



Brawndo Javascript Client

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

- For PHP documentation go [HERE](#)
- For GO documentation go [HERE](#)
- For Ruby documentation go [HERE](#)
- For C# documentation go [HERE](#)

Table of Contents

- [Client Info](#)
 - [Configuration](#)
 - [Getting Your Account Info](#)
 - [Enterprise Managed Clients](#)
 - [Order Properties](#)
 - [Getting Pricing Estimates](#)
 - [Placing an Order](#)
 - [Cancelling an Order](#)
 - [Getting a Specific Order](#)
 - [Getting a Page of Order](#)
- [Signature Image URL](#)
- [Tips](#)
 - [Creating](#)
 - [Deleting](#)
 - [Reading](#)
- [Webhook Info](#)
 - [Webhook Backoff Algorithm](#)
 - [Webhook Events](#)

- [Managed Client Events](#)
- [Order Simulation](#)

Using the client

This code should be identical in a browser (assuming you have browserified brawndo.js) or in nodejs. Brawndo is access via a require call.

```
var brawndo = require('brawndo');
```

Configuration

You will then have to configure the brawndo instance with the configure function.

```
var configure_params = {};  
configure_params.api_url = 'https://sandbox-brawndo.dropoff.com/v1';  
configure_params.host = 'sandbox-brawndo.dropoff.com';  
configure_params.public_key = 'b123bebbce97f1b06382095c3580d1be4417dbe376956ae9';  
  
configure_params.private_key = '87150f36c5de06fdf9bf775e8a7a1d0248de9af3d8930da0';  
  
// or .....  
  
configure_params.hasher_url = 'https://myserver.com/sign';
```

- **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. Can be substituted with **hasher_url**
- **hasher_url** - the endpoint that can be called to generate the Authorization header on the server side in case you don't want the private key to be stored on your client.

Implementing a hasher_url endpoint

It's pretty straightforward if you are storing the user information on the server side. This example will assume that the public and private keys are part of the session.

```

var brawndo = require('brawndo');

// ..... express setup .....

app.post('/do_sign', function(req, res) {
  if (req.session && req.session.public_key && req.session.private_key) {
    brawndo.configure({
      api_url : 'https://qa-brawndo.dropoff.com/v1',
      public_key : req.session.public_key,
      private_key : req.session.private_key
    });
    var signature = brawndo.createHash(params);
    res.status(200).json({ signature : signature });
  } else {
    res.status(401).end();
  }
});

```

The client will handle the rest.

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.

```

brawndo.info(function(error, data) {
  if (error) {
    .....
  } else {
    .....
  }
});

```

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.

Order Properties

Depending on your client, you may have the option to add properties to your order. In order to determine whether or not your client has properties, you can make a call the **properties** function. It will return all properties that can be applied to your orders during creation.

```
var properties_params = {};  
properties.params.company_id = ""; // optional  
  
brawndo.order.properties(properties_params, function(error, props) {})
```

If you include a **company_id** you will retrieve that company's properties only if your account credentials are managing that account.

An example of a successful response will look like this:

```

{
  "data": [
    {
      "id": 1,
      "label": "Leave at Door",
      "description": "If recipient is not at home or at office, leave order at the door.",
      "price_adjustment": 0,
      "conflicts": [
        2
      ],
      "requires": []
    },
    {
      "id": 2,
      "label": "Signature Required",
      "description": "Signature is required for this order.",
      "price_adjustment": 0,
      "conflicts": [
        1
      ],
      "requires": []
    },
    {
      "id": 3,
      "label": "Legal Filing",
      "description": "This order is a legal filing at the court house. Please read order remarks carefully.",
      "price_adjustment": 5.50,
      "conflicts": [],
      "requires": [
        2
      ]
    }
  ],
  "count": 3,
  "total": 3,
  "success": true
}

```

- **id** - the id of the property, you will use this value if you want to add the property to an order you are creating
- **label** - a simple description of the property.
- **description** - more details about the property.

- **price_adjustment** - a number that describes any additional charges that the property will require.
- **conflicts** - an array of other property ids that cannot be included in an order when this property is set. In the above response you cannot set both "Leave at Door" and "Signature Required".
- **requires** - an array of other property ids that must be included in an order when this property is set. In the above response, when "Legal Filing" is set on an order, then "Signature Required" should be set as well.

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 estimate_params = {};
estimate_params.origin = '117 San Jacinto Blvd, Austin, TX 78701, United States'; //
  required
estimate_params.destination = '800 Brazos Street, Austin, TX 78701, United States'; /
/ required
estimate_params.utc_offset = '-06:00'; // required
estimate_params.ready_timestamp = 1425578400; // 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.

```
brawndo.order.estimate(estimateParameters, function(error, estimate_data) { });
```

An example of a successful response will look like this:

```

{
  success : true,
  timestamp: '2015-03-05T14:51:14+00:00',
  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: '21.00',
      ETA: '243.1',
      Distance: '0.62'
    },
    service_type: 'standard'
  }
}

```

- **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 be delivered within an hour of the ready time.
- **two_hr** - the pricing for an order that needs to be delivered within two hours of the ready time.
- **four_hr** - the pricing for an order that needs to be delivered within four hours of the ready time.
- **all_day** - the pricing for an order that needs to be 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.

```
var 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
};

var 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

```
var details = {
  quantity : 1,           // required
  weight : 5,             // required
  eta : void(0),          // required
  distance : void(0),     // required
  price : void(0),        // required
  ready_date : 1425578400, // required
  type : void(0),         // required
  reference_name : void(0), // optional
  reference_code : void(0) // 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*, *twohr*, *fourhr*, *afterhr*, or *holiday* depending on the *readydate*. Required.
- **reference_name** - a field for your internal referencing. Optional.
- **reference_code** - a field for your internal referencing. Optional.

Order properties data.

The properties section is an array of [property ids](#) to add to the order

```
var properties = [ 2, 3 ];
```

This is an optional piece of data.

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

```
brawndo.order.create({
  origin : origin,
  destination : destination,
  details : details,
  properties : properties
}, function(error, data) {
});
```

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`.

```
brawndo.order.create({
  origin : origin,
  destination : destination,
  details : details,
  properties : properties,
  company_id : '1111111111111'
}, function(error, data) {
});
```

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

```
brawndo.order.cancel(order_id, function(error, data) {});
```

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

```
brawndo.order.cancel({
  order_id : 'zzzz-zzzz-zzz',
  company_id : '1111111111111'
},
function(error, 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

```
brawndo.order.read({order_id : 'zzzz-zzzz-zzz'}, function(error, data) {
});
```

Example response

```
{
  data: {
    destination: {
      order_id: 'ac156e24a24484a382f66b8cadf6fa83',
      short_id: 'zzzz-zzzz-zzz',
      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: 'zzzz-zzzz-zzz',
    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: 'zzzz-zzzz-zzz',
    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'
    },
    properties : [
        {
            "id": 2,
            "label": "Signature Required",
            "description": "Signature is required for this order.",
            "price_adjustment": 0
        },
        {
            "id": 3,
            "label": "Legal Filing",
            "description": "This order is a legal filing at the court house.
Please read order remarks carefully.",
            "price_adjustment": 5.50
        }
    ]
},
success: true,
timestamp: '2015-03-09T18:42:15+00:00'
}

```

Getting a page of orders

Get the first page of orders

```

brawndo.order.read(function(error, data) {
});

```

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

```

brawndo.order.read({last_key : 'zhjklzvchjladfshjklafdsknvjklfadjlhafdsjlkavdnjlvads
lnjkdas'}, function(error, data) {
});

```

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

```

brawndo.order.read({company_id : '111111111111'}, function(error, data) {
});

```

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


```
brawndo.order.read({
  last_key : 'zhjklzvxchjladfshjklafdsknvjklfadjlhafdsjlkavdnjlvadslnjkdas'
  company_id : '1111111111111'
},
function(error, data) {
});
```

Example response

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

Signature Image URL

Some orders will contain signatures. If you want to get a url to an image of the signature you can call the **signature** function. Note that the signature may not always exist, for example when the delivery was left at the door of the destination.

```
var signature_params = {};
signature_params.order_id = "Rr0W-e1OL-Lr0"; // required
signature_params.company_id = ""; //optional
brawndo.order.signature(signature_params, function(error, data) {
});
```

Example response

```
{
  "success": true,
  "url": "https://s3.amazonaws.com/....."
}
```

The signature url is configured with an expiration time of 5 minutes after the request for the resource was made

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**).

```
brawndo.order.tip.create({ order_id : '61AE-Ozd7-L12', amount : 4.44 }, function(error, data) {
});
```

Deleting a tip

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

```
brawndo.order.tip.delete('61AE-Ozd7-L12', function(error, data) {
});
```

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

```
brawndo.order.tip.delete({
  order_id : '61AE-Ozd7-L12',
  company_id : '11111111111111'
},
function(error, data) {
});
```

Reading a tip

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

```
brawndo.order.tip.read('61AE-Ozd7-L12', function(error, data) {
});
```

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

```
brawndo.order.tip.read({
  order_id : '61AE-Ozd7-L12',
  company_id : '111111111111'
},
function(error, data) {
});
```

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'
  }
}
```

- **orderstatuscode** 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 occurred.
- **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/b7aa983243ccbfa43410888dd205c298/worker_photo.png?AWSAccessKeyId=AKIAJN2ULWKTZXXEQDA&Expires=1431695270&Signature=AFKNQdT33lh1EddrGp0kINAR4uw%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 occurred.
- **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.

```
brawndo.order.simulate({ market : 'austin' }, function(error, result) {});
```