# Pushing products

EVA is not a product information management system. We tried, we weren't particularly good at it, and customers were using other systems as their golden record anyways. So, we stopped trying to be something we're not. Nowadays, we EXPECT our customers to use some other PIM system, and we just provide a neat little service to push all of your product information into EVA.

## Single product

A simple, single product upload through `ImportProducts` would look something like this:

```json
{
  "SystemID": "InsomniaImport",
  "Products": [
    {
      "ID": "0001",
      "Name": "Unisex T-shirt",
      "TaxCode": "High"
    }
  ]
}
```

In the above example, we create a product called 'NewBorn T-shirt' with a `BackendID` (ID) and a `TaxCode`. We use `SystemID` to identify the system from which we are importing our products.

In addition to the BackendID we provide in the `ID` field, a product can have a `CustomID`. If you don't explicitly specify a `CustomID`, it will be the same as your `BackendID`.

#### Response

Your products will always have an `EvaID` assigned upon creation. Your response will include these EVA IDs, grouped by which products were updated or created. When products are created, they are also always updated. Additionally, we return a `BackendID`/`EvaID` mapping.

```json
{
 "UpdatedProductIDs": [
   1
 ],
 "CreatedProductIDs": [
   1
 ],
 "ProductMap": [
   {
     "BackendID": "0001",
     "ID": 1
   }
 ]
}
```

:::info Notes

Products with different `BackendIDs` can have the same `CustomID`, for example when the latter replaces the first as a successor. :::

## Product types

A product can have several types:

To describe your product as being one or more of these types, we use the `Type` object:

```json
```

{

  "SystemID": "PimCore",

  "Products": [

   {

    "ID": "121",

    "Name": "NewBorn T-Shirt",

    "TaxCode": "High",

    "Type": {

     "Stock": true

    }

   }

  ]

}

~~~

Barcodes

To add a barcode to a product, use the **Barcodes** element. You can also specify a **Barcode** (and its **Quantity**) for an existing [unit of measure](link) in the call to allow for easy scanning.

<Tabs>
<TabItem value="json1" label="Adding Barcodes">

```json

{

{

"SystemID": "PimCore", "Products": [

"ID": "121",
"Name": "NewBorn T-Shirt", "TaxCode": "High", "Barcodes": [

"978020137957"
~~~

] }

]

}

</TabItem>
<TabItem value="json2" label="Adding UOM Barcodes">

```json

{

"SystemID": "PimCore", "Products": [

{
"ID": "121",
"Name": "NewBorn T-Shirt", "TaxCode": "High", "Barcodes": [

"978020137957"

] "UnitBarcodes": [

"Barcode":"4566569787", "Quantity":"48", "UnitOfMeasureID": "5"

] }

]

}

```

</TabItem>
</Tabs>

## Product hierarchy

Product hierarchy is defined by the `Variations` property. This in turn contains a new `Products` property where every product should contain a value for the variation. The following request contains a product with four sizes:

```json {

"SystemID": "PimCore", "Products": [
{
"ID": "121",
"Name": "NewBorn T-shirt",

"TaxCode": "High", "Variations": {

"Property": "size", "LogicalLevel": "size", "Products": [

{
"ID": "121-978020137957", "Name": "NewBorn T-shirt", "VariationValues": [

{
"Value": "3-6 months"

} ]

}, {

"ID": "121-978020137958", "Name": "NewBorn T-shirt", "VariationValues": [

{
"Value": "6-12 months"

} ]

}, {

"ID": "121-978020137959", "Name": "NewBorn T-shirt", "VariationValues": [

{
"Value": "12-18 months"

} ]

}, {

"ID": "121-978020137960", "Name": "NewBorn T-shirt", "VariationValues": [

{
"Value": "18-24 months"

} ]

} ]

} }

] }
```

~~~
:::danger Note

Both `color` and `size` are NOT native EVA product properties. These have to be created before they can be used. See [Custom product properties](/link).

:::

## Failure responses

### Partial failure

Let's assume you made an API call to update 100 products but out of this only 90 products were updated. The remaining 10 failed to update in EVA. The response would then contain the parts of the call that failed, and the reason why they failed.

### Unknown tax code

```json {

"Error": {
"Message": "Product 123 uses unknown TaxCode", "Type":
"ImportProducts:UnknownTaxCode", "Code": "JMJVDCGE",
"RequestID": "1cc5912d5059455594a992d1c17c1a5b"

} }
~~~

### Validation failure

```json {

"Error": {
"Message": "The provided request did not pass validation. Failures: Reason:

MissingField, ProductID: , Field: PrimitiveName, Message: \nReason: MissingField, ProductID: , Field: PrimitiveName, Message: \nReason: DuplicateBackendID, ProductID: , Field: BackendID, Message: 123",

"Type": "ImportProducts:ValidationFailures", "Code": "BMIYFDBY",
"RequestID": "2161b89f451b4c47901c74f714c55280"

} }
```

The message contains technical-human-readable text that can be used for debugging.

## Editing products

To edit products, just use the same request and alter the information. EVA will know to update the product if the specified `ID` is already in use.

## Deleting products

To delete products, use the same request, but add the `"IsDeleted": true` property on the products and set it to true. The products will then be deleted. It's good to know that the products actually remain in EVA, their status will just be: *Deleted* and they won't be visible on the front ends anymore.

<!--

-->

# Product

**Product

`Content`

## Images

We will illustrate how this works using a simple product with a simple image first:

```json {

"SystemID": "InsomniaImport", "Products": [

{
"ID": "1",
"Name": "Unisex T-shirt", "TaxCode": "High", "Content": [

{
"Images": [

{
"ImageUrl": "https://picsum.photos/200"

} ]

} ]

} ]
```

} ```

## Languages

Note how the `Content` property in our example request is an array, that's because you can add content for multiple languages:

```json {

"SystemID": "InsomniaImport", "Products": [

{

content

content can be included in the `ImportProducts` service using the array on product level.**

"ID": "1",
"Name": "Unisex T-shirt", "TaxCode": "High", "Content": [

{
"LanguageID": "nl",
"ShortDescription": "Unisex heavyweight t-shirt met geborduurd Nerd

logo.",
{

logo.",
{

} ]

} ]

} ]

} ```

## Native

content options

"Images": [
"ImageUrl": "https://picsum.photos/"

} ]
```

}, {

"LanguageID": "en",
"ShortDescription": "Unisex heavyweight t-shirt with embroidered Nerd

"Images": [
"ImageUrl": "https://picsum.photos/"

EVA ships
front-ends. If you are looking for possibilities beyond our native options though, see [Custom product properties](/link).

## USP Text and USP Blobs

Using the `UniqueSellingPointTexts` and `UniqueSellingPointBlobIDs` you can display product information on **POS** and **Companion App** directly in the search result by providing the respective input to the referred to properties.

Here is what this would look like using our NewBorn T-Shirt example: ```json
{

"SystemID": "PimCore", "Products": [

{
"ID": "121",
"Name": "NewBorn T-Shirt", "TaxCode": "High",

some native product properties which have some functionalities in our

    "Content": [
      {

"LanguageID": "nl",
"ShortDescription": "NewBorn t-shirt voor de kleintjes.", "Images": [

{
"ImageUrl": "https://i.ibb.co/tDVGybm/newborn-logo.jpg"

}],

"UniqueSellingPointTexts": ["Green choice",
"Vegan"

], "UniqueSellingPointBlobIDs": [

"290c4a1e-3ac0-4d69-b7d2-c622a55352e2",

"62cc81df-c975-48de-bba4-0ddf0e85cead"

] }

] }

] }

~~~

:::note
- The text and blobs are order sensitive. So in our example, "Green choice" will link to the first BlobID, "Vegan" to the second, and so forth.
- For the BlobID you can refer to [Blob management](/link).
:::

Properties

You might be wondering about the meaning of each property, but it's quite self-explanatory. Here's a breakdown:

Property	Description
ID	This is the product ID.
TaxCode	Give the correct [tax code](/link) for your product.

| LanguageID
displayed. |
| ShortDescription | A brief description of your product. | | ImageURL | The URL to your product image. |

Custom product properties

| The language in which the short description and image will be

The `ImportProducts` service can be used to fill custom product properties that already exist in EVA. It is also possible to fill properties that do not yet exist, if you define these properties on root level first.

Creating product properties

To create a custom product property, use [CreateProductPropertyType](/link):

```json {
~~~

"TypeID": "eu_ecolabel", "CategoryID": "default", "DataType": 3

} ` ` `

Except for ` TypeID` and ` CategoryID` , this service works the exact same as our [CustomField](/link) services.

### Display values for custom properties

In order to give your properties custom names to be displayed in front ends, use [EditProductPropertyType]( /link):

` ` ` json {

"ID": "eu_ecolabel", "Edits": [

{
"LayerID": 1,

"Content": {
"display_name": "European economy label"

} }

] }

` ` `

Here, ` LayerID` represents your contentlayer ID. ### Additional services

- [SearchProductPropertyTypes]( /link) - [DeleteProductPropertyType]( /link) - [ListProductPropertyType]( /link)
- [DeleteProductPropertyType]( /link)

### Product property categories

Product property categories can be managed using the following services:

- [CreateProductPropertyCategory]( /link) - [EditProductPropertyCategory]( /link) - [ListProductPropertyCategories]( /link) - [CreateProductPropertyCategory]( /link)

:::info Note
Product property categories can't be deleted. :::

## Defining values in ` ImportProducts`
To add values for the custom product property we've just created, we use the

` Content` object in ` ImportProducts` :

```json {

"SystemID": "PimCore", "Products": [

{
"ID": "121",
"Name": "NewBorn T-Shirt", "TaxCode": "High", "Content": [

{
"CustomContent": {

"eu_ecolabel": true }

} ]

} ]

} ```

Since

our custom product property was just a boolean, our case is pretty simple.

:::info Note
Since these product properties live in the content object, they can have different values for different languages.
:::

## Creating product properties in `ImportProducts`
If we didn't have our property preconfigured, we could also just provide it in the

`ImportProducts` message:

```json {

"SystemID": "PimCore", "CustomPropertyTypes": [

{

"ProductPropertyTypeID": "eu_ecolabel", "CategoryID": "default",
"DataType": 3

} ],

"Products": [ {

"ID": "121",
"Name": "NewBorn T-Shirt", "TaxCode": "High", "Content": [

{
"CustomContent": {

"eu_ecolabel": true }

} ]

} ]

} ```

After
will be set on the product.

this message, the product property will be known in EVA and the correct value

## Copying properties to parents

### Overview

On an e-commerce site overview page, you typically want to display color or style-level products instead of size-level products. However, you still want to provide filters on available colors and sizes. This data is often not present at the style or color level unless explicitly provided.

Using the `ProductPropertyType` called `CopyToParentProductPropertyTypeID` you can refer to the ID of another property whose value will be copied to its parent.

### Example Usage

For instance, if a size-level product is available in "S", "M", and "L", setting `CopyToParentProductPropertyTypeID` on the size property to `available_sizes` ensures that these values are copied to the parent (color-level) and grandparent (style-level) products under the `available_sizes` property.

```json

{

 "CustomPropertyTypes": [

  {

   "ProductPropertyTypeID": "size",

   "CopyToParentProductPropertyTypeID": "available_sizes",
```

```
    "DataType": 0,
    "IndexType": 0,
    "IsArray": false
   },
   {
    "ProductPropertyTypeID": "color",
    "CopyToParentProductPropertyTypeID": "available_colors",
    "DataType": 0,
    "IndexType": 0,
    "IsArray": false
   }
  ]
}
```

This configuration automatically copies all distinct size values to their respective parents under `available_sizes`, as well as color.

### Example Response

From `SearchProducts`:

```json
{
  "product_id": 164,
  "available_colors": [
   "blue",
   "red"
  ],
  "available_sizes": [
```

    "S",
    "M",
    "XL",
    "XS"
  ]
}
```

Notes

- Changing the value of `CopyToParentProductPropertyTypeID` for an existing property type affects ONLY products provided in the same request. To apply changes to all products, use the `ComposeProducts` service for a full recompose.

- Properties created via `CopyToParentProductPropertyTypeID` are automatically indexed and set as array properties for filtering and aggregation.

Including content of children to parents/siblings

Overview

This feature enhances display capabilities by allowing child or sibling content to be included in the parent product. While the previous feature focuses on filtering and aggregation, this functionality is purely for display purposes.

Configuration

Two settings are used to define this behavior:
```

- `PIM:Composition:VariationPropertiesToCopy:Children`
- `PIM:Composition:VariationPropertiesToCopy:Siblings`

Both accept a comma-separated list of `ProductPropertyTypes`, such as `color_name,color_hex_value`.

### Example Usage

When configured, the resulting product content includes a `variations` array:

```json
{
  "product_id": 123,
  "variations": [
    {
      "product_id": 456,
      "type": "child",
      "color_name": "Cyan",
      "color_hex_value": "0000FF"
    },
    {
      "product_id": 789,
      "type": "child",
      "color_name": "Bright red",
      "color_hex_value": "FF0000"
    }
  ]
}
```

### Differences from `CopyToParentProductPropertyTypeID`

1. There is a direct reference to the ID of the child/sibling through a product_id property, and the values of the content. With the CopyToParentProductPropertyTypeID commit, you only have a distinct list of values, no relation to products. Having just the values is useful for filtering/aggregation, but not for display/linking.
2. The contents of variation is entirely unindexed, it's not possible to filter on anything inside of it. It's purely meant as enrichment data, not for search/filtering.

### Notes

- By default, variations are not returned by SearchProducts or other services that accept an IncludedFields property, you specifically have to select it by requesting the variations field.

- **Changing the value of these two settings has no immediate effect**, only when a product is composed is the value of this setting checked. So, after changing it you should do a ComposeProducts. We don't do this automatically as this is quite an expensive operation and you might want to test it by composing a few products first.