PEFTrust Public API

Main informations

Environment	Version	Base url
Demo (test)	5.8.0	https://api-demo.peftrust.com
Production	5.8.0	https://api.peftrust.com

Postman collection

The PEFTrust Public API Postman collection is a perfect way to understand how the api works and try it with specific example.

Download the collection

Note that the collection has variables (user, password etc) set at the collection level with a login script to avoid repeated logins.

Authentication

First of all, the API is protected with a JWT Bearer that must be set in Authorization header.

Method	Entity	Endpoint	Description
GET	Authentication	/auth/login	Will return a temporary JWT token to handle authentication

Entities endpoints

As an API Consumer you can perform CRUD operation on some entities related to your organization account.

Method	Entity	Endpoint	Description
GET	Evaluation category	/evaluations/categories	Fetch all evaluation categories (as a tree)
GET	Evaluation category	/evaluations/categories/:id	Fetch one specific evaluation category
GET	Evaluation	/evaluations	Fetch all evaluations
POST	Evaluation	/evaluations	Create an evaluation

Method	Entity	Endpoint	Description
GET	Evaluation	/evaluations/:id	Fetch one specific evaluation
PATCH	Evaluation	/evaluations/publish/:id	Publish a specific evaluation reference (create a version)
GET	Evaluation	/evaluations/schema/:id	Fetch the current payload schema of a given evaluation
PUT	Evaluation	/evaluations/:id	Update a specific evaluation
PATCH	Evaluation	/evaluations/:id	Update a specific evaluation with atomic operation
DELETE	Evaluation	/evaluations/:id	Delete a specific evaluation
GET	Material	/materials	Fetch all materials
GET	Material	/materials/:id	Fetch one material
GET	Teams	/teams	Fetch all teams
GET	Tag	/tags	Fetch all tags
GET	Accessory	/accessories	Fetch all accessories
GET	Accessory	/accessories/:id	Fetch one accessory
GET	Mask	/masks	Fetch all masks
GET	Mask	/masks/:id	Fetch one mask
POST	Mask	/masks	Create a mask
PUT	Mask	/masks/:id	Update a mask
DELETE	Mask	/masks/:id	Delete a mask
GET	Product	/products	Fetch all products
GET	Product	/products/:id	Fetch one product
POST	Product	/products	Create a product
PUT	Product	/products/:id	Update a product
DELETE	Product	/products/:id	Delete a product
GET	Product packaging	/products/packagings	Fetch all product packagings
GET	Product packaging	/products/packagings/:id	Fetch one product packaging
POST	Product packaging	/products/packagings	Create a product packaging

Method	Entity	Endpoint	Description
PUT	Product packaging	/products/packagings/:id	Update a product packaging
DELETE	Product packaging	/products/packagings/:id	Delete a product packaging

Schema endpoints

Schema endpoints provide a readable and complete JSON schema that describes all of the related fields for a given entity (underlying parameters, lifecycles, processes etc.). These endpoints are absolutely necessary when it comes to build a payload in order to create an evaluation.

Every categories and materials have their own information groups, processes and last but not least their own parameters. Those parameters could have the same label, functional effect etc. but their ids are different because they are not related to the same entity and they could diverge over time.

Note that the returning schema contains some properties that are not necessary for the evaluation create or update payload but they try to provide the most understandable schema with de aim of create it.

Method	Entity	Endpoint	Description
GET	Evaluation category schema	/evaluations/categories/schema/:id	Fetch schema for a specific evaluation category
GET	Material schema	/materials/schema/:id	Fetch schema for a specific material
GET	Mask schema	/masks/schema	Fetch schema for a mask (they all have the same schema)
GET	Product packaging schema	/products/packagings/schema	Fetch schema for a product packaging (they all have the same schema)

Results endpoint for Product and Evaluation

An evaluation is not evaluated once it is created, therefore there are some endpoints to perform this operation and fetch related results. For products, the results are available once the product is created as it is a coumpound entity that contains all of the related precalculated evaluations.

Method	Entity	Endpoint	Description
GET	Evaluation evaluate	/evaluation/evaluate/:id	Evaluate the evaluation

Method	Entity	Endpoint	Description
GET	Evaluation evaluate full	/evaluation/evaluate/:id/full	Evaluate the evaluation (standard + exobalyse)
GET	Evaluation evaluate ecobalyse	/evaluation/evaluate/:id/ecobalyse	Evaluate the evaluation with Ecobalyse API
GET	Evaluation check Ecobalyse availability	/evaluation/check/:id/ecobalyse	Check whether the evaluation can be evaluated with Ecobalyse API or not
GET	Evaluation results	/results/evaluation/:id	Get evaluation results
GET	Evaluation results IPC	/results/evaluation/:id/ipc	Get IPC results
GET	Evaluation results full	/results/evaluation/:id/full	Get enhanced evaluation results
GET	Evaluation results ecobalyse	/results/evaluation/:id/ecobalyse	Get evaluation ecobalyse results
GET	Product results IPC	/results/evaluation/:id/ipc	Get IPC results
GET	Product results full	/results/evaluation/:id/full	Get enhanced product results
GET	Product results ecobalyse	/results/evaluation/:id/ecobalyse	Get evaluation ecobalyse results

Asynchronous results using webhooks

While using evaluate endpoints you can specify 3 optional parameters :

- async: will not await for the result and confirm processing
- webhook: specify an url where we will send the result when it's ready (the results will be sent using `POST` method)
- webhook_token: specify a token to secure the webhook call, the token will be pass in `X-Auth-Token` header (you've to implement the logic to secure your application)

Notice that async and webhook_token are required why specifying webhook url.

Webhook payload

evaluate endpoints : /evaluation/evaluate/:id

```
{
  "full": {
    // Same content as /results/evaluation/:id/full
  },
  "ipc": {
    // Same content as /results/evaluation/:id
  }
}
```

evaluate endpoints : /evaluation/evaluate/:id/full

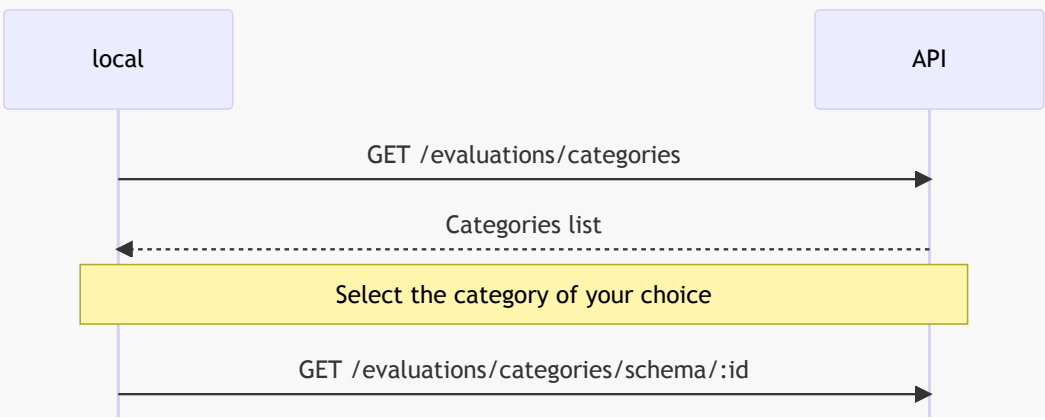
```
{
  "full": {
    // Same content as /results/evaluation/:id/full
  },
  "ipc": {
    // Same content as /results/evaluation/:id
  },
  "ecobalyse": {
    // Same content as /results/evaluation/:id/ecobalyse
  }
}
```

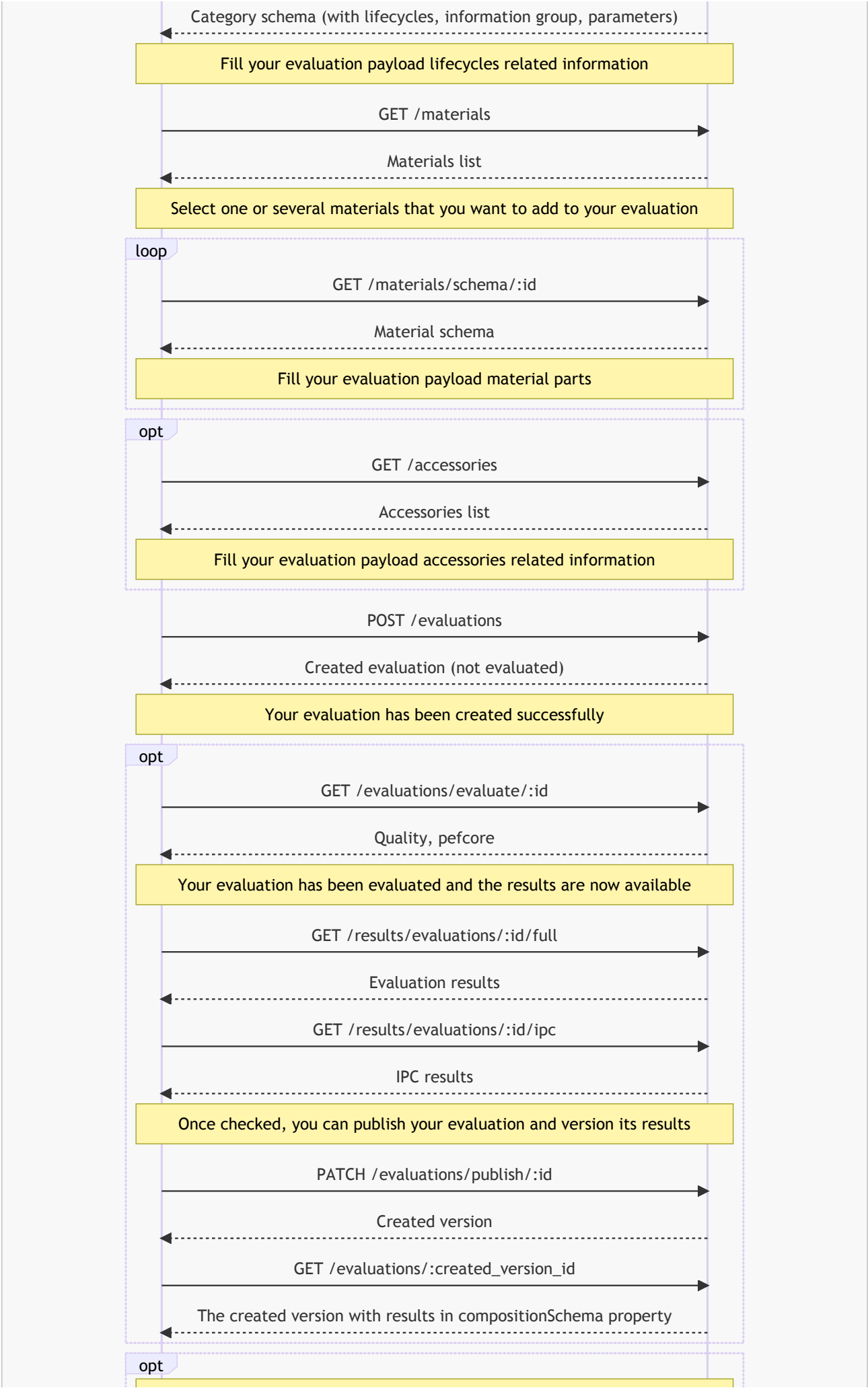
evaluate endpoints : /evaluation/evaluate/:id/ecobalyse

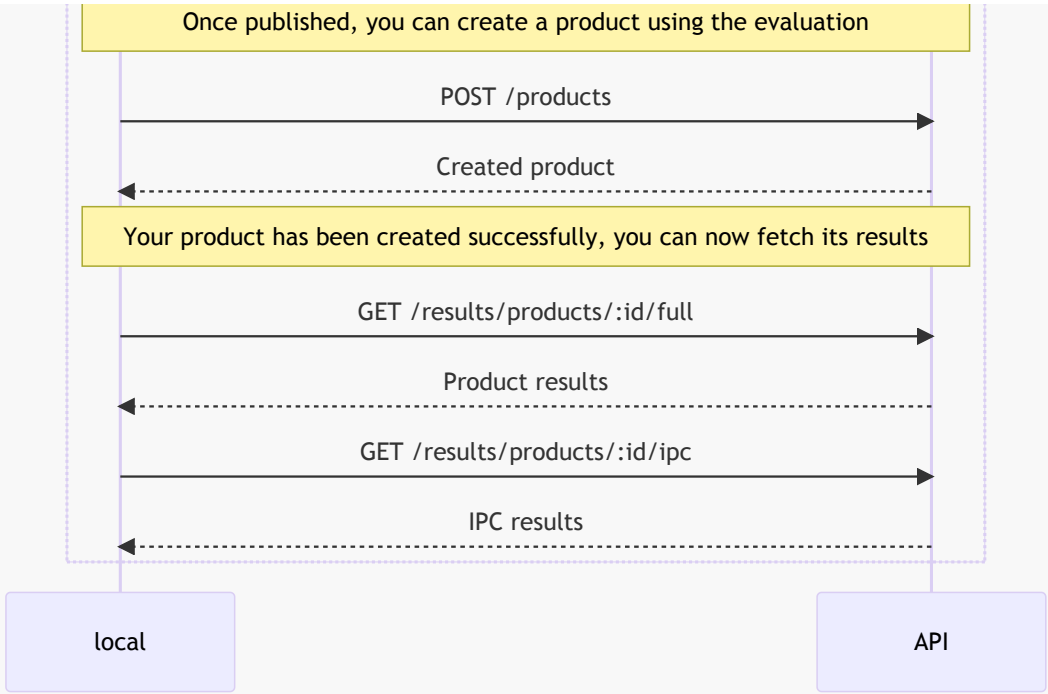
```
{
  // Same content as /results/evaluation/:id/ecobalyse
}
```

Evaluation & Product creation process

Sequence diagram




Category schema (with lifecycles, information group, parameters)
Fill your evaluation payload lifecycles related information
GET /materials
Materials list
Select one or several materials that you want to add to your evaluation
loop
GET /materials/schema/:id
Material schema
Fill your evaluation payload material parts
opt
GET /accessories
Accessories list
Fill your evaluation payload accessories related information
POST /evaluations
Created evaluation (not evaluated)
Your evaluation has been created successfully
opt
GET /evaluations/evaluate/:id
Quality, pefcore
Your evaluation has been evaluated and the results are now available
GET /results/evaluations/:id/full
Evaluation results
GET /results/evaluations/:id/ipc
IPC results
Once checked, you can publish your evaluation and version its results
PATCH /evaluations/publish/:id
Created version
GET /evaluations/:created_version_id
The created version with results in compositionSchema property
opt



Parameters

Parameters have different kind of type that come with their own implicit formating rules and constraints. They are formatted as following in the payload:

```
{
  "id": "{{id}}",
  "value": "{{value}}",
}
```

Types

Type	Expected format
text	Must be a string
number	Must be a number
email	Must be an email
date	Must be a date
checkbox	Must be a boolean
select	Must be one of the allowed values
select-quantity	Must be one of the allowed values separated by a dash "/" and a quantity as a string number
select-multiple	Must be one or several allowed values eparated by a coma ","

Select parameters

Since v5.7.9, slug as been added to the schema allowed values and you can use another property than the id as value. It works with select, select-quantity and select-multiple types.

For example, here is a schema for a given select:

```
{
    "id": "7990de5c-7a29-476e-b801-9517b0cab0e4",
    "name": "Size",
    "identifier": "size",
    "fieldType": "select",
    "qualityCategoryId": "d6fee280-0610-4cb1-95e4-ed2bcdfa6938",
    "defaultValue": "0",
    "allowedValues": [
        {
            "name": " 50 - Men",
            "value": "50 - Men",
            "slug": "50_men"
        }
        // ...
    ],
    "constraints": {},
    "customizable": false
}
```

Instead of sending the raw value you can send the slug or even the name specifying useProperty:

```
{
  "id": "7990de5c-7a29-476e-b801-9517b0cab0e4",
  "useProperty": "slug",
  "value": "50_men"
}
```

Another example with a location select parameter:

```
{
    "id": "100cee5c-59fb-4bb2-bd4c-0e5927e72c23",
    "name": "Location",
    "identifier": "location",
    "fieldType": "select",
    "qualityCategoryId": "905adc86-a0fb-4e9c-aea5-af72eef9c62a",
    "defaultValue": "5a5446dc-b05f-47d4-8604-02040bd31e9f",
    "allowedValues": [
        {
            "name": "Africa",
            "value": "e25469ee-e4db-4215-adbb-d6e12658fd22",
            "slug": "africa",
```

```
            "code": "002",
            "code_alpha_2": null,
            "code_alpha_3": null
        },
        // ...
    ],
    "constraints": {},
    "customizable": false
}
```

You can use any of the allowedValue object property such as `code` for example:

```
{
  "id": "100cee5c-59fb-4bb2-bd4c-0e5927e72c23",
  "useProperty": "code",
  "value": "002"
}
```

Constraints

Type	Expected format
required	This parameter is required (must be filled if it has no default value)
min	Minimum value (for number type)
max	Maximum value (for number type)
pattern	RegEx that must match with the filled value
minDate	Minimum date
maxDate	Maximum date
integer	Must be an integer (!= float)

Custom parameters

Parameters flagged as `custumizable` come with nested parameters in their property `customParameters`. In that case you could either:

- Fill its value as usual and just ignore this information
- Fill its value with the keyword "custom" and complete all of its `customParameters`

These parameters allow user to be more specific and complete its own values that make the API able to calculate these parameters impacts much more accurately than pre-completed values.

Payload format

The payload should be an application/json object which contains the following keys at root :

Name	Type	Mandatory	Description
category.id	string	YES	The evaluation category id
mask.id	string	NO	The optional mask to apply on the evaluation
packaging.id	string	NO	The optional packaging apply to the evaluation
teams.id	string	YES	The evaluation teams list (with their ids)
tags.id	string	NO	The evaluation tags (with their ids)
lifecycles	array	YES	Array of lifecycles that contains their parameters
composition.parts	array	YES	Array of evaluation parts that contains their materials per percentage
composition.accessories	object	NO	Object that contains accessories related informations

Every parameters that are not specified in the payload will be filled with their default value.

Using weight as base unit

Since version 1.9.8 it's possible to use weight instead of percentage if you want to.

To do so, you just have to change composition baseUnit to "weight" instead of percentage and specify weight everywhere you put percentage before. Then, the API will ignore percentage key and calculate it by itself also the API will take care that the calculated percentage of your evaluation match the requirements otherwise you'll get related error messages.

Update entity

If you want to update an entity, you have to use the PUT method and provide the entity id in the URL. This method will replace the entity with the new one, you need to provide all the fields even if they are not modified.

Since version 5.6.0 you can update an entity with atomic operation, you have to use the PATCH method and provide the entity id in the URL. This currently only works for evaluation entity and will update only the fields that are provided in the payload.

The documentation about the payload format and operations is available in the documentation below :

- Operations: https://www.npmjs.com/package/json-merger
- Queries (for $match): https://www.npmjs.com/package/jsonpath

Allowed operations are :

- $merge
- $remove

- $replace
- $concat
- $combine
- $append
- $prepend
- $insert
- $match
- $move
- $select
- $repeat

Notice that any FS operation are disallowed ($import, $include etc) and the top operation is always $merge but you don't have to specify the source.

Show the Postman collection for more details about the payload format an example with multiple atomic operations is available in the collection.