseCo Milan
- EnterpriseCo NA
- EnterpriseCo Chicago
- EnterpriseCo New York
- EnterpriseCo Los Angeles

Simulating an order

You can simulate an order via the brawndo api in order to test your webhooks.

The simulation will create an order, assign it to a simulation agent, and move the agent from pickup to the destination.

You can only run a simulation once every fifteen minutes.

```
simulation_response = brawndo.order.simulate('austin')
```