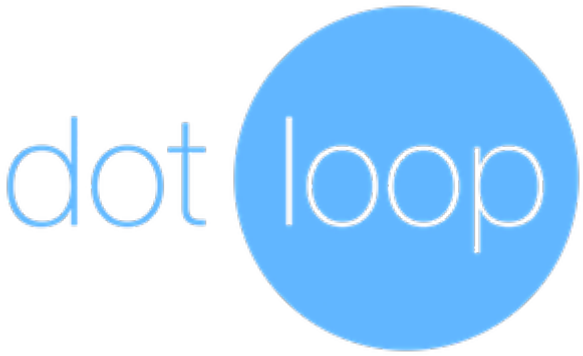


Dotloop Platform - Developer Guide - Public API Version 2

Table of Contents

Download PDF Version

1. Overview

1.1. What's New

Loop-It Facade API

We now provide a simple 'Facade' API which allows client applications to create a loop and populate data into it via a single request, which includes inserting property information, adding loop participants into the contacts directory, and creating a loop from loop templates etc.

New Authentication Scheme

We introduce a new authentication scheme (OAuth2) which gets client applications access to user accounts upon user's consent.

New Request / Response Schemas

All APIs and their corresponding request/response schemas have been refreshed/updated - we continue to support existing integrations using dotloop's external API.

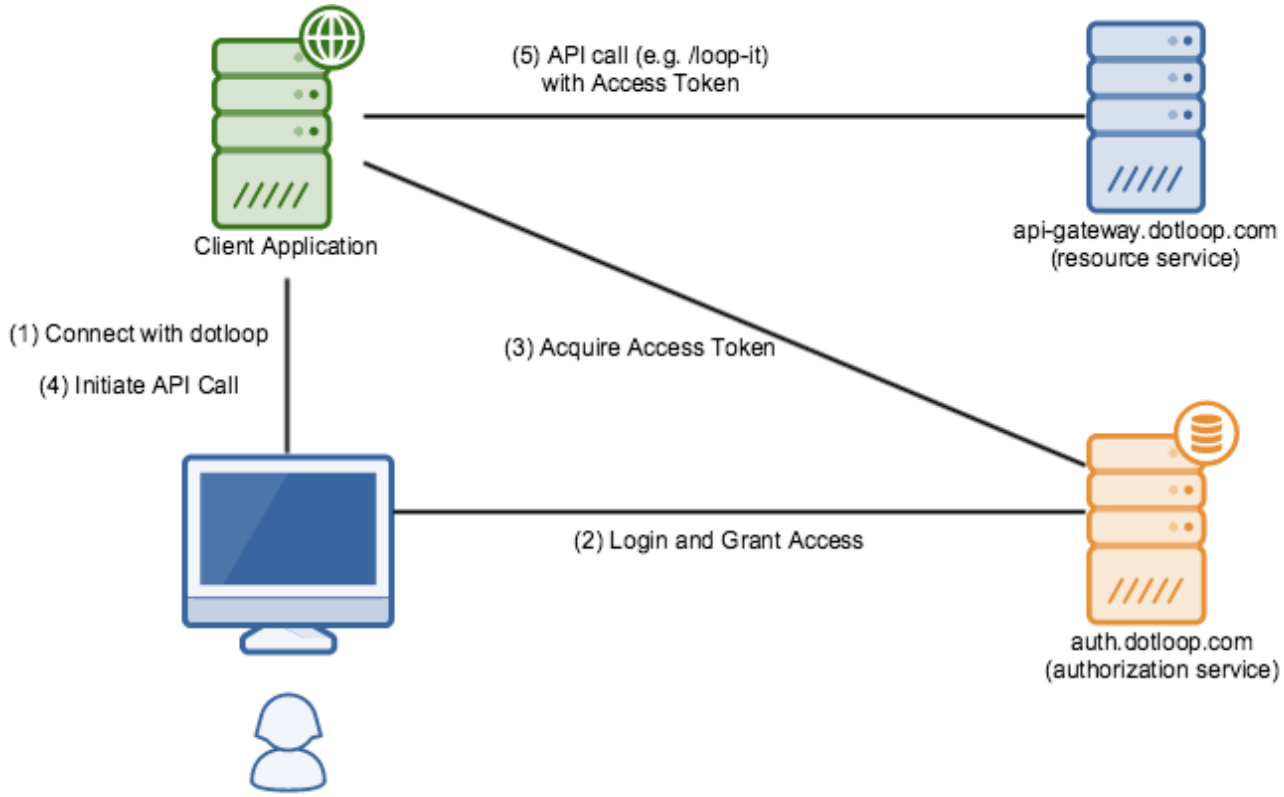
Read/Write Access

We introduce a new set of APIs in addition to `GET` APIs already available in external API, so clients can now for the first time *create* and *update* new resources (e.g. create/update loops or profiles via `POST` and `PATCH`).

1.2. Authentication

Dotloop's Public API Version 2 makes use of the OAuth 2.0 protocol for authentication and authorization and initially support scenarios for web server applications only (3-legged-OAuth).

By using this protocol, we allow dotloop users to control data access by third-party applications and provide users the ability to revoke previously granted access at a later time.



1.2.1. Client Registration

To register your application to integrate with this API, please request access at http://info.dotloop.com/developers. Upon registration, we will issue you a client id and client secret which are prerequisites in order to use the API.

1.2.2. Obtaining Access Token

In order to obtain an access token to access any resources on behalf of a dotloop user, client applications need to obtain an access token for each user provided the user gives his consent. Access tokens are short-living and expire usually after *12 hours*, hence need to be refreshed once expired. For more information on the Oauth2 protocol, e.g. error codes please see RFC6749.

Step 1 - Obtain an Authorization Code

The first step towards acquisition of access and refresh tokens is to obtain an *Authorization Code*. The code will be issued after the user approved access to the client application to his account. To prompt the user to give his approval, redirect the user to (usually done in a popup window):

```
https://auth.dotloop.com/oauth/authorize?response_type=code&client_id=<client_id>&redi
rect_uri=<redirect_url>[&state=<state>&redirect_on_deny=(true|false)]
```

Parameters

Name	Type	Description
response_type	string	[required] only `code` is supported today

Name	Type	Description
client_id	string	[required] client id (UUID issued when registring client application)
redirect_uri	string	[required] URL the user agent gets redirected to with authorization code after user consent
state	string	[optional] random string to protect against CSRF
redirect_on_deny	boolean	[optional] defines whether the action behind deny button redirects to `redirect_uri` or just closes the browser window; [true | false] (default: false)

Once the user approved the request of your client to access his account, we'll issue a user agent redirect (`302` with `Location` header) back to the URL provided (`redirect_uri` param) with the authorization code added in the query, ie. `<redirect_url>?code=<code>`. Please not that the `state` param is recommended and should be used to protect your site against CSRF.

Step 2 - Acquire Access/Refresh Token

With the `code` received in step 1, the client application can obtain an access token by making a request against the `/token` endpoint.

The `/token` endpoint requires an authentication header (HTTP Basic Authentication) sent by the client application.

Request

```
POST
https://auth.dotloop.com/oauth/token?grant_type=authorization_code&code=<code>&redirec
t_uri=<redirect_url>&state=<state>
```

Header

```
Authorization: Basic <encode_base64(ClientID:ClientSecret)>
```

- Example
  - ClientId: `69bcf590-71b7-41a4-a039-a1d290edca11`
  - ClientSecret: `3415e381-bdc4-49b7-bde2-69b3c5cd6447`
  - Resulting Header:

```
Authorization: Basic
NjliY2Y1OTAtNzFiNy00MWE0LWEwMzktYTFkMjkwZWRjYTExOjM0MTVlMzgxLWJkYzQtNDliNy1iZGUy
LTY5YjNjNWNkNjQ0Nw==
```

Response

```
Status: 200 OK
```

```
{
  "access_token": "0b043f2f-2abe-4c9d-844a-3eb008dcba67",
  "token_type": "Bearer",
  "refresh_token": "19bfda68-ca62-480c-9c62-2ba408458fc7",
  "expires_in": 43145,
  "scope": "profile:*, loop:*"
}
```

1.2.3. Refreshing Access Token after expiration

Access tokens have a short lifetime and need to be refreshed every 12 hours. The client application can either pro-actively refresh tokens before they expire and lazily refresh them upon receiving an authentication error (`401 Unauthenticated`) when accessing the API.

```
POST
https://auth.dotloop.com/oauth/token?grant_type=refresh_token&refresh_token=<refresh_t
oken>
```

Header

```
Authorization: Basic <encode_base64(ClientID:ClientSecret)>
```

Response

```
Status: 200 OK
```

```
{
  "access_token": "86609772-aa95-4071-ad7f-25ad2d0be295",
  "token_type": "Bearer",
  "refresh_token": "19bfda68-ca62-480c-9c62-2ba408458fc7",
  "expires_in": 43199,
  "scope": "account:read, profile:*, loop:*, contact:*, template:read"
}
```

When refreshing access tokens, any previously issued access token becomes invalid. If you manage tokens in a clustered environment, make sure to share/use and refresh the token once across your cluster instances to avoid race conditions during the token update when triggered from multiple instances concurrently.

1.2.4. Access Revocation

A client or the actual user of the client may decide to disconnect from dotloop and revoke previously given permission to his dotloop account. To revoke access, the client application should call the following endpoint which will invalidate access and refresh token for future use.

```
POST https://auth.dotloop.com/oauth/token/revoke?token=<access_token>
```

1.3. Endpoint

All APIs listed below are available under a common base path:

https://api-gateway.dotloop.com/public/v2/

All paths below are relative to this url, e.g. `https://api-gateway.dotloop.com/public/v2/loop-it`

1.4. Content Type

This API is a JSON API, so request and response payload are expected to be of content type `application/json` if not marked otherwise.

1.5. Authorization Header

Each API request requires a valid Access Token to be presented in an `Authorization` header, e.g.

```
Authorization: Bearer 0b043f2f-2abe-4c9d-844a-3eb008dcba67
```

2. Loop-It™

The *Loop-It™* API makes it easy to create a new *Loop* and and populate various details into the loop, e.g. setup loop participant's contact data into the contacts directory, pulls listing property data, authenticates the caller if an NRDS Id or MLS Agent Id is available to get access to form templates, etc.

Required scope: `loop:write`

```
POST /loop-it?profile_id=<profile_id>
```

2.1. Parameters

Name	Type	Description
profile_id	integer	[optional] Id of the individual profile the loop will be created in; *required* in case the account has more than one profile
transactionType	string	[required] Type of transaction (see addendum)
status	string	[required] Status of the loop (see addendum)
name	string	[required] Name of the loop, usually either property address line or lead name (max 200 chars)
streetName	string	[optional] Street name
streetNumber	string	[optional] Street number
unit	string	[optional] Unit number
city	string	[optional] City
state	string	[optional] State
zipCode	string	[optional] Zip code
county	string	[optional] County
country	string	[optional] Country
participants.fullName	string	[optional] Participant's full name
participants.email	string	[optional] Participant's email address
participants.role	string	[optional] Participant's role
templateId	integer	[optional or required] Loop Template Id: (note: my be required by the user's organization (parent profile)
mlsPropertyId	string	[optional] MLS Property Id
mlsId	string	[optional] MLS Id required to search listing
mlsAgentId	string	[optional] MLS Agent Id
nrdsId	string	[optional] NRDS Id

Please be aware of that access to loops is currently restricted to `INDIVIDUAL` profiles only

2.2. Example

```
POST /loop-it?profile_id=4711
{
  "name": "Brian Erwin",
  "transactionType": "PURCHASE_OFFER",
  "status": "PRE_OFFER",
  "streetName": "Waterview Dr",
  "streetNumber": "2100",
  "unit": "12",
  "city": "San Francisco",
  "zipCode": "94114",
  "state": "CA",
  "country": "US",
  "participants": [
    {
      "fullName": "Brian Erwin",
      "email": "brianerwin@newkyhome.com",
      "role": "BUYER"
    },
    {
      "fullName": "Allen Agent",
      "email": "allen.agent@gmail.com",
      "role": "LISTING_AGENT"
    },
    {
      "fullName": "Sean Seller",
      "email": "sean.seller@yahoo.com",
      "role": "SELLER"
    }
  ],
  "templateId": 1424,
  "mlsPropertyId": "43FSB8",
  "mlsId": "789",
  "mlsAgentId": "123456789"
}
```

2.3. Response

The response contains a property `loopUrl`, which can be used to redirect the user to the loop on dotloop.com.

```
Status: 201 Created
```

```
{
  "data": {
    "id": 34308,
    "profileId": 4711,
    "name": "Brian Erwin",
    "transactionType": "PURCHASE_OFFER",
    "status": "PRE_OFFER",
    "created": "2017-05-30T21:42:17Z",
    "updated": "2017-05-31T23:27:11Z",
    "loopUrl": "https://www.dotloop.com/m/loop?viewId=34308"
  }
}
```


2.4. Design Guidelines

In order to allow users to easily spot and interact with Dotloop's *Loop-It™* functionality in 3rd-party products, we advise to implement and visualize the feature as a *Loop-It™* button.

As an example, the *Loop-It™* Button could be rendered next to a property listing, which allows an agent to easily create a loop associated with the listed property. Upon response from the *Loop-It™* API, which contains a perma-link to the created loop, the agent gets prompted by the 3rd party application, whether he wants to transition into dotloop to continue manage loop details or move the transaction forward.

Download Links: PNG | SVG

3. Account

3.1. Get Account Details

Retrieve account details

Required scope: `account:read`

```
GET /account
```

3.1.1. Parameters

None

3.1.2. Response

```
Status: 200 OK
```


```
{
  "data": {
    "id": 1,
    "firstName": "Brian",
    "lastName": "Erwin",
    "email": "brianerwin@newkyhome.com",
    "defaultProfileId": 42
  }
}
```


4. Profiles

4.1. List all Profiles

List all profiles associated with the user.

```
GET /profile
```

Required scope: `profile:read`

4.1.1. Parameters

None

4.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 3
  },
  "data": [
    {
      "id": 3,
      "name": "My Profile",
      "type": "INDIVIDUAL",
      "company": "MyCompany",
      "phone": "+0 (123) 456 7890",
      "fax": "+0 (123) 456 7890",
      "address": "1234 Wall St",
      "city": "New York",
      "state": "NY",
      "zipCode": "10005",
      "default": true,
      "requiresTemplate": true
    },
    ...
  ]
}
```


4.2. Get a Profile

Retrieve an individual profile by id.

Required scope: `profile:read`

```
GET /profile/:profile_id
```

4.2.1. Parameters

None

4.2.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 3,
    "name": "My Profile",
    "type": "INDIVIDUAL",
    "company": "MyCompany",
    "phone": "+0 (123) 456 7890",
    "fax": "+0 (123) 456 7890",
    "address": "1234 Wall St",
    "city": "New York",
    "state": "NY",
    "zipCode": "10005",
    "requiresTemplate": true
  }
}
```


4.3. Create a Profile

Create a new profile.

Required scope: `profile:write`

```
POST /profile
```

4.3.1. Parameters

Name	Type	Description
name	string	profile name
company	string	company name
phone	string	phone number
address	string	address line
city	string	city
zipCode	string	zip code
state	string	state
country	string	country

4.3.2. Example

```
{
  "name": "My Profile",
  "company": "MyCompany",
  "phone": "+0 (123) 456 7890",
  "fax": "+0 (123) 456 7890",
  "address": "1234 Wall St",
  "city": "New York",
  "state": "NY",
  "zipCode": "10005"
}
```

4.3.3. Response

```
Status: 201 Created
```

```
{
  "data": {
    "id": 3,
    "type": "INDIVIDUAL",
    "name": "My Profile",
    "company": "MyCompany",
    "phone": "+0 (123) 456 7890",
    "fax": "+0 (123) 456 7890",
    "address": "1234 Wall St",
    "city": "New York",
    "state": "NY",
    "zipCode": "10005"
  }
}
```

4.4. Update a Profile

Update an existing profile by id.

- This API allows partial updates
- Required scope: `profile:write`

```
PATCH /profile/:profile_id
```

4.4.1. Parameters

Name	Type	Description
name	string	profile name
company	string	company name
phone	string	phone number
address	string	address line
city	string	city
zipCode	string	zip code
state	string	state
country	string	country

4.4.2. Example

```
{
  "name": "My Changed Profile Name",
  "company": "My New Company"
}
```

4.4.3. Response

```
Status: 200 OK
```


```
{
  "data": {
    "id": 3,
    "type": "INDIVIDUAL",
    "name": "My Changed Profile Name",
    "company": "My New Company",
    "phone": "+0 (123) 456 7890",
    "fax": "+0 (123) 456 7890",
    "address": "1234 Wall St",
    "city": "New York",
    "state": "NY",
    "zipCode": "10005"
  }
}
```


5. Loop Summaries

5.1. List all Loops

List all loops associated with a profile.

Required scope: `loop:read`

```
GET
/profile/:profile_id/loop[?batch_size=<batch_size>&batch_number=<batch_number>&sort=<s
ort>&filter=<filter>&include_details=true]
```

5.1.1. Parameters

Name	Type	Description
batch_size	integer	[optional] size of batch returned (default=20, max=100)
batch_number	integer	[optional] batch/page number (default=1)
sort	string	[optional] string which contains the sort category and optionally the sort direction (default ascending); format: `<category>[:asc|desc]`, e.g. `address` or `address:asc` produce the same results. Possible sort categories: `default`, `address`, `created`, `updated`, `purchase_price`, `listing_date`, `expiration_date`, `closing_date`, `review_submission_date`
filter	String	[optional] format: `<filter_key>=<filtervalue>`, filter keys: `updated_min=<timestamp>`, `created_min=<timestamp>`, `transaction_type=<type>[|<type>|…]`, `transaction_status=<status>[|<status>|…]`
include_details	boolean	[optional] flag to include loop details with each record returned; [true|false] (default: false)

5.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 10
  },
  "data": [
    {
      "id": 34308,
      "name": "Atturo Garay, 3059 Main, Chicago, IL 60614",
      "status": "ARCHIVED",
      "transactionType": "PURCHASE_OFFER",
      "totalTaskCount": 5,
      "completedTaskCount": 3,
      "updated": "2017-05-30T21:42:17Z",
      "created": "2017-05-17T01:18:37Z",
      "loopUrl": "https://www.dootloop.com/m/loop?viewId=34308"
    },
    ...
  ]
}
```


5.2. Get a Loop

Retrieve an individual loop by id.

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id
```

5.2.1. Parameters

None

5.2.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 34308,
    "name": "Atturo Garay, 3059 Main, Chicago, IL 60614",
    "status": "ARCHIVED",
    "transactionType": "PURCHASED",
    "totalTaskCount": 5,
    "completedTaskCount": 3,
    "updated": "2017-05-30T21:42:17Z",
    "created": "2017-05-17T01:18:37Z",
    "loopUrl": "https://www.dootloop.com/m/loop?viewId=34308"
  }
}
```


```
Status: 301 Moved Permanently
```

In some scenarios, two separate loops can be merged together, which can change the original loop ID. In those cases, attempting to access the original loop ID will produce a 301 response that points to the new loop. Any clients that persist loop IDs should account for this scenario and be able to update any references when a 301 is encountered.

See the support article for more information on the loop merging process: [Merge Loops](https://support.dotloop.com/s/article/Merge-Loops).

The 301 response will be have a `Location` header present with a path to redirect to the new Loop View.

Headers

```
Location: /public/v2/profile/3/loop/30004
```

To get the Loop use the value from the `Location` header to do the next call.

When Autoredirect is enabled the second call will be done automatically and will get you a response as shown in `200 Response`.

5.3. Create a Loop

Create a new loop.

Required scope: `loop:write`

```
POST /profile/:profile_id/loop
```

5.3.1. Parameters

Name	Type	Description
name	string	the name of the loop (max 200 chars)
status	string	status of the loop
transactionType	string	type of transaction

5.3.2. Example

```
{
  "name": "Atturo Garay, 3059 Main, Chicago, IL 60614",
  "status": "PRE_LISTING",
  "transactionType": "LISTING_FOR_SALE"
}
```

5.3.3. Response

```
Status: 201 Created
```

```
{
  "data": {
    "id": 34308,
    "profileId": 23483,
    "name": "Atturo Garay, 3059 Main, Chicago, IL 60614",
    "transactionType": "LISTING_FOR_SALE",
    "status": "PRE_LISTING",
    "totalTaskCount": 5,
    "completedTaskCount": 3,
    "created": "2017-05-17T01:18:37Z",
    "updated": "2017-05-17T01:18:37Z",
    "loopUrl": "https://www.dootloop.com/m/loop?viewId=34308"
  }
}
```

5.4. Update a Loop

Update an existing loop by id.

- This API allows partial updates
- Required scope: `loop:write`

```
PATCH /profile/:profile_id/loop/:loop_id
```

5.4.1. Parameters

Name	Type	Description
name	string	the name of the loop (max 200 chars)
status	string	status of the loop
transactionType	string	type of transaction

5.4.2. Example

```
{
  "status": "SOLD"
}
```

5.4.3. Response

```
Status: 200 OK
```


```
{
  "data": {
    "id": 34308,
    "name": "Atturo Garay, 3059 Main, Chicago, IL 60614",
    "transactionType": "LISTING_FOR_SALE",
    "status": "SOLD",
    "totalTaskCount": 5,
    "completedTaskCount": 3,
    "updated": "2017-05-30T21:42:17Z",
    "created": "2017-05-17T01:18:37Z",
    "loopUrl": "https://www.dootloop.com/m/loop?viewId=34308"
  }
}
```


6. Loop Details

6.1. Get Loop Details

Retrieve loop details by id.

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/detail
```

6.1.1. Parameters

Details Section	Field	Type	Description
'Property Address'	'Country'	string	
'Property Address'	'Street Number'	string	
'Property Address'	'Street Name'	string	
'Property Address'	'Unit Number'	string	
'Property Address'	'City'	string	
'Property Address'	'State/Prov'	string	
'Property Address'	'Zip/Postal Code'	string	
'Property Address'	'County'	string	
'Property Address'	'MLS Number'	string	
'Property Address'	'Parcel/Tax ID'	string	
'Financials'	'Purchase/Sale Price'	string	
'Financials'	'Sale Commission Rate'	string	
'Financials'	'Sale Commission Split % - Buy Side'	string	
'Financials'	'Sale Commission Split % - Sell Side'	string	
'Financials'	'Sale Commission Total'	string	
'Financials'	'Earnest Money Amount'	string	
'Financials'	'Earnest Money Held By'	string	
'Financials'	'Sale Commission Split $ - Buy Side'	string	
'Financials'	'Sale Commission Split $ - Sell Side'	string	
'Contract Dates'	'Contract Agreement Date'	string	date string, e.g. 01/31/2017
'Contract Dates'	'Closing Date'	string	date string, e.g. 01/31/2017
'Offer Dates'	'Inspection Date'	string	date string, e.g. 01/31/2017

Details Section	Field	Type	Description
'Offer Dates'	'Offer Date'	string	date string, e.g. 01/31/2017
'Offer Dates'	'Offer Expiration Date'	string	date string, e.g. 01/31/2017
'Offer Dates'	'Occupancy Date'	string	date string, e.g. 01/31/2017
'Offer Dates'	'Offer Date'	string	date string, e.g. 01/31/2017
'Contract Info'	'Transaction Number'	string	
'Contract Info'	'Class'	string	
'Contract Info'	'Type'	string	
'Referral'	'Referral %'	string	
'Referral'	'Referral Source'	string	
'Listing Information'	'Expiration Date'	string	date string, e.g. 01/31/2017
'Listing Information'	'Listing Date'	string	date string, e.g. 01/31/2017
'Listing Information'	'Original Price'	string	
'Listing Information'	'Current Price'	string	
'Listing Information'	'1st Mortgage Balance'	string	
'Listing Information'	'2nd Mortgage Balance'	string	
'Listing Information'	'Other Liens'	string	
'Listing Information'	'Description of Other Liens'	string	
'Listing Information'	'Homeowner's Association'	string	
'Listing Information'	'Homeowner's Association Dues'	string	
'Listing Information'	'Total Encumbrances'	string	
'Listing Information'	'Property Includes'	string	
'Listing Information'	'Property Excludes'	string	
'Listing Information'	'Remarks'	string	
'Geographic Description'	'MLS Area'	string	
'Geographic Description'	'Legal Description'	string	

Details Section	Field	Type	Description
'Geographic Description'	'Map Grid'	string	
'Geographic Description'	'Subdivision'	string	
'Geographic Description'	'Lot'	string	
'Geographic Description'	'Deed Page'	string	
'Geographic Description'	'Deed Book'	string	
'Geographic Description'	'Section'	string	
'Geographic Description'	'Addition'	string	
'Geographic Description'	'Block'	string	
'Property'	'Year Built'	string	
'Property'	'Bedrooms'	string	
'Property'	'Square Footage'	string	
'Property'	'School District'	string	
'Property'	'Type'	string	
'Property'	'Bathrooms'	string	
'Property'	'Lot Size'	string	

6.1.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "Property Address": {
      "Country": "USA",
      "Street Number": "333",
      "Street Name": "Main St",
      "Unit Number": "123",
      "City": "San Francisco",
      "State/Prov": "CA",
      "Zip/Postal Code": "94105",
      "County": "USA",
      ...
    },
    "Financials": {
      "Sale Commission Rate": "3",
      "Sale Commission Split % - Buy Side": "50",
      "Sale Commission Split % - Sell Side": "50",
      "Sale Commission Total": "10000",
      "Sale Commission Split $ - Buy Side": "50",
      "Sale Commission Split $ - Sell Side": "20000",
      ...
    },
    ...
  }
}
```


6.2. Update Loop Details

Update loop details by id.

- This API allows partial updates
- Required scope: `loop:write`

```
PATCH /profile/:profile_id/loop/:loop_id/detail
```

6.2.1. Parameters

See Get Loop Details above.

6.2.2. Example

```
{
    "Financials": {
      "Purchase/Sale Price": "342342"
    }
}
```

6.2.3. Response

```
Status: 200 OK
```

```
{
  "data": {
    "Property Address": {
      "Country": "USA",
      "Street Number": "333",
      "Street Name": "Main St",
      "Unit Number": "123",
      "City": "San Francisco",
      "State/Prov": "CA",
      "Zip/Postal Code": "94105",
      "County": "USA",
      ...
    },
    "Financials": {
      "Purchase/Sale Price": "342342",
      "Sale Commission Rate": "3",
      "Sale Commission Split % - Buy Side": "50",
      "Sale Commission Split % - Sell Side": "50",
      "Sale Commission Total": "10000",
      "Sale Commission Split $ - Buy Side": "50",
      "Sale Commission Split $ - Sell Side": "20000",
      ...
    },
    ...
  }
}
```

7. Loop Folders

7.1. List all Folders

List all folders in a loop

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/folder[?include_documents=<include_documents>]
```

7.1.1. Parameters

Name	Type	Description
include_documents	boolean	Include a list of all documents in all folders

7.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 4
  },
  "data": [
    {
      "id": 423424,
      "name": "Disclosures",
      "created": "2017-05-17T01:18:37Z",
      "updated": "2017-05-30T21:42:17Z"
    },
    ...
  ]
}
```

7.2. Get a Folder

Retrieve an individual folder by id.

Required scope: `loop:read`

```
GET
/profile/:profile_id/loop/:loop_id/folder/:folder_id[?include_documents=<include_documents>]
```

7.2.1. Parameters

Name	Type	Description
include_documents	boolean	include a list of all documents in the folder

7.2.2. Response

```
Status: 200 OK
```

```
{
  "data":{
    "id": 423424,
    "name": "Disclosures",
    "created": "2017-05-17T01:18:37Z",
    "updated": "2017-05-30T21:42:17Z"
  }
}
```

7.3. Create a Folder

Create a new folder.

Required scope: `loop:write`

```
POST /profile/:profile_id/loop/:loop_id/folder/
```

7.3.1. Parameters

Name	Type	Description
name	string	the name of the folder (max ??? chars)

7.3.2. Example

```
{
  "name": "Disclosures"
}
```

7.3.3. Response

```
Status: 201 Created
```

```
{
"data":{
    "id": 423424,
    "name": "Disclosures",
    "created": "2017-05-17T01:18:37Z",
    "updated": "2017-05-30T21:42:17Z"
  }
}
```


7.4. Update a Folder

Update an existing folder by id.

- This API allows partial updates
- Required scope: `loop:write`

```
PATCH /profile/:profile_id/loop/:loop_id/folder/:folder_id
```

7.4.1. Parameters

Name	Type	Description
name	string	the name of the folder (max ??? chars)

7.4.2. Example


```
{
  "name": "Disclosures (renamed)"
}
```


7.4.3. Response

```
Status: 200 OK
```


```
{
  "data":{
    "id": 423424,
    "name": "Disclosures (renamed)"
    "created": "2017-05-17T01:18:37Z",
    "updated": "2017-05-30T21:42:17Z"
  }
}
```

8. Loop Documents

8.1. List all Documents

List all documents in a loop

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/folder/:folder_id/document
```

8.1.1. Parameters

None

8.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 3
  },
  "data": [
    {
      "id": 561621,
      "filename": "disclosures.pdf",
      "created": "2017-05-17T01:18:37Z",
      "updated": "2017-05-17T01:18:37Z"
    }, ...
  ]
}
```

8.2. Get a Document

Retrieve an individual document by document_id

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/folder/:folder_id/document/:document_id
Accept: application/json
```

8.2.1. Parameters

None

8.2.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 561621,
    "name": "disclosures.pdf",
    "created": "2017-05-17T01:18:37Z",
    "updated": "2017-05-17T01:18:37Z"
  }
}
```

8.3. Upload a Document

Upload a individual document (binary) via multipart form post

Required scope: `loop:write`

```
POST /profile/:profile_id/loop/:loop_id/folder/:folder_id/document/
content-type: multipart/form-data; boundary=<BOUNDARY>
content-length: XXX


--<BOUNDARY>
Content-Disposition: form-data; name="file"; fileName="disclosures.pdf"
Content-Type: application/pdf

<binary data>
--<BOUNDARY>--
```

8.3.1. Parameters

Name	Type	Description
fileName	string	fileName of the pdf

8.3.2. Example using curl:

```
$ curl -F
"file=@\"/Users/you/Documents/disclosures.pdf\";fileName=\"my_disclosures.pdf\";type=a
pplication/pdf" -H "Authorization: Bearer <token>" https://api-
gateway.dotloop.com/public/v2/profile/:profile_id/loop/:loop_id/folder/:folder_id/docu
ment/
```

8.3.3. Response

```
Status: 201 OK
```

```
{
  "data": {
    "id": 561621,
    "name": "my_disclosures.pdf",
    "created": "2017-05-17T01:18:37Z",
    "updated": "2017-05-17T01:18:37Z"
  }
}
```

9. Loop Participants

9.1. List all Loop Participants

List all loop participants in a loop

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/participant
```

9.1.1. Parameters

None

9.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 3
  },
  "data": [
    {
      "id": 2355,
      "fullName": "Brian Erwin",
      "email": "brianerwin@newkyhome.com",
      "role": "BUYER",
      "Phone": "(555) 555-5555"
    },
    {
      "id": 57567,
      "fullName": "Allen Agent",
      "email": "allen.agent@gmail.com",
      "role": "LISTING_AGENT",
      "Phone": "(555) 555-1234",
      "Company Name":  "Allen Realty"
    },
    {
      "id": 24743,
      "fullName": "Sean Seller",
      "email": "sean.seller@yahoo.com",
      "role": "SELLER",
      "Street Name": "123",
      "Street Number": "Main St.",
      "City": "Cincinnati",
      "Zip/Postal Code": "45123",
      "Country":  "USA",
      "Cell Phone": "(555) 555-4444"
    }
  ]
}
```


9.2. Get a Loop Participant

Retrieve loop participants details of an individual loop participant.

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/participant/:participant_id
```

9.2.1. Parameters

None

9.2.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 2355,
    "fullName": "Brian Erwin",
    "email": "brianerwin@newkyhome.com",
    "role": "BUYER",
    "Phone": "(555) 555-5555"
  }
}
```

9.3. Add a Loop Participant

Add a new loop participant

- Required scope: `loop:write`

```
POST /profile/:profile_id/loop/:loop_id/participant
```

9.3.1. Parameters

Name	Type	Description
fullName	string	First and last name of the participant
email	string	participant email
role	string	participant role
'Street Name'	string	[optional] street number of participant's address
'Street Number'	string	[optional] street name of participant's address
'City'	string	[optional] city of participant's address
'State/Prov'	string	[optional] state/providence of participant's address
'Zip/Postal Code'	string	[optional] postal code of participant's address
'Unit Number'	string	[optional] unit # of participant's address
'Country'	string	[optional] country of participant's address
'Phone'	string	[optional] participant phone number
'Cell Phone'	string	[optional] participant mobile number
'Company Name'	string	[optional] participant company

Additional role-specific fields can also be provided. See Built-in Contact/Loop Participant Roles for details.

9.3.2. Example

```
{
  "fullName": "Brian Erwin",
  "email": "brian@gmail.com",
  "role": "BUYER",
  "Street Name": "123",
  "Street Number": "Main St.",
  "City": "Cincinnati",
  "Zip/Postal Code": "45123",
  "Country":  "USA",
  "Phone": "(555) 555-5555",
  "Cell Phone": "(555) 555-4444",
  "Company Name":  "Buyer's Company"
}
```

9.3.3. Response

```
Status: 201 Created
```

```
{
  "data": {
    "id": 2355,
    "fullName": "Brian Erwin",
    "email": "brianerwin@newkyhome.com",
    "role": "BUYER",
    "Street Name": "123",
    "Street Number": "Main St.",
    "City": "Cincinnati",
     "Zip/Postal Code": "45123",
     "Country":  "USA",
     "Phone": "(555) 555-5555",
     "Cell Phone": "(555) 555-4444",
     "Company Name":  "Buyer's Company"
  }
}
```

9.4. Update a Loop Participant

Update an existing participant

- This API allows partial updates
- Required scope: `loop:write`

```
PATCH /profile/:profile_id/loop/:loop_id/participant/:participant_id
```

9.4.1. Parameters

Name	Type	Description
fullName	string	First and last name of the participant
email	string	participant email
role	string	participant role
'Street Name'	string	street number of participant's address
'Street Number'	string	street name of participant's address
'City'	string	city of participant's address
'State/Prov'	string	state/providence of participant's address
'Zip/Postal Code'	string	postal code of participant's address
'Unit Number'	string	unit # of participant's address
'Country'	string	country of participant's address
'Phone'	string	participant phone number
'Cell Phone'	string	participant mobile number
'Company Name'	string	participant company

9.4.2. Example

```
{
  "email": "brian@gmail.com"
}
```

9.4.3. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 2355,
    "fullName": "Brian Erwin",
    "email": "brian@gmail.com",
    "role": "BUYER",
    "Phone": "(555) 555-5555"
  }
}
```


9.5. Delete a Loop Participant

Delete an existing participant by id.

Required scope: `loop:write`

```
DELETE /profile/:profile_id/loop/:loop_id/participant/:participant_id
```

9.5.1. Parameters

none

9.5.2. Response

```
Status: 204 No Content
```

10. Loop Tasks

10.1. List all Loop Task Lists

List all task lists in a loop

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/tasklist/
```

10.1.1. Parameters

None

10.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 3
  },
  "data": [
    {
      "id": 1234,
      "name": "My Tasks"
    }, ..
  ]
}
```

10.2. Get a Loop Task List

Retrieve an individual task list.

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/tasklist/:task_list_id
```

10.2.1. Parameters

None

10.2.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 1234,
    "name": "My Tasks"
  }
}
```

10.3. List all Loop Task List Items

List all task items in a task list

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/tasklist/:task_list_id/task
```

10.3.1. Parameters

None

10.3.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 4
  },
  "data": [
    {
      "id": 125736485,
      "name": "contract",
      "due": "2016-10-21T00:00:00-04:00",
      "completed": true
    },
    ...
  ]
}
```

10.4. Get a Loop Task List item

Retrieve an individual task list item.

Required scope: `loop:read`

```
GET /profile/:profile_id/loop/:loop_id/tasklist/:task_list_id/task/:task_list_item_id
```

10.4.1. Parameters

None

10.4.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 125736485,
    "name": "contract",
    "due": "2016-10-21T00:00:00-04:00"
  }
}
```

11. Loop Activities

11.1. List all Loop Activities

List all activities for a loop

Required scope: `loop:read`

```
GET
/profile/:profile_id/loop/:loop_id/activity[?batch_size=<batch_size>&batch_number=<bat
ch_number>]
```

11.1.1. Parameters

Name	Type	Description
batch_size	integer	size of batch returned (default=20, max=100)
batch_number	integer	batch/page number (default=1)

11.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": -1
  },
  "data": [
    {
      "message": "User One viewed document Agency Disclosure Statement - Seller",
      "date": "2017-01-09T13:10:14Z"
    },
    ...
  ]
}
```

The `meta/total` count is currently returned as `-1`, ie in order to know how many activities are present the caller needs to paginate the the entire result.

12. Contacts

12.1. List all Contacts

List all contacts in the user account.

Required scope: `contact:read`

```
GET /contact[?batch_size=<batch_size>&batch_number=<batch_number>&filter=<filter>]
```

12.1.1. Parameters

Name	Type	Description
batch_size	integer	size of batch returned (default=20, max=100)
batch_number	integer	batch/page number (default=1)
filter	String	format: `<filter_key>=<filtervalue>`, filter keys: `updated_min=<timestamp>`

12.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 10
  },
  "data": [
    {
      "id": 3603862,
      "firstName": "Brian",
      "lastName": "Erwin",
      "email": "brianerwin@newkyhome.com",
      "home": "(415) 8936 332",
      "office": "(415) 1213 656",
      "fax": "(415) 8655 686",
      "address": "2100 Waterview Dr",
      "city": "San Francisco",
      "zipCode": "94114",
      "state": "CA",
      "country": "US",
      "updated": "2017-04-20T03:48:30Z"
    },
    ...
  ]
}
```


12.2. Get a Contact

Retrieve an individual contact by id.

Required scope: `contact:read`

```
GET /contact/:contact_id
```

12.2.1. Parameters

None

12.2.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 3603862,
    "firstName": "Brian",
    "lastName": "Erwin",
    "email": "brianerwin@newkyhome.com",
    "home": "(415) 8936 332",
    "office": "(415) 1213 656",
    "fax": "(415) 8655 686",
    "address": "2100 Waterview Dr",
    "city": "San Francisco",
    "zipCode": "94114",
    "state": "CA",
    "country": "US",
    "updated": "2017-04-20T03:48:30Z"
  }
}
```


12.3. Create a Contact

Create a new contact.

Required scope: `contact:write`

```
POST /contact
```

12.3.1. Parameters

Name	Type	Description
firstName	string	first name
lastName	string	last name
email	string	email address
home	string	home phone number
office	string	office phone number
fax	string	fax number
address	string	address line
city	string	city
zipCode	string	zip code
state	string	state
country	string	country

12.3.2. Example

```
{
  "firstName": "Brian",
  "lastName": "Erwin",
  "email": "brianerwin@newkyhome.com",
  "home": "(415) 8936 332",
  "office": "(415) 1213 656",
  "fax": "(415) 8655 686",
  "address": "2100 Waterview Dr",
  "city": "San Francisco",
  "zipCode": "94114",
  "state": "CA",
  "country": "US"
}
```

12.3.3. Response

```
Status: 201 Created
```

```
{
  "data": {
    "id": 3603862,
    "firstName": "Brian",
    "lastName": "Erwin",
    "email": "brianerwin@newkyhome.com",
    "home": "(415) 8936 332",
    "office": "(415) 1213 656",
    "fax": "(415) 8655 686",
    "address": "2100 Waterview Dr",
    "city": "San Francisco",
    "zipCode": "94114",
    "state": "CA",
    "country": "US",
    "updated": "2017-04-20T03:48:30Z"
  }
}
```

12.4. Update a Contact

Update an existing contact by id.

- This API allows partial updates
- Required scope: `contact:write`

```
PATCH /contact/:contact_id
```

12.4.1. Parameters

Name	Type	Description
firstName	string	first name
lastName	string	last name
email	string	email address
home	string	home phone number
office	string	office phone number
fax	string	fax number
address	string	address
city	string	city
zipCode	string	zip code
state	string	state
country	string	country

12.4.2. Example

```
{
  "home": "(415) 888 8888"
}
```

12.4.3. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 3603862,
    "firstName": "Brian",
    "lastName": "Erwin",
    "email": "brianerwin@newkyhome.com",
    "home": "(415) 888 8888",
    "office": "(415) 1213 656",
    "fax": "(415) 8655 686",
    "address": "2100 Waterview Dr",
    "city": "San Francisco",
    "zipCode": "94114",
    "state": "CA",
    "country": "US",
    "updated": "2017-04-20T03:48:30Z"
  }
}
```


12.5. Delete a Contact

Delete an existing contact by id.

Required scope: `contact:write`

```
DELETE /contact/:contact_id
```

12.5.1. Parameters

none

12.5.2. Response

```
Status: 204 No Content
```

13. Loop Templates

13.1. List all Loop Templates

List all loop templates in the profile

Required scope: `template:read` or `loop:write`

```
GET /profile/:profile_id/loop-template
```

13.1.1. Parameters

None

13.1.2. Response

```
Status: 200 OK
```

```
{
  "meta": {
    "total": 5
  },
  data: [
    {
      "id": 423,
      "profileId": 732453,
      "name": "My Loop Template",
      "transactionType": "PURCHASE_OFFER",
      "shared": true,
      "global": false
    },
    ...
  ]
}
```

13.2. Retrieve an Individual Loop Template

Retrieve an individual loop template by id.

Required scope: `template:read` or `loop:write`

```
GET /profile/:profile_id/loop-template/:loop_template_id
```

13.2.1. Parameters

None

13.2.2. Response

```
Status: 200 OK
```

```
{
  "data": {
    "id": 423,
    "profileId": 732453,
    "name": "My Loop Template",
    "transactionType": "PURCHASE_OFFER",
    "shared": true,
    "global": false
  }
}
```


14. Addendum

14.1. Types / Constants

14.1.1. Built-in Contact/Loop Participant Roles

- `ADMIN`
  - optional fields: `ID`, `License #`
- `APPRAISER`
  - optional fields: `ID`, `License #`
- `BUYER_ATTORNEY`
  - optional fields: `ID`, `License #`
- `BUYER`
  - optional fields: `Marital Status`
- `BUYING_AGENT`
  - optional fields: `Fax`, `ID`, `License #`
- `BUYING_BROKER`
  - optional fields: `Fax`,`ID`,`License #`
- `ESCROW_TITLE_REP`
  - optional fields: `ID`, `License #`
- `HOME_IMPROVEMENT_SPECIALIST`
  - optional fields: `ID`, `License #`
- `HOME_INSPECTOR`
  - optional fields: `ID`, `License #`
- `HOME_SECURITY_PROVIDER`
  - optional fields: `ID`, `License #`

- HOME_WARRANTY_REP
  - optional fields: ID, License #
- INSPECTOR
  - optional fields: ID, License #
- INSURANCE_REP
  - optional fields: ID, License #
- LANDLORD
  - optional fields: ID, License #
- LISTING_AGENT
  - optional fields: Fax, ID, License #
- LISTING_BROKER
  - optional fields: Fax, ID, License #
- LOAN_OFFICER
  - optional fields: ID, License #
- LOAN_PROCESSOR
  - optional fields: ID, License #
- MANAGING_BROKER
  - optional fields: ID, License #
- MOVING_STORAGE
  - optional fields: ID, License #
- OTHER
  - optional fields: ID, License #
- PROPERTY_MANAGER
  - optional fields: ID, License #
- SELLER_ATTORNEY
  - optional fields: ID, License #
- SELLER
  - optional fields: Marital Status
- TENANT_AGENT
  - optional fields: ID, License #
- TENANT
  - optional fields: ID, License #, Marital Status
- TRANSACTION_COORDINATOR

    - optional fields: `ID`, `License #`
- `UTILITIES_PROVIDER`
    - optional fields: `ID`, `License #`

Custom roles are not listed here

14.1.2. Profile Types

- `INDIVIDUAL`
- `TEAM`
- `OFFICE`
- `COMPANY`
- `ASSOCIATION`
- `NATIONAL_PARTNER`

14.1.3. Loop Transactions corresponding Statuses

- `PURCHASE_OFFER`
    - `PRE_OFFER`
    - `UNDER_CONTRACT`
    - `SOLD`
    - `ARCHIVED`
- `LISTING_FOR_SALE`
    - `PRE_LISTING`
    - `PRIVATE_LISTING`
    - `ACTIVE_LISTING`
    - `UNDER_CONTRACT`
    - `SOLD`
    - `ARCHIVED`
- `LISTING_FOR_LEASE`
    - `PRE_LISTING`
    - `PRIVATE_LISTING`
    - `ACTIVE_LISTING`
    - `UNDER_CONTRACT`
    - `LEASED`
    - `ARCHIVED`
- `LEASE_OFFER`
    - `PRE_OFFER`
    - `UNDER_CONTRACT`
    - `LEASED`
    - `ARCHIVED`

- REAL_ESTATE_OTHER
  - NEW
  - IN_PROGRESS
  - DONE
  - ARCHIVED
- OTHER
  - NEW
  - IN_PROGRESS
  - DONE
  - ARCHIVED

15. Client Errors

15.1. HTTP status codes

This API follows the common HTTP status code semantics. List of possible Client errors:

Error Code	Description
400 Bad Request	The request is invalid, e.g. request payload can't be parsed
401 Unauthorized	The access token is invalid or expired. Obtain a new token using the refresh token and insert into request header: `Authorization: Bearer <access token>`
403 Forbidden	The request is denied, e.g. you don't have the privileges to access or create the resource
404 Not Found	The requested resource does not exist
422 Unprocessable Entity	The syntax of the request is correct but semantically erroneous
429 Too Many Requests	The caller exceeded the rate limit

15.2. Error responses

Error responses may contain one or many `error` items. A human-readable explanation may be provided in the `detail` property. The `source` may indicate to the origin of the error and `code` provides a application specific error code.

15.2.1. Example

```
Status: 400 OK
```

```
{
  "errors": [
    {
      "code": "123",
      "source": { "pointer": "/data/attributes/profileid" },
      "detail": "Profile id invalid"
    },
    {
      "source": { "parameter": "include" },
      "detail": "include param required but not present"
    }
  ]
}
```

16. Changelog

- *09/06/2018*
  - Added support to filter for multiple transaction types, e.g. `transaction_status=PRE_OFFER|PRE_LISTING` Loop API
- *08/08/2018*
  - Adding support for updating custom loop template fields in Loop Detials API
- *04/20/2018*
  - Include document metadata in Folder API via "include_documents" parameter
  - `role` is now a required field when creating Loop Participants
- *03/13/2018*
  - Added additional data fields to participants api
- *11/01/2017*
  - Fix issue where opening downloaded documents via the API triggered a print dialog to appear
- *10/04/2017*
  - Fix issue where `updated` field for loops was not updated upon document uploads
- *09/20/2017*
  - Global Templates are now applied at loop creation time
- *08/24/2017*
  - Fix issue where `updated` field for loops was not updated upon task changes
  - Add `updated` timestamp for document entities

07/26/2017

- Added support to filter for multiple transaction types, e.g. `transaction_type=PURCHASE_OFFER|LISTING_FOR_SALE`
- Introducing Folder API to create, list and rename folders
- Introducing Document API to upload and download documents
- FIX: `404 Not Found` was returned in some users whose default profile id is not set correctly.

- *07/12/2017*
  - Introduce Activity API which retrieve all loop activities
  - New Filter syntax with param `?filter=⋯` (we continue to support `updated_min` parameter which is now deprecated) Examples: Contact API, Loop API
  - 2 new Loop filters: a) `created_min` to list all loops created after a specific time, b) `transaction_type` to list all loops of the specified transaction type
  - New sort field: Sort Loops by `created` timestamp
  - Include Loop details in summaries via `?include_details=true`

- *06/28/2017*
  - FIX: templates can now also be read with `loop:write` scope as they might be required to create a loop
  - FIX: Loop-It™ does not fail if the a participant is added with the same email as the account email (ignored)

- *06/16/2017*
  - Allow redirect to the `redirect_uri` via new `&redirect_on_deny=[true|false]` param, which allows client applications to control the experience if user denies access, see here.
  - FIX: Show correct total counts (in meta section) in `GET /loop` responses
  - FIX: Fix `403 Forbidden` issue affecting some users

- *05/31/2017*
  - Loops and Contacts returned with `created` timestamp (in addition to `updated` field)
  - Custom contact or loop participant roles are supported now

- *05/17/2017*
  - Loop list supports new query param `updated_min` to select loops updated since a timestamp
  - `requiresTemplate` flag on profile which defines whether a template id is required to create a loop
  - FIX: transaction type in the request takes precedence over loop template transaction type now

- *05/10/2017*
  - Loop Summaries contains `loopUrl` property now
-

    - Contacts API supports `company` and `role` property now
    - Loop list can be sorted now (e.g. `⋯&sort=purchase_price:desc`)

- *04/19/2017*
    - Added the ability to add, update and delete Loop Participants
    - Paginate thru loop and contact lists via new params `batch_size` and `batch_number`
    - Contacts API changes
        a. contact items now have a last `updated` timestamp
        b. introduce `updated_min` query param to select contacts updated since a timestamp

- *04/05/2017*
    - Introducing Loop Tasks APIs

- *03/23/2017*
    - Introduced `deactivated` flag for profiles
    - Loop-it™ returns now mobile-ready URLs
    - Loop task count (total/completed) returned in loop APIs

- *03/08/2017*
    - Introduced `default` flag for profiles
    - Added new `GET /account` API (token requires `account:read` scope to be able to access this api)

17. FAQ

Are there more than one access or refresh token valid for a particular client/user combination at a given time?

No, there can be only 1 token valid at a time. This also means if you refresh an access token, any previously issued access token for that user will be invalidated.

Does my application need to share refresh/access token across instances in a clustered environment?

Yes, otherwise all instances within your cluster may start racing to refresh the access token.

Are there request/rate limits in place?

Yes. We rate limit client requests in order to protect our service against abuse or DOS attacks. Today we allow each client application to make up to 100 requests per minute for a user. Once client exceed this limit, they'll receive a 429 Error response and need to wait till the rate limit gets reset. Clients can track their limits by evaluating the following response headers which indicate the actual limit, the remaining calls and when the limit gets reset (ms), e.g.:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 34
X-RateLimit-Reset: 32000
```

I'm getting a 403 ACCESS DENIED error returned from the API - what could be wrong?

There could be multiple reasons for a 403 error returned by the API:

1. *You may be attempting to access a non-*`INDIVIDUAL` *profile (e.g.* `OFFICE` *or* `COMPANY` *profiles)*
2. *You may be using the wrong token when accessing a profile. Tokens are issued on behalf of a user, so you can only access resources which are the user has permission to access*
3. *The scope of you client may be incorrect hence the client is not authorized to make the API request. Example: The application is trying to update a loop but the application has only* `loop:read` *scope (required:* `loop:write` *or* `loop:*`*)*