com.dropoff.service.brawndo.client

This is the 3rd party dropoff go client for creating and viewing orders and adding tips.

Table of Contents

Using the client

Configuration

To configure things you will have to create both a Brawndo Client and a Transport. The client contains the methods that you can call while the transport will contain the information required to properly sign the requests.

```
.....

import (
    "dropoff.com/brawndo"
)

var t brawndo.Transport
t.ApiURL = "https://sandbox-brawndo.dropoff.com/v1"
t.Host = "sandbox-brawndo.dropoff.com"
t.PublicKey = "91e9b320b18375927592759179d0b3d5415db4b80d4b553f46580a60119afc8"
t.SecretKey = "7f8fee62743d7febcda6482364826dfbeacbf4726f62d6fda26a3b906817482"

var b brawndo.Client
b.Transport = &t
```

- **ApiURL** - the url of the brawndo api. This field is required.
- **Host** - the api host. This field is required.
- **PublicKey** - the public key of the user that will be using the client. This field is required.
- **SecretKey** - the secret key of the user that will be using the client.

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.

```
res, err := brawndo.Info()
```

A successful response will be a struct in this format:

```
type GetInfoManagedClient struct {
  CompanyName     string
  Id              string
  Level           int
  Children        []*GetInfoManagedClient
}

type GetInfoClient struct {
  CompanyName     string
  Id              string
}

type GetInfoUser struct {
  FirstName       string
  LastName        string
  Id              string
}

type InfoResponseData struct {
  User            *GetInfoUser
  Client          *GetInfoClient
  ManagedClients  *GetInfoManagedClient
}

type InfoResponse struct {  // The response
  Data            *InfoResponseData
}
```

The main sections in Data are User, Client, and ManagedClients.

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 ManagedClients 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 could look 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.

```
var req brawndo.EstimateRequest

_, zone := time.Now().Zone()

req.Origin = "2517 Thornton Rd, Austin, TX 78704"
req.Destination = "800 Brazos St, Austin, TX 78704"
req.UTCOffset = zone
req.ReadyTimestamp = -1
req.CompanyId = ""
```

- **Origin** - the origin (aka the pickup location) of the order. Required.
- **Destination** - the destination (aka the delivery location) of the order. Required.
- **UTCOffset** - the utc offset of the timezone where the order is taking place. Value is in seconds. Required.
- **ReadyTimestamp** - the unix timestamp (in seconds) representing when the order is ready to be picked up. If not set we assume immediate availability for pickup.

* CompanyId - 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.

```
res, err := b.Estimate(origin, destination, o, ready)
```

This is the structure of a successful response:

```
type EstimateServiceType struct {
    ETA, Distance, Price string
}

type EstimateData struct {
    ETA, Distance, ServiceType string
    Asap, TwoHr, FourHr *EstimateServiceType
}

type EstimateResponse struct {  // This is the response
    Data      *EstimateData
    Success   bool
    Timestamp string
}
```

- **Success** - if true the request was processed successfully, if false, it could not be processed.
- **Timestamp** - the time at which the request completed.
- **Data** - contains pricing info
- **ServiceType** - the service type that the pricing reflects. Can be standard,holiday, or after_hr.
- **Asap** - contains pricing for asap delivery from the ready time.
- **TwoHr** - contains pricing for delivery within two hours of the ready time.
- **FourHr** - contains pricing for delivery within four hours of the ready time.
- **AllDay** - the pricing for an order that needs to delivered by end of business on a weekday..
- **ETA** - the estimated time (in seconds) it will take to go from the origin to the destination.
- **Distance** - the distance from the origin to the destination. In miles.
- **Price** - the price of the delivery for the time frame and service type.

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.

New Order Structure

In order to create a new order you would instantiate a CreateOrderRequest struct:

```
type CreateOrderRequest struct {
    Details     *CreateOrderDetails
    Origin      *CreateOrderAddress
    Destination *CreateOrderAddress
    CompanyId   string
}
```

- **Details** - contains data specific to the order
- **Origin** - contains data specific to the origin (pickup location) of the order
- **Destination** - contains data specific to the destination (dropoff location) of the order

* CompanyId - 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 create an order for. This is optional and only works for enterprise clients.

Origin and Destination data.

The Origin and Destination contain information regarding the addresses in the order. You would instantiate a CreateOrderAddress struct for each one

```
type CreateOrderAddress struct {
    CompanyName     string
    Email           string
    Phone           string
    FirstName       string
    LastName        string
    AddressLine1    string
    AddressLine2    string
    City            string
    State           string
    Zip             string
    Remarks         string
    Lat             float64
    Lng             float64
}
```

- **CompanyName** - the name of the business for the origin or destination. Required.
- **Email** - the email address for the origin or destination. Required.
- **Phone** - the contact number at the origin or destination. Required.
- **FirstName** - the first name of the contact at the origin or destination. Required.
- **LastName** - the last name of the contact at the origin or destination. Required.
- **AddressLine1** - the street information for the origin or destination. Required.
- **AddressLine2** - additional information for the address for the origin or destination (ie suite number). Optional.
- **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.
- **Remarks** - additional instructions for the origin or destination. Optional.
- **Lat** - the latitude for the origin or destination. Required.

* Lng - the longitude for the origin or destination. Required.

Order details data.

The Details contain information about the order

```
type CreateOrderDetails struct {
    Quantity      int64
    Weight        int64
    ETA           string
    Distance      string
    Price         string
    ReadyDate     int64
    Type          string
    ReferenceCode string
    ReferenceName string
}
```

- **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.

- **ReadyDate** - 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 depending on the ready_date. Required.
- **ReferenceName** - a field for your internal referencing. Optional.

* ReferenceCode - a field for your internal referencing. Optional.

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

```
var cor brawndo.CreateOrderRequest
var cor_det brawndo.CreateOrderDetails
var cor_o, cor_d brawndo.CreateOrderAddress

cor_det.Quantity = 1
cor_det.Weight = 5
cor_det.ETA = "448.5"
cor_det.Distance = "0.64"
cor_det.Price = "13.99"
cor_det.ReadyDate = time.Now().Unix()
cor_det.Type = "two_hr"
cor_det.ReferenceCode = "reference code 0001"
cor_det.ReferenceName = "reference name"

cor_o.CompanyName = "Dropoff GO Origin"
cor_o.Email = "noreply+origin@dropoff.com"
cor_o.Phone = "5124744877"
cor_o.FirstName = "Napoleon"
cor_o.LastName = "Bonner"
cor_o.AddressLine1 = "117 San Jacinto Blvd"
//cor_o.AddressLine2 = ""
cor_o.City = "Austin"
cor_o.State = "TX"
cor_o.Zip = "78701"
cor_o.Lat = 30.263706
cor_o.Lng = -97.741703
cor_o.Remarks = "Be nice to napoleon"

cor_d.CompanyName = "Dropoff GO Destination"
cor_d.Email = "noreply+destination@dropoff.com"
cor_d.Phone = "5555554444"
cor_d.FirstName = "Del"
```

```
cor_d.LastName = "Fitzgitibit"
cor_d.AddressLine1 = "800 Brazos Street"
cor_d.AddressLine2 = "250"
cor_d.City = "Austin"
cor_d.State = "TX"
cor_d.Zip = "78701"
cor_d.Lat = 30.269967
cor_d.Lng = -97.740838
//cor_d.Remarks = "Optional remarks"

cor.Details = &cor_det
cor.Destination = &cor_d
cor.Origin = &cor_o

res,err := b.CreateOrder(&cor)
```

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

```
type CreateOrderData struct {
    OrderId     string
    ShortId     string
    URL         string
}

type CreateOrderResponse struct {  // this is returned
    Message     string
    Timestamp   string
    Success     bool
    Data        *CreateOrderData
}
```

Cancelling an order

```
var req brawndo.OrderRequest

req.OrderId = "abcdef1234567890fedcba"
req.CompanyId = ""

res, err := b.CancelOrder(&req)
```

- **OrderId** - the id of the order to cancel.

* CompanyId - 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

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

Getting a specific order

```
var req brawndo.OrderRequest

req.OrderId = order_id
req.CompanyId = company_id

res, err := b.GetOrder(&req)
```

- **OrderId** - the id of the order to view.

* CompanyId - 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 get an order for. This is optional and only works for enterprise clients.

This will return a GetOrderResponse struct

```
type GetOrderResponse struct {
    Data      *GetOrderData
    Success   bool
    Timestamp string
}
```

- **Data** - contains specifics about the order
- **Success** - true if the order was retrieved, false otherwise.

* Timestamp - the time that the opration completed

The struct for GetOrderData looks like this:

```
type GetOrderData struct {
    Details     *GetOrderDetails
    Origin      *GetOrderAddress
    Destination *GetOrderAddress
}
```

- **Details** - contains data specific to the order
- **Origin** - contains data specific to the origin (pickup location) of the order

* Destination - contains data specific to the destination (dropoff location) of the order

The struct for GetOrderDetails looks like this:

```
type GetOrderDetails struct {
    OrderId             string
    CustomerName        string
    Price               string
    Distance            string
    Quantity            int64
    Weight              int64
    Market              string
    ServiceType         string
    TimeFrame           string
    Timezone            string
    UTCOffsetMinutes    int64
    CreateDate          int64
    UpdateDate          int64
    ReadyForPickupDate  int64
    OrderStatusCode     int64
    OrderStatusName     string
    ReferenceCode       string
    ReferenceName       string
}
```

- **OrderId** - the id of the order
- **CustomerName** - the name of the client that placed the order.
- **Price** - the price for the order.
- **Distance** - the distance from the origin to the destination.
- **Quantity** - the number of packages in the order.
- **Weight** - the weight of the packages in the order.
- **Market** - the market that the order was in.
- **ServiceType** - the service type of the order, can be standard, holiday, or after_hr.
- **TimeFrame** - the order window. Can be asap, two*hr, four*hr depending on the ready_date.
- **TimeZone** - the timezone of the order.
- **UTCOffsetMinutes** - the UTC offset of the timezone the order was in.
- **CreateDate** - the time the order was created. unix timestamp.
- **UpdateDate** - the time the order was updated. unix timestamp.
- **ReadyForPickupDate** - the time the order was ready to be picked up. unix timestamp.
- **OrderStatusCode** - the current status code for the order.
    - -1000 is cancelled.
    - 0 is submitted.
    - 1000 is assigned.
    - 2000 is pickedup.

    - 3000 is delivered.

- **OrderStatusName** - a string description of the status.
- **ReferenceName** - a field for your internal referencing.

* ReferenceCode - a field for your internal referencing.

The struct for GetOrderAddress looks like this:

```
type GetOrderAddress struct {
    CompanyName   string
    FirstName     string
    LastName      string
    AddressLine1  string
    AddressLine2  string
    City          string
    State         string
    Zip           string
    Lng           float64
    Lat           float64
    Email         string
    Phone         string
    CreateDate    int64
    UpdateDate    int64
}
```

- **CompanyName** - the name of the business for the address.
- **FirstName** - the first name of the contact at the address.
- **LastName** - the last name of the contact at the address.
- **AddressLine1** - the street information for the address.
- **AddressLine2** - additional street information for the address.
- **City** - the city for the address.
- **State** - the state for the address.
- **Zip** - the zip code for the address.
- **Lat** - the latitude for the address.
- **Lng** - the longitude for the address.
- **Email** - the email address for the address.
- **Phone** - the contact number at the address.
- **CreateDate** - the unix timestamp of creation.
- **UpdateDate** - the unix timestamp of the last update.

* Remarks - additional instructions for the address.

Getting a page of orders

```
//  Get an the first order page for the client your keys represent
  var req brawndo.OrderRequest
  res, err := b.GetOrderPage(&req)

//  Get an the order page after the given key for the client your keys represent
  var req brawndo.OrderRequest
  req.LastKey = "1234567890abcdeffedcbakdjsaynzcvjkdsauiadfsjkfasdkjfsadkjadfshk"
  res, err := b.GetOrderPage(&req)

//  Get an the first order page for a managed client if you are an enterprise client
  var req brawndo.OrderRequest
  req.CompanyId = "1234567890abcdeffedcba"
  res, err := b.GetOrderPage(&req)

//  Get an the order page after the given key for a managed client if you are an ente
rprise client
  var req brawndo.OrderRequest
  req.CompanyId = "1234567890abcdeffedcba"
  req.LastKey = "1234567890abcdeffedcbakdjsaynzcvjkdsauiadfsjkfasdkjfsadkjadfshk"
  res, err := b.GetOrderPage(&req)
```

- **LastKey** - the key that marks the next page of orders. optional.

* CompanyId - 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 get a page of orders for. This is optional and only works for enterprise clients.

This will return a GetOrdersResponse struct when successful

```
type GetOrdersResponse struct {
  Total       int64
  Count       int64
  LastKey     string
  Data        []*GetOrderData
  Success     bool
  Timestamp   string
}
```

Use **LastKey** to get the subsequent page of orders.

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 specifying an order id and an amount.

```
var req brawndo.OrderTipRequest
req.OrderId = "12345abcdef67890fedcba"
req.Amount = "7.50"
req.CompanyId = ""

res, err := b.CreateOrderTip(&req)
```

- **OrderId** - the order id you want to add the tip to.
- **Amount** - the amount of the tip.

* CompanyId - if you are using brawndo as an enterprise client that manages other dropoff clients you can specify the managed client id who who has an order you want to add a tip to. This is optional and only works for enterprise

clients.

Response Struct:

```
type TipResponseData struct {
  Amount        string
  Description   string
  CreateDate    string
  UpdateDate    string
}

type TipResponse struct {
  Message       string
  Timestamp     string
  Success       bool
  Tip           *TipResponseData
}
```

Deleting a tip

Tip deletion requires specifying an order id.

```
var req brawndo.OrderTipRequest
req.OrderId = "12345abcdef67890fedcba"
req.CompanyId = ""

res, err := b.DeleteOrderTip(&req)
```

Response Struct:

```
type DeleteTipResponse struct {
  Message     string
  Timestamp   string
  Success     bool
}
```

- **OrderId** - the order id you want to delete the tip from.

* CompanyId - if you are using brawndo as an enterprise

client that manages other dropoff clients you can specify the managed client id who who has an order you want to remove a tip from. This is optional and only works for enterprise clients.

Reading a tip

Tip reading requires specifying an order id.

```
var req brawndo.OrderTipRequest
req.OrderId = "12345abcdef67890fedcba"
req.CompanyId = ""

res, err := b.GetOrderTip(&req)
```

- **OrderId** - the order id who's tip you want to see.

* CompanyId - if you are using brawndo as an enterprise client that manages other dropoff clients you can specify the managed client id who who has an order who's tip you want to see. This is optional and only works for enterprise clients.

Response Struct:

```
type GetTipResponse struct {
  Amount        string
  Description   string
  CreateDate    string
  UpdateDate    string
}
```

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.

```
res, err := b.SimulateOrder(market)
```

The struct response is:

```
type SimulateOrderResponse struct {
    OrderId             string
    OrderDetailsUrl     string
    Timestamp           string
    Success             bool
}
```

- **OrderId** - the id of the simulated order.
- **OrderDetailsUrl** - the url of the order details page.
- **Timestamp** - the timestamp that the simulation request was completed.

* Success - true if the simulation was started.