PUT / GET /v1/product-categories

Simplified Business Specification (Hackathon Version)

---

1. Introduction

1.1 Public Overview

The purpose of this endpoint is to receive model product category data from the calling process API and map and transform the data as per defined business logic to determine whether product category records should be **created**, **updated**, **deleted**, or **retrieved**.

This specification represents a **simplified implementation** of the full Product Category synchronization process, retaining original business rules and transformation logic while limiting scope for a hackathon use case.

1.2 Target Customers

The system API will be triggered by a process API responsible for managing Model MDA data.

1.3 Trigger

These endpoints are triggered via real-time HTTPS requests from the calling process API.

1.4 Execution Frequency

Real-time.

1.5 Caching Requirements

N/A

---

2. Business Logic

Once the PUT endpoint is triggered, the following business logic is executed.

2.1 Business Keys

Each product category record is uniquely identified using the following business keys:

- productLine
- productCategoryCode
- productCategoryGroupCode

2.2 Fetch Existing Product Category Records

The system evaluates whether an existing product category record is present for the provided business keys.

For this simplified implementation, existing data is represented logically and may be stored in-memory or mocked for comparison purposes.

2.3 Compare Incoming and Existing Data

Incoming product category data is compared against existing data using the following attributes:

- productCategoryName
- doorType
- cylinderCount
- motivePower
- productCategoryGroupName

2.4 Determine Operation

The operation is determined based on the comparison results and delete indicator.

Insert Scenario

Perform **Insert** if:

- No existing record is found for the given business keys.

Action Code: `A`

Update Scenario

Perform **Update** if:

- Existing record is found, and
- One or more attributes differ, and
- businessKeyDeleteIndicator = `"N"`.

Action Code: `C`

Delete Scenario

Perform **Delete** if:

- Existing record is found, and
- businessKeyDeleteIndicator = `"Y"`.

Action Code: `D`

No-Operation Scenario

If:

- Existing record is found,
- No attributes have changed, and
- businessKeyDeleteIndicator = "N",

then no operation is performed.

---

3. Interface Details

3.1 PUT /v1/product-categories

Receives model product category data and applies transformation logic to determine whether records should be inserted, updated, or deleted.

PUT is used to represent both create and update semantics based on business rules.

3.2 GET /v1/product-categories

Retrieves product category data for the provided business keys.
This endpoint is used to verify records created or updated via the PUT operation.

3.3 Headers

Standard headers are expected, including but not limited to:

- consumerId
- consumerName
- correlationId
- callerName

- language

Header validation is assumed to be handled upstream.

---

4. Transformation Logic

4.1 Field-Level Transformation Logic

Column Name	Sample Value	Transformation Logic
actionCode	A / C / D	Derived from operation determination logic. A = Insert, C = Update, D = Delete
productLine	A	Mapped directly from incoming request payload
productCategoryCode	200	Mapped directly from incoming request payload
productCategoryName	Sedan	Populated for Insert and Update scenarios only
productCategoryGroupCode	SG1	Mapped directly from incoming request payload
doorType	4D	Populated for Insert and Update scenarios
cylinderCount	4	Populated for Insert and Update scenarios
motivePower	Gas	Populated for Insert and Update scenarios
groupName	Passenger Vehicles	Derived from productCategoryGroupName

---

5. Source System Queries (Logical Representation)

> Note: The following SQL statements are **representative** and used to document business intent.
> They are not executed against a physical database in this implementation.

5.1 SELECT Existing Product Category

```
SELECT
  product_line,
  product_category_code,
  product_category_group_code,
  product_category_name,
  door_type,
  cylinder_count,
  motive_power,
  product_category_group_name,
  business_key_delete_indicator
FROM PRODUCT_CATEGORY
WHERE product_line = :productLine
  AND product_category_code = :productCategoryCode
  AND product_category_group_code = :productCategoryGroupCode;
```

5.2 INSERT Product Category (Insert Scenario)

```
INSERT INTO PRODUCT_CATEGORY (
  product_line,
  product_category_code,
  product_category_name,
  product_category_group_code,
  door_type,
  cylinder_count,
  motive_power,
  product_category_group_name
)
VALUES (
  :productLine,
  :productCategoryCode,
  :productCategoryName,
  :productCategoryGroupCode,
  :doorType,
  :cylinderCount,
  :motivePower,
  :groupName
);
```

5.3 UPDATE Product Category (Update Scenario)

```
UPDATE PRODUCT_CATEGORY
SET
  product_category_name = :productCategoryName,
  door_type = :doorType,
  cylinder_count = :cylinderCount,
  motive_power = :motivePower,
  product_category_group_name = :groupName
WHERE product_line = :productLine
  AND product_category_code = :productCategoryCode
  AND product_category_group_code = :productCategoryGroupCode;
```

5.4 DELETE Product Category (Delete Scenario)

```
DELETE FROM PRODUCT_CATEGORY
WHERE product_line = :productLine
  AND product_category_code = :productCategoryCode
  AND product_category_group_code = :productCategoryGroupCode;
```

---

6. Response Handling

- PUT returns:
    - transformed payload for Insert / Update / Delete
    - empty response for No-Operation
- GET returns:
    - current representation of product category data
- Errors follow standard HTTP status codes (400 / 500)

---

7. Scope Limitations

The following are intentionally out of scope:

- History tables
- Transaction rollback logic
- Multi-system synchronization
- Physical database persistence