OpenTHC API

Table of Contents

The OpenTHC API Specification is not an API for a specific system, rather it is an approach towards a **common** API for the cannabis industry. This document, and it's contents should be viewed as proposed guidelines.

Chapter 1. Overview

There are currently 100s of new software vendors in the cannabis technology space, some with APIs, some without. Each of these systems, as well as the government software provided by BioTrack, METRC, LeafData, etc, has a unique approach, with unique terminology to the same core data. This makes interoperability difficult, or at least tedious.

These proposals include a common data model, with provided JSON schema and samples as well as a REST (or JSON-RPC) style API. These models and interface are hopefully useful for others constructing tools in this space.

We want these standard base data models for objects in the Cannabis Industry to represent the common data all of our software shares with a common language and provide a basis for data increased interoperability.

With this foundation maybe we can all move a little faster.

Chapter 2. General

Unless stated otherwise, these things generally hold true:

Chapter 3. Terms

Common terminology helps

3.1. Basic Elements

Company

A Company is a basic container for the License and Contact details. A Company has a unique identifier and a name but little else.

License

A License is the container for all regulated materials such as Plants, Hash Lots and Transfers. A License will have a unique identifier, a name and also a **License_Type**.

Contact

A Contact is a container for any natural-person or bot-script interacting with the system. They may have an email address, phone.

3.2. Materials

Product

A descripton of items being sold, including name, weight, volume, package details. An Inventory Lot is of a specific Product, that is self is of a **Product_Type**.

Variety

Mostly called Strain, but generally, across agriculture and horticulture this word is used more often.

Inventory

Or, properly Inventory Lot is any unique production run of some Product. Sales of a Product are taken from one or more Inventory.

B2B Transaction

Also called **Transfer** or **Manifest**. A sale or transfer of materials from one **License** to another.

B2C Transaction

Also called **Sale** or **Retail**. A sale or transfer of materials from one **License** to end-consumer (Contact). == Data Format Standards

3.3. Object Identifiers

By default all objects in the system are identified using ULID values. These are awesome because distributed systems can use them without collision, they're easy to search.

Part of the ULID generation routine depends on time, to generate the first few bits. This also allows us to use special values. Well known objects in the OpenTHC universe have all been assigned values from our historical birthdate.

- ULID Prefix: 018NY6XC00
- Date: 2014-04-20T00:00:00.000+00:00
- UNIX Timestamp: 1397952000

This unique pool of **2^80** identifiers MUST be well-published and future values should be assigned by consensus.

3.3.1. Alternate Identifiers

The OpenTHC system, is capable of using nearly any identifier type, so ULID could be replaced with a special scheme. An implementation specific object-identifier-adapter would need to be constructed. This allows OpenTHC to also accept and work with identifiers generated by other systems such as BioTrack, Franwell/METRC or MJ Freeway/LeafData.

3.4. Dates and Times

Date and Time values should be expressed using RFC 3339 formats which are an extension of ISO-8601. Systems are expected to be able to handle data with either of those types. Any date-time values that are missing time-zone information MUST be assumed as UTC.

3.5. Units of Measure

Weights and Volume values should be expressed using the International System of Units (https://en.wikipedia.org/wiki/International_System_of_Units)

The system store all values internally in grams or liters, accurate to four decimal places. That is, accurate to 0.1 milligram/milliliter; expressed internally as grams or liters.

Weights can be input in any of the following values:

- Grams (g)
- Milligrams (mg)
- Kilograms (kg)
- US Pounds (lb)
- US Ounces (oz)

Volume can be input in any of the following values

- Liters (l)
- Milliliters (ml)
- US Ounces (fl oz)

The input parameter (typically **uom**) are specified using the standard abbreviation. === REST API

The OpenTHC API specification follows typical REST defacto-standards. We use the HTTP verbs in traditional ways. Using GET or HEAD or OPTIONS to check the status of an object. Using POST, PUT

or PATCH to create or modify objects.

All responses will be in `application/json` format have at most two keys/properties. The first is `meta` which will contain interesting information. The second is `data` which will contain either a singular object, or an array of objects of a specific type.

3.5.1. GET & HEAD

All end-points accept a GET, and HEAD query. For non-specific resource endpoints the GET query performs a search.

```
GET /plant
```

Would return a list of all the plant objects, or one could add a filter.

```
GET /plant?status=live&section=BadMoterFinger
```

For specific resources, the request includes their ID, no query paramters are used.

```
GET /plant/01DC7XWCB3R3XMRMJZ2839M41E
```

For specific resources, to simply check their status use HEAD

```
HEAD /plant/01DC7XWCB3R3XMRMJZ2839M41E
```

3.5.2. POST

The POST requests are use to create or update resources

To create an object, for example, send FormData or JSON to:

```
POST /plant
```

And to update an object one would POST or PUT to that specific resource.

```
POST /plant/01DC7XWCB3R3XMRMJZ2839M41E
content-type: application/json

{ ... }
```

3.5.3. DELETE

For regulated data, the DELETE is a two step operation. On the first request to DELETE an object,

such as:

```
DELETE /plant/01DC7XWCB3R3XMRMJZ2839M41E
```

The system will respond with a `202 Accepted` level status code. The DELETE request has been accepted. The system must receive a **second** DELETE request to confirm, the response of `410 Gone` will confirm the removal. Subsequent DELETE requests for the object would return `423 Locked`.

See Also: https://tools.ietf.org/html/rfc7231#section-4.3.5

Chapter 4. Authentication

Authentication to OpenTHC can occur through different methods with a preference for oAuth2 When OpenTHC is connecting through to a back-end system some of those parameters may need to be passed as well.

4.1. Open Connection

Authentication occurs through an `/auth/open` request that looks something like this.

```
$API_BASE/auth/open
```

OpenTHC will respond to both set a cookie your client libraries can use to retain the session. Or, this token can be included in a request header as a **Bearer** token.

This Session based authentication is common

4.2. Using JWT

JWT is very popular method as well. You will need to request an issuer token from the necessary service to use JWT. The basic JWT can be extended with necessary information for usage in the different back-end environments.

4.2.1. Variations

No all API compatible systems will use the same authentication methods. For example

- the OpenTHC Compliance Reporting Engine system uses credential pairs and HMACs;
- the OpenTHC Pipe service uses mapped-pass-through credentials dependent on the backend.
- the OpenTHC P2P system uses pre-shared keys for authentication and signing

Refer to the implementation specific documentation for authentication methods

4.3. Ping

The `/auth/ping` endpoint provides a method for a client to check the status of their connexion. It should respond with some type of JSON, which may be dependent on which upstream system is in play.

```
$API_BASE/auth/ping
```

4.4. Shut

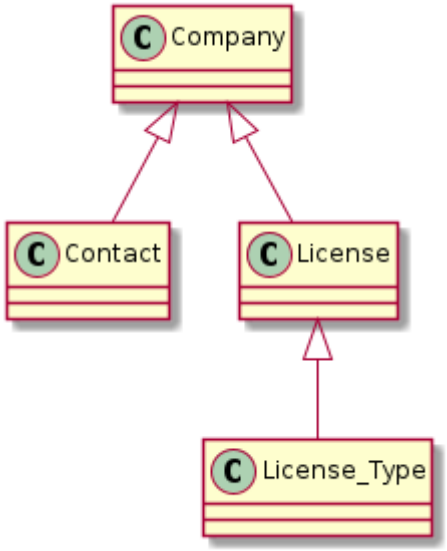
Any request to `/auth/shut` with a session or access token will terminate/revoke this session or token. This request should always respond with an HTTP Staus of 206 for success or an appropriate HTTP Status on error.

```
$API_BASE/auth/shut
```

Chapter 5. Core System Data

5.1. Company

A Company is a container for one or more Contact objects. A Company will have one or more Contacts and one or more Licenses. A **Company** is a container object, which will contain one or more **License** objects, and one or more **Contact** objects.



5.2. License

A **License** is a container for the tracked materials, the license will have a License_Type and is generally tied to a specific physical address.

A license is a unique code assigned by the state governing body, usually numbers and letters, that designates an individual operation of a company. A valid license allows the company to do business in a particular segment of the market (e.g. Cultivation, Manufacturing, or Retail). A company's license must be associated with all tracked actions and transactions by that company, whether they are within the company (a plant or product transition) or outside of it (a B2B or B2C transaction of product). The license is the container for all lots, products, transactions, etc. Every tracked event will be associated with a minimum of one license.

5.3. Contact

A **Contact** is a human, as a member of a Company. A Contact may be a User or an Employee or simply a record for a visitor to a Company/License location. A Contact may authenticate to the system.

5.3.1. Contact User Accounts

Each user is identified by their email address, which must be unique across the system. A standard contact object contains name, phone and email address.

5.3.2. User Account Security

User access to the system is logged. Group ownership is checked on all object access. Each user has an access control list expressed for them.

Chapter 6. Core Config Data

Chapter 7. Products

A Product is an object that represents one unique model of material to be sold. A Product describes the name, package weight/volume and optionally the Variety (Strain.) Each Inventory will be linked to a Product (and to a Variety) to describe the Inventory.

An OpenTHC compatible system should be able to understand Bulk items, Individual (Each) retail items, Packaged Retail items. Specific Product types require Dose/Serving information, including

Package qty: 10 (ea) qom: 0.6 uom: g Serving

7.1. Source Products

A Bulk Product simply specifies the unit-of-measure, the amount of material is stored in the Inventory Quantity The Package is always **1 ea** for a Product of this type. And the Unit is also always **1** with a UOM that will be used to count the Inventory, typically a Weight or Volume unit.

- name = "Bulk Materials"
- package.type = 'bulk'
- package.pack_qom = 1
- package.pack_uom = 'ea'
- package.unit_qom = 1
- package.unit_uom = 'g'

```
{
    "type": "bulk",
    "dose": {}
    "pack": {},
    "unit": {}
}
```

When combined with a Inventory with a Quantity of 2200.000, we would have a single, **2.2 kg** Inventory Lot.

7.2. Retail Products

A Retail Products are packaged, singled, mulitiples and have varying dosage requirements as well. These products can be marked as Each, such as single cans of Coke. Or as a Pack, which is like a case of Coke, with 12 individual units inside.

- name = "Flower Bag"
- package.type = "each"
- package.pack_uom = 1
- package.pack_qom = 'ea'

- package.unit_qom = "3.5"
- package.unit_uom = "g"
- name = "Pre-Roll 1g"
- package_type = "each"
- package_size = "1"
- package_unit = "g"
- name = "Pre-Roll 3 (0.5g each)"
- package_type = "each"
- package_size = 1.5
- package_unit = "g"

7.3. Retail Products - Packaged Singles

A Retail Product is packaged and ready to be shipped, in a Inventory of a specific quantity to another License. These however have a total weight represented in package_size and an individual count in package_each

- name = "Pre-Roll 3 (0.5g each)"
- package_type = "pack"
- package_size = 1.5
- package_unit = "g"
- package_each = 3

package: { type: "bulk|each|pack" size: unit: pack_size: pack_unit: }

package: { type: "bulk" size: 1 unit: "gm", pack_size: null, pack_unit: null, }

package: { type: "each" size: 3.5 unit: "gm", pack_size: null, pack_unit: null, }

package: { type: "pack" unit_qom: 10 unit_uom: "mg", pack_qom: 10, pack_uom: "ea", }

- name = "Sour Mints"
- package_type = "pack"
- package_size = 200
- package_unit = "mg"
- package_each = 20
- name = "Six Pack of 50ml Things (25mg dose)"
- package_type = "pack"
- package_size = 300
- package_unit = "ml"

- package_each = 6

7.4. Retail Products - Packaged Multiple

7.5. Variety (Strain)

Elsewhere they don’t use the word Strain very often — so we don’t either. Internally, we say Variety but to the human-user it may still be labeled as "**Strain**"

Chapter 8. Inventory - Source Material

8.1. Inventory - Source

Source type Inventory are Seeds, Clones, Plants or Plant Tissue (although the last one is rarely used).

Chapter 9. Plants & Crops

Plants are growing items in the System

9.1. Plants :: Create

Creating new Plants from either Clones or Seeds or other allowed Inventory Lot types.

9.1.1. Parameters

- Source: the Source Inventory Identifier
- Variety: Variety (Strain) Name
- Batch: Some systems operate with a Batch concept, which could be provided here
- Stage: Descriptive text of the Stage of the plant
- Planted: Date Planted, ISO Date format
- Section: The Section the Plant is located in

```
curl -X POST $API_BASE/plant

{
    license: {
        id: "ABC123"
    },
    source: {
        id: "I1A",
    },
    variety: {
        id: "S2B"
    },
}

{
    id: "P3C"
    variety: {
        name: "Alpha"
        id: "S2B"
    }
}
```

9.2. Plants :: Modify

Change either the Batch, Variety, Stage, Section, Planting Date, Mother Designation or other regulatory system defined attributes. When present, attributes will follow the OpenTHC JSON Schema. An implementation is free to extend these attributes.

```
curl -X POST $API_BASE/plant/$ID

{
    "variety": {
        "id": "S3C"
    }
    "section": {
        "id": "Z4D"
    }
}
```

9.3. Plants :: Delete

Marking a Plant as Deleted is the method to mark or confirm destruction of plant material.

If the compliance engine requires confirmation then a DELETE method is sent once to mark as scheduled for removal, and a second DELETE request to confirm. The first request responds with a `410` and the confirmation request which responds with a `423`, empty body.

```
curl -X DELETE $API_BASE/plant/$ID

{
    "meta" {
        "detail": "Requires Confirmation"
    }
}
```

And then send the second request

```
curl -X DELETE $API_BASE/plant/$ID

{}
```

9.4. Growth

Add documentation about feed, fertilizer, nutrients, pesticides and other things added to or on the plants.

9.5. Plants :: Grow

Additives record the application of some material to the plants, including pesticies and nutrients.

9.5.1. Apply Grow Materials

9.5.2. Attach Grow Notes to Multiple Plants

9.6. Plants :: Notes

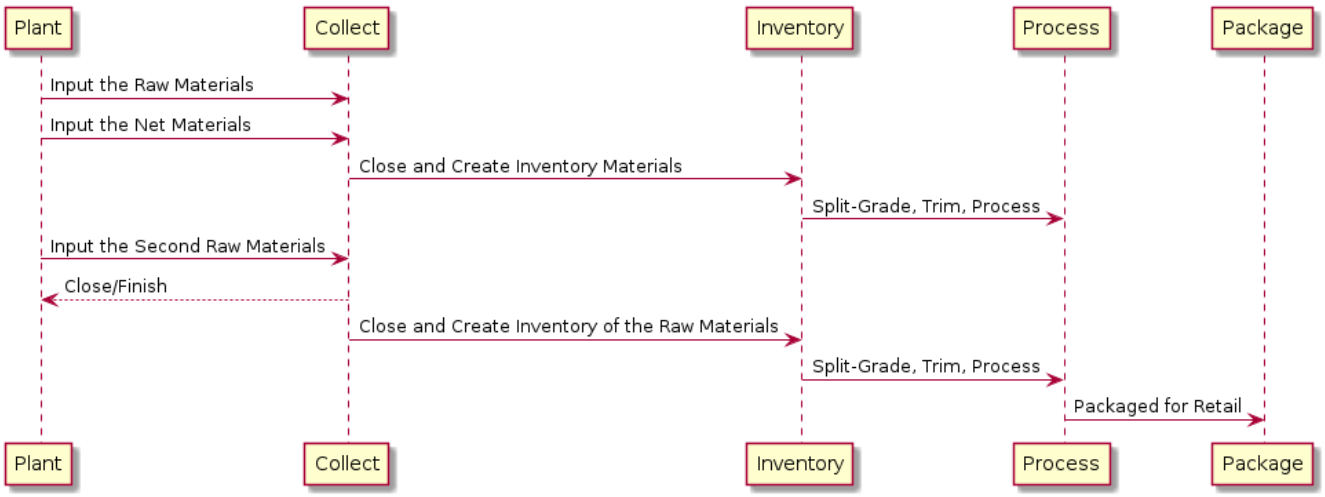
Notes on the plants record the application of nutrieinets, pesticides and other matter. Farmers may also use the Note field to attach comments or photos to the records.

9.6.1. Create a Plant Note

9.6.2. Attach Note to multiple Plants

9.7. Collecting Materials

A Raw Collection, sometimes called a Harvest, Manicure or Wet Weight, is the process of taking raw materials from the crop. A Net Collection, sometimes called a Cure or Dry Weight, is the process of recording the amount of Raw materials to continue for production processing.

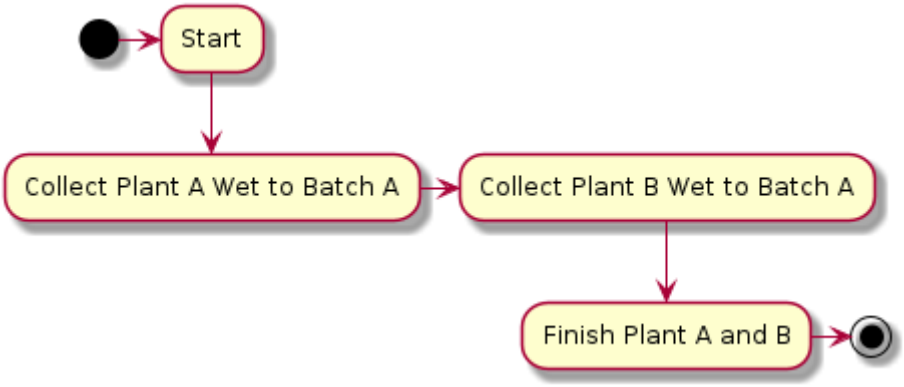


9.8. Plants :: Raw Collect

Raw materials collection is also known as *Harvesting* or *Manicuring*. A **Raw Collection** may be a collection from the entire plant, or a portion.

From Plants one or more **Raw Collections** can be made. A raw collection is to enter materials that have been directly collected from the plants.

Once the process is complete, as determined by farmer, this harvest bundle is closed



9.9. Plants :: Net Collect

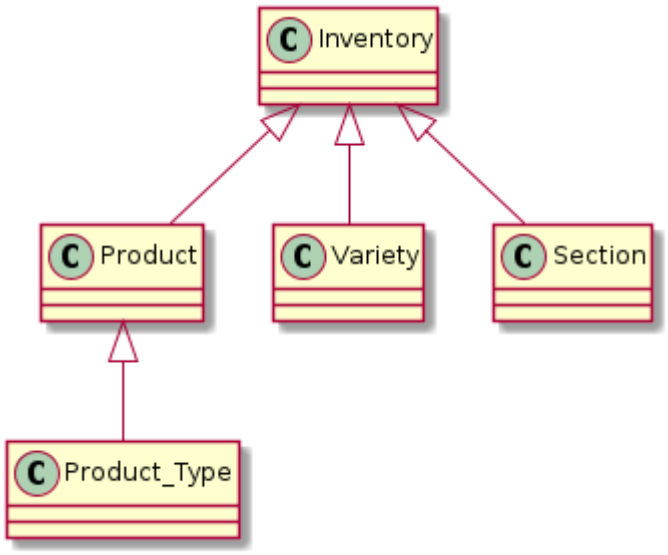
Net materials collection is also known as *Curing* or *Dry Weight*. These materials are sub-set of the raw materials that will enter the production pipeline.

The **Net Collection** process operates on the group of plants from a **Raw Collection** process. When entering a Net weight the collection group will be closed and new bulk Inventory will be created.

This call can be repeated for each type of material collected.

Chapter 10. Inventory - Bulk Material

Inventorys, sometimes called Lots, represent all Source, Product, Processing and Retail materials



10.1. Listing Inventory

```
curl $API_BASE/inventory
```

10.2. Inventory / Adjust

A regulatory system specific type of adjustment to the inventory, generally requires a note.

```
curl -X PATCH $API_BASE/inventory/{OID}

{
    qty: 55,
    code: 'audit',
    note: 'mis-count in processing'
}
```

10.3. Inventory / Modify

Only Permitted Modifications will be Allowed For Modification of Weight or Volume requires docuemntation

```
curl -X PATCH $API_BASE/inventory/{OID}

{
    qty: 55,
    code: 'audit',
    note: 'mis-count in processing'
}
```

10.3.1. Inventory / Move

For Moving Inventory to a New Section (aka Area, Room, Zone)

10.4. Inventory / Convert

The process of taking one or more Source lots and converting into one Output lot.

```
curl -X POST $API_BASE/inventory --data-binary <-
{
    source: [
        {
            "id": $ID_A,
            "qty": 900,
        },
        {
            "id": $ID_B
            "qty": 100,
        }
    ],
    output: {
        product: {
            id: $PRODUCT_ID
        }
        qty: 1000
    }
}
```

This will record the removal from each of the indicated source items and record the linkage to the single output item.

10.5. Inventory / Split

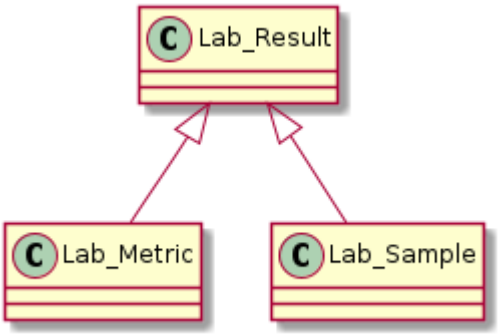
Slice off a portion of an existing inventory, also known as Sub-Lotting. Send a POST similar to Create a Inventory but do not include an output product type. Only one Source is permitted.

```
curl -X POST $API_BASE/inventory --data-binary <-
{
    source: $SOURCE_ID
    output: {
        qty: 500
    }
}
```

Chapter 11. Laboratory / Quality Assurance

Create a Lab Sample from a product, provide Lab Result that contain Lab Result Metrics

All Lab Metric **qom** fields should be reported with four decimal places of precision, regardless of **uom**



11.1. Sample :: Create

From an Inventory create a Lab Sample, which is a special type of sub-lot from the primary Inventory Lot. This Sample item will have a unique identifier and a child relationship to the source.

```
curl /inventory/{ID}/sample -X POST -d '
{
    type: "Lab",
    qty: "5"
}'

HTTP 201 Created
{
    "meta": {},
    "data": {}
}
```

11.2. Sample :: Detail

Return the details of the Lab Sample, including which tests are required/requested. Similar to **requiredlabtestbatches** API call in METRC.

```
curl /lab/sample/{ID}

HTTP 200 OK
{
    "meta": {},
    "data": {}
}
```

11.3. Sample :: Destroy

A Sample is Destroyed by the Laboratory when they have finished sampling the materials. Or, in the case where a supplier no longer wants the test, the material should be destroyed. If the material is being returned to the supplier, one should use Void

```
curl -X DELETE /lab/sample/{ID}

{}
```

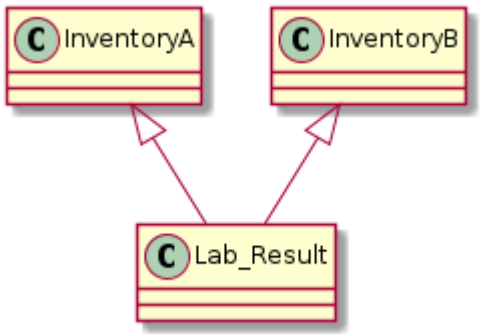
11.4. Sample :: Void

If the sample is no longer valid and the material is being returned to the supplier, use Void. Then transfer the sample identifier (by ID) back to the origin license.

11.5. Result :: Create

Generally the Laboratory (or sometimes the Licensed Operator) will update the Lab results in the system. Either through the WebUI or via API.

A Lab Result can be attached to one, or more Inventory objects.



Values are sent in the

```
curl -X POST lab/result

{
    "status": "pass",
    "metric": [
        {
            "id": "018NY6XC00LM49CV7QP9KM9QH9",
            "type": "potency",
            "name": "THC",
            "qom": 12.3456
            "uom": "percent",
            "lod": 0.1234,
            "loq": 0.1234
        }
    ]
}
```

The "qom" field values are always expressed as floating point numbers, with four decimal points of precision, eg: 12.3456

Percent values are expressed as values between 0 and 100, values outside of that range may be silently rejected

Pesticides should include their CAS identifier and be reported in parts-per-billion or PPB.

11.6. Result :: Void

Generally the Laboratory (or sometimes the Licensed Operator) will need to remove the the Lab Result. The **DELETE** verb will accomplish this — but it must be called twice.

```
curl -X DELETE /lab/result/{ID}

HTTP 248 Something
{
    "meta": {
        "detail": "Call Delete again to confirm"
    }

}
```

Chapter 12. B2B Transaction

A B2B Transaction is prepared in two parts. The B2B Outoing Transaction (aka: Manifest, Transfer) is prepared to indicate the Transfer of materials from one license holder to another.

12.1. Outgoing

12.1.1. B2B Transaction / Transfer Outgoing / Create

12.1.2. B2B Transaction Commit

Once the Outgoing Transfer has been configured properly, with Target License and Tranfer Line Items it may be committed. Once Commited the Transfer is available for the receiver to accept. ==== B2B Transaction / Transfer / Outgoing / Update

12.2. Incoming

B2B Incoming Transfers is the process of receiving a request, processing the materials into the target License inventory. Typically in a regulated environment, the supply-side actor will complete a B2B Outgoing and the demand-side actor will file a corrisoponding B2B Incoming.

12.2.1. B2B Transaction / Transfer / Incoming / Accept

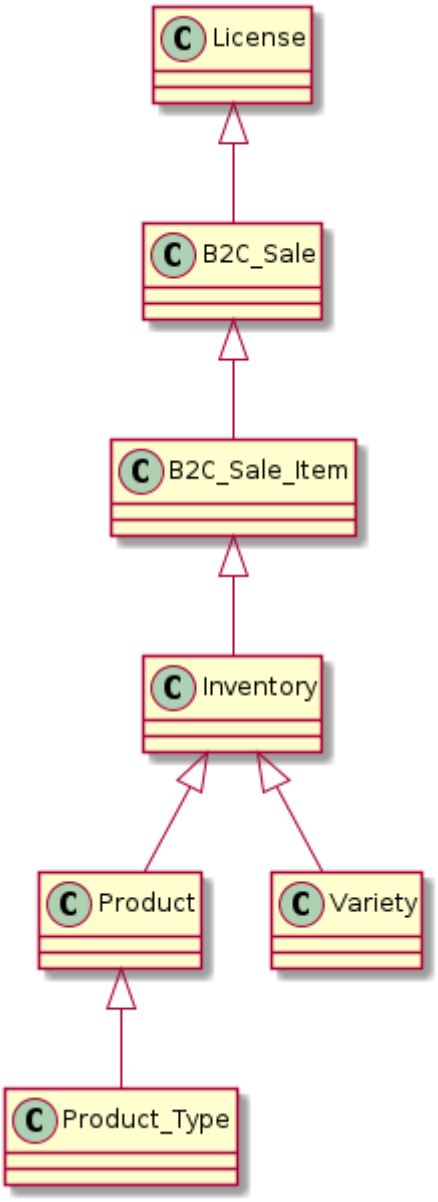
Chapter 13. Retail Sales

Chapter 14. B2C Transaction

A transaction selling one or more items to a retail customer. This customer may or may not be tied to a specific Contact (or a Generic Contact such as "walk-in")

14.1. B2C Transaction Item

Each line-item, and it’s tied back to the B2C Transaction as well as the specific Inventory Lot.



Chapter 15. Adminstration

All the Core data can be manipulated via the API.

15.1. Permissions

Chapter 16. Access Control

We're using RBAC and building on the Casbin library.

16.1. Grow Materials

Inputs for grow supplies; adding a bulk item, with cost and the removing portions.

Inputs for grow journals; adding a note and a metric to a plant (or group of plants, but tracked per-plant)

Chapter 17. Customization / Extending

If it's a really good idea please consider a pull request.

Additionally, the JSON can be extended, without affecting the base. The addition of an **x-[vendor]** attribute to a JSON model should suffice. This is shown in some of the examples.