Taxonomy

2025-03-14

# The Core Idea Behind Taxonomies in Open Food Facts

Open Food Facts organizes its products and ingredients into **taxonomies**. A taxonomy is essentially a collection of category names (in multiple languages) connected by parent–child relationships. In other words:

* **A category “A” may have children**: these are subcategories more specific than A.
* **A category “A” may have one or more parents**: the category can inherit properties from multiple parents, if it belongs to multiple higher-level categories. For instance, *Whole black olives* may be both a child of *Black olives* and *Whole olives*.
* **No cycles allowed**: Taxonomies are structured as a **Directed Acyclic Graph (DAG)**—meaning you cannot loop back to a previously defined category once you move forward.

[Taxonomies](https://wiki.openfoodfacts.org/Global_taxonomies) form the backbone of data organization in Open Food Facts.  
They are stored in raw text files (.txt) that primarily include translations, classifications, labels, ingredient lists, and hierarchies.

**Their functionality can be summarized in two core aspects**:

1. **Classification**  
   With taxonomies, we can establish hierarchies (or DAG structures) for food products. For example, consider the product [Nutella](https://world.openfoodfacts.org/product/8000500310427/biscuits-croquants-au-coeur-onctueux-de-nutella). It has category labels such as *Snacks, Sweet snacks, Biscuits and cakes, Biscuits and crackers, Biscuits, Filled biscuits*. These labels are organized in a hierarchy:
   * *Snacks* → *Sweet snacks* → *Biscuits and cakes* → *Biscuits and crackers* → *Biscuits* → *Filled biscuits*  
     This means it is first recognized as a “Snack,” more specifically a “Sweet snack,” and so on, until we reach “Filled biscuits.”
2. **Translation**  
   Taxonomies also help maintain translations for ingredients, measurement units, nutrients, and more. Local variations, alternate spellings, and synonyms can make direct machine translation inaccurate. Having a central, curated system solves that problem.

Open Food Facts has separate files for different taxonomies (e.g., categories.txt, ingredients.txt, labels.txt, etc.). Each file follows the same basic structure described below.

# Main Components Found in a Taxonomy File

Let’s consider categories.txt as our example. This file (also available [here](https://github.com/openfoodfacts/openfoodfacts-server/blob/main/taxonomies/food/categories.txt)) defines the product categories: it lays out parent–child relationships, translations, synonyms, and more.

1. **Category Names**  
   Typically, you’ll see lines like:

* en: Baby foods  
  fr: Aliments pour bébé  
  it: Alimenti per bambini
* This means the canonical category name is Baby foods (in English), and there are translations in French and Italian.

1. **Parent–Child Relationships**  
   For instance:

* < en:Baby foods  
  en: Cereals for babies
* The notation < en:Baby foods indicates that the **next category** (en: Cereals for babies) is a **child** of Baby foods. Because multiple parents can exist, you may also find something like:
* < en:Black olives  
  < en:Whole olives  
  en: Whole black olives
* which shows that *Whole black olives* has **two parents**: *Black olives* and *Whole olives*.

1. **Synonyms**  
   Lines starting with synonyms:en: (or similar for other languages) provide equivalent names. For example:

* synonyms:fr: poulet-crudités, crudités-poulet
* means both “poulet-crudités” and “crudités-poulet” are recognized as the same category. **Recursion** applies here: if “Yoghurt” is a synonym of “Yogurt,” then “Banana yoghurt” automatically becomes a recognized synonym of “Banana yogurt.”

1. **Stopwords**  
   Sometimes you’ll see lines such as:

* stopwords:fr: le, la, du, à
* These are words that get ignored (for example, articles and short filler words). They don’t affect the structure but help match categories more effectively (e.g., you don’t have to list every preposition variant explicitly).

1. **Additional Properties**  
   Lines like expected\_nutriscore\_grade:en:c or ignore\_energy\_calculated\_error:en:yes define **properties** for each term. These can refine how products are analyzed (e.g., Nutri-Score).  
   For a more comprehensive list of recognized properties, check the [Taxonomy Properties](https://wiki.openfoodfacts.org/Taxonomy_Properties) page on the Open Food Facts wiki.

# Typical Organization: A Concrete Example

Look at this short (simplified) snippet:

< en:Baby foods  
en: Cereals for babies  
  
< en:Baby foods  
en: Snacks and desserts for babies  
  
< en:Snacks and desserts for babies  
en: Biscuits for babies

* < en:Baby foods tells us that categories defined right after it (until we see a new < en: line) are children of **Baby foods**.
* Then en: Cereals for babies creates the new category under Baby foods.
* We repeat the logic:
  + < en:Baby foods → parent is Baby foods
  + en: Snacks and desserts for babies → child is Snacks and desserts for babies
* Lastly, < en:Snacks and desserts for babies shows a child: en: Biscuits for babies.

We end up with:

Baby foods  
 ├─ Cereals for babies  
 └─ Snacks and desserts for babies  
 └─ Biscuits for babies

You can visualize it as a **directed tree** (or more broadly, a DAG) where each arrow points from parent to child.

# How to Read and Interpret categories.txt

Within the file, each block may include:

* **Comments** (start with #)
* **Property lines** (e.g., incompatible\_with:en:categories:en:...)
* **Synonym lines** (e.g., synonyms:fr:…)
* **Stopword lines** (e.g., stopwords:fr:…)
* **Parent definitions**:
  + For example, < en:ParentCategory (and possibly multiple parents, each line starting with < en:).
* **Child definitions**:
  + en: SomeChild
  + fr: UnEnfant
  + (and so on for other languages)

Whenever you see:

< en:ParentCategory  
< en:AnotherParent  
en: ChildCategory  
...

it means “the category ChildCategory is a child of both ParentCategory and AnotherParent.”

# Example of Python Code to Build a Parent–Child Graph

Below is an **illustrative** (not fully production-ready) script to parse categories.txt:

1. **Read** the file line by line.
2. **Identify** lines that define parent–child relationships.
3. **Store** these relationships in a graph structure (using [NetworkX](https://networkx.org/)).
4. **Draw** a small graph using matplotlib.

**Note:** The actual categories.txt is very long and more complex. This code assumes a simplistic approach: each time we see < en:XYZ, the **next** line starting with en: is treated as a child. In practice, you might need to handle translations, synonyms, or multiple parents more carefully.

import networkx as nx  
import matplotlib.pyplot as plt  
  
def build\_taxonomy\_graph(file\_path):  
 G = nx.DiGraph()  
 parent\_categories = []  
  
 with open(file\_path, "r", encoding="utf-8") as f:  
 for line in f:  
 line = line.strip()  
 # Ignore empty lines or comment lines  
 if not line or line.startswith("#"):  
 continue  
  
 # If line starts with "< en:", capture the parent name  
 if line.startswith("< en:"):  
 parent\_name = line[len("< en:"):].strip()  
 parent\_categories.append(parent\_name)  
  
 # If line starts with "en:", treat it as a child  
 elif line.startswith("en:"):  
 child\_name = line[len("en:"):].strip()  
  
 # For simplicity, link the child to the most recently declared parent  
 if parent\_categories:  
 current\_parent = parent\_categories[-1]  
 G.add\_node(current\_parent)  
 G.add\_node(child\_name)  
 G.add\_edge(current\_parent, child\_name)  
 else:  
 # If no parent is set, this might be a top-level category  
 G.add\_node(child\_name)  
  
 else:  
 # We ignore synonyms, stopwords, or other properties in this simple script  
 pass  
  
 return G  
  
def main():  
 file\_path = "categories.txt" # Path to your file  
 taxonomy\_graph = build\_taxonomy\_graph(file\_path)  
  
 # Quick drawing of the graph  
 plt.figure(figsize=(12, 8))  
 pos = nx.spring\_layout(taxonomy\_graph, k=0.5, seed=42)  
 nx.draw(taxonomy\_graph, pos, with\_labels=True, node\_size=3000, font\_size=8, arrows=True)  
 plt.title("Example of a parent–child graph from categories.txt")  
 plt.show()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## Code Explanation

1. **Imports**
   * networkx for the directed graph.
   * matplotlib.pyplot for visualization.
2. **build\_taxonomy\_graph**
   * Reads the file line by line, skips comments/empty lines.
   * Identifies parent definitions (< en:Name) and child definitions (en:Name).
   * Creates a directed edge from the **last-seen parent** to the child.
   * Note that to properly handle *multiple parents*, you would want to store **all** parents seen before the next en: line, rather than just the latest one.
3. **Graph Display**
   * Uses spring\_layout for automatic node placement.
   * Draws labels and arrows with nx.draw.

# Contribution and Further References

1. **Community Maintenance**: Anyone can contribute to these taxonomy files by adding or correcting translations, parents, and synonyms. The [Taxonomy Editor UI](https://ui.taxonomy.openfoodfacts.org/) is available to simplify editing.
2. **Taxonomy Properties**: For a full list of properties and how they are applied, see the [Taxonomy Properties page](https://wiki.openfoodfacts.org/Taxonomy_Properties).
3. **Slack Discussion**: The #taxonomies channel on Open Food Facts’ Slack is a good place to ask questions, discuss ideas, or propose improvements.
4. **Multiple Files**: Open Food Facts manages different taxonomy files for ingredients, labels, allergens, packaging, and more. All are stored in the [taxonomies folder on GitHub](https://github.com/openfoodfacts/openfoodfacts-server/tree/main/taxonomies).
5. **No Cycles, Possibly Multiple Parents**: The structure must remain acyclic, but each term can have more than one parent if needed, reflecting real-world overlaps (e.g., “Whole black olives” can be both “Whole olives” and “Black olives”).