APIs’ Technical documentation

[1. Presentation 4](#_Toc417485820)

[2. Authentication 4](#_Toc417485821)

[1. Process 4](#_Toc417485822)

[2. Technical precisions 5](#_Toc417485823)

[3. Format of the requests 5](#_Toc417485824)

[4. Data models 5](#_Toc417485825)

[1. Identification of products and offers 5](#_Toc417485826)

[2. Catalog 5](#_Toc417485827)

[3. Product 5](#_Toc417485828)

[4. Offer 6](#_Toc417485829)

[5. Category 7](#_Toc417485830)

[6. Orders 7](#_Toc417485831)

[Sorting possibilities 7](#_Toc417485832)

[Orders retrieval 8](#_Toc417485833)

[Orders update 9](#_Toc417485834)

[Orders update response 9](#_Toc417485835)

[5. Products 10](#_Toc417485836)

[1. Query products with criterias 10](#_Toc417485837)

[2. Response format 11](#_Toc417485838)

[6. Categories 13](#_Toc417485839)

[7. Offers 14](#_Toc417485840)

[1. Offers query by criterias 14](#_Toc417485841)

[8. Catalog 16](#_Toc417485842)

[1. Publishing a catalog of offers/products and asynchronous treatment 16](#_Toc417485843)

[2. Catalog submission 16](#_Toc417485844)

[Request 16](#_Toc417485845)

[Limitation 17](#_Toc417485846)

[Response Code 1: Batch created (JSON) 18](#_Toc417485847)

[Response Code 2: Technical error with no details (JSON) 18](#_Toc417485848)

[3. Batch interrogations 18](#_Toc417485849)

[Process 18](#_Toc417485850)

[Response (JSON) 18](#_Toc417485851)

[Limitations 19](#_Toc417485852)

[9. Merchants 19](#_Toc417485853)

[10. Clients 19](#_Toc417485854)

[11. Cart 19](#_Toc417485855)

[1. Presentation 19](#_Toc417485856)

[2. Add an offer to the cart 20](#_Toc417485857)

[3. Cart querying 20](#_Toc417485858)

[4. Cart validation, and creation of new order(s) 20](#_Toc417485859)

[12. Orders 21](#_Toc417485860)

[1. Query orders 21](#_Toc417485861)

[2. Order update 22](#_Toc417485862)

[Status update 22](#_Toc417485863)

[Request 22](#_Toc417485864)

[Response 23](#_Toc417485865)

[Response in case of a mistake 24](#_Toc417485866)

[13. Appendix 26](#_Toc417485867)

[1. Catalog: Request example 26](#_Toc417485868)

[Product and offer creation 26](#_Toc417485869)

[Catalog: offer creation on existing product 28](#_Toc417485870)

[Catalog: update of an existing offer 30](#_Toc417485871)

[Catalog: offer creation with options 31](#_Toc417485872)

# Presentation

This document defines the interaction between the market place and the Market Place factory’s APIs. The API permits to undertake several actions such as:

* Add / Change products and/or offers
* List categories
* Export products and offers
* Manage orders

Some precision first concerning products and offers:

* A product is a selling available entity on the market place. It is defined by a title, a brand, an account identifier, and a description.
* An offer is the product delivered by a merchant. It includes selling information like the price, the sales information, and the stock.

An offer is directly related to a product, and a product can have multiple offers.

**Important**: URLs given as examples, and based on <http://api.marketplacefactory.com> are purely fictional and meant to represent the API URL of a marketplace based on Market Place Factory technology.

**Important**: there are two levels of privileges required for the use of the Market Place factory APIs. Certain methods are meant to be used by the merchants (i.e. orders, and catalogs methods). Other methods are meant to be used by higher privileged partners, and are meant to give access to core functions of the MP (i.e. offers, products and cart methods).

# Authentication

## Process

To use the API, you first need an API account and a private key given with your subscription.

For each request submitted to the APIs, the following information will be mandatory:

* Your API’s account identifier
* The GMT+0 hour of your request sending
* The same hour signature, with HMCA- SHA1 with your private key, converting in a base 64.

These data have to be included in the http header of each request. If a data is missing or incorrect, the API will send back a 401 status (unauthorized). If the authentication is a success, the submitting request is registered for treatment.

## Technical precisions

The requesting information has to be given with:

* **API’s account identifier:** This account identifier is a **Globally unique identifier (Guid).** It means a coding value on 128 bits which appears as a 32 hexadecimal number divided into 5 groups separated by a dash.
* **The sending hour request:** The waiting hour is the GMT+0 hour in the DataTime format. We can verify the hour of the request thanks to this value, which has also to be signed with your private key. This value has to be in the header “GMT-date” of the http request.
* **Time signature**: To sign the time, you must convert the GMT+0 time from DateTime to String, the converted time must now be in a HMAC-SHA1 hash function. The key of this hash function will be the private key that you have been provided with. Then, the outcome has to be encoded in base 64. This value has to be put in the "request-signature" header of the http request.

# Format of the requests

Market Place Factory API is compatible with both JSON and XML data formats, for requests and response. The sending and returning formats must be configured in the http headers. The sending format is put in the "Content-type" header. The returning format is specified using the "Accept" header. These headers accept the following values:

* **"application/json"** for the **JSON** format
* **"application/xml"** for the **XML** format

# Data models

## Identification of products and offers

The products and offers are identified using one of the following ID:

* **EAN:** European identifier on 13 characters (string)
* **ISBN**: International Standard Book Number (string)
* **UPC**: Universal Product Code on 12 characters (string)
* **MPID:** internal product identifier of the marketplace (int)
* **SKU**: internal offer identifier of the merchant (string)

## Catalog

**catalog**: list of “catalogitem”. Each item is composed of 1 product and/or 1 offer.

## Product

**partnerld (int):** Reference id of the product of the partner

**Title (string)** **[mandatory]:** model of the product

**Images (string []):** list of URLs of the product images. The first one is defined as main image.

**Description (string):** long description of the product in the marketplace folder-tree view.

**shortdescription (string):** short description of the product

**Brand (string):** brand

**categoryld (int):** ID of the category used for putting the product in the marketplace folder-tree view

**categories (string) :** full hierarchy of the product in the MP’s taxonomy

**ean / isbn / upc (string):** identifier of the product

**mpld (int):** identifier of a product within a database

**brandld (int)** : identifier of a manufacturer

**reference (string):** reference of the product

**dateAdded (DateTime):** Date added of the product

**dateModified (DateTime):** date of modification of the product

**validation (short):** validation of a product

**source (short):** source of a product

mpn (string): manufacturer’s part number

## Offer

**sku (string):** compulsory when the offer is created only. Must be unique in the manufacturer catalogue

**offerld (int):** identifier of an offer within a database

**deliveryModes:** delivery mode for a zone (defined by the marketplace) or a country. Should delivery prices be different for two countries belonging to the same zone, the higher price will be taken into account. It is possible to set a common delivery price that will be applied to all the zones.

**name (sting):** name of the delivery mode

**price (int):** default price

**zones:** list of delivery zones concerned by the offer

**zoneName (string):** name of a delivery zone

**priceFirst (float):** shipping price applied to the first article of an order

**priceNext (float):** shipping price applied to every new article

**countries:** list of countries in which the offer proposes to deliver

**countryCode (string):** international code of the concerned country (ex: "FR" for France)

**priceFirst (float):** shipping price applied to the first article of an order

**priceNext (float):** shipping price applied to every new article

**options:** list of options for simple products with options. For each option:

**name:** name of the option (ex: “L” or “10.5”)

**stock:** quantity available in stock for the option

**optionsName:** name of the option, relative to the options list (ex: “size” or “shoes size”)

**price (decimal)** **[compulsory]:** price in euros

**pricePromo (decimal):** Promo prices in euros (facultative)

**promoDateStart (datetime):** date of the launch of the promo (yyyy-mm-dd format). Facultative, even though the pricepromo is set. If it is not specified, the promo is still active.

**promoDateEnd (datetime):** date of the end of the promo (yyyy-mm-dd format). Facultative, even though the pricepromo is set. If it is not specified, the promo is still active.

**shippingDelay (int):** delay before the order is shipped by the merchant, in days. Facultative.

## Category

**id**: identifier of the category within the MarketPlaceFactory database. This identifier is used in the registration of product on our platform.

**Name**: name of the category

**listSubCategories:** list of the sub-categories linked to the concerned category

## Orders

### Sorting possibilities

**orderld:** filter the commands by identifier number

**sortBy (string):** Type of sorting desired to get merchant's list of commands

**sortOrder (string):** Sort by ascendant order ("asc") or by desendant order ("desc")

**status (int):** Sort the commands by status (Cancelled, Waiting, Accepted, Shipped)

**dateStart:** Sort the commands by date of creation of the command

**dateEnd:** Sort the commands by date of last modification of the command

**payment (short):** Filter the command by payment modes

**page (int):** Desired page of results

### Orders retrieval

**orderld (int):** identifier of the order within the MarketPlaceFactory database. This identifier is used in the registration of the commands on our platform

**date (date):** date of creation of the order in DateTime format.

**articlesNbr (int):** Number of articles of the order

**amount (int):** amount of the command all taxes included

**clientName (string):** Name of the client who ordered

**status (string):** State of the command among three possible states:

* Pending: the command is waiting for the validation of the seller
* Validated: the command is validated by the seller
* Shipped: the command is shipped by the seller

**trackingNumber (string):** Tracking number of the parcel after the shipping

**deliveryName (string):** Delivery address of the client who made the order

**deliveryCompanyName (string):** Delivery address of the client who made the order

**deliveryPostcode (strin):** Delivery address of the client who made the order

**deliveryCountry (string):** Delivery address of the client who made the order

**billingName (string):** Billing address of the client who made the order

**billingCompanyName (string):** Billing address of the client who made the order

**billingAddress (string):** Billing address of the client who made the order

**billingPostcode (string):** Billing address of the client who made the order

**billingCity (string):** Billing address of the client who made the order

**billingCountry (string):** Billing address of the client who made the order

**products:** List of the products of the command

**productld (ind):** identifier of the product

**image (string):** URL address of the image corresponding to the product

**title (string):** Name of the product

**brand (string):** Name of the manufacturer of the product

**oneProducePrice (decimal):** Price of the product per unit

**quantity (decimal):** Quantity of the product in the command

**subPrice (decimal):** Price of the product per unit and multiplicate by the quantity

**sku (strin):** Stock Keeping Unit, internal identifier of the seller

**merchantComment (string):** Comment done by the merchant on the command

### Orders update

**comment (string):** free text comment

**orderld (int)** **[compulsory]**: ID of the command to be modified

**status (string)** **[compulsory]:** New status to be attributed to the command

**trackingNumber (string)** [compulsory]: Tracking number to specify when a command gets the shipped status

**products:** list of the products to be validated during a partial validation. The non-specified products in the list are all validated.

**productld (int):** ID of the product to be validated

**quantity (decimal):** Quantity of a product to be validated

### Orders update response

**code (string):** return code of the modification of the command

**orderld (int):** Id of the concerned command

**description (string):** Precised description of the modification return of the command

**status (string):** Status of the modification of the command

**productsError:** List of products that encounter an error during the modification

**productld (int):** ID of the product in error

**productName (string):** Name of the product in error

**baseQuantity (decimal):** Quantity of the product before a modification test

**error (string):** Error report concerning the product

# Products

Using Cloud Commerce Factory API, one can retrieve and query the products of the MP.

It is a GET request at the following address:

<http://api.marketplacefactory.com/products/>

## Query products with criterias

Following criteras can be used:

* mpId
* partnerId
* title
* brand
* ean
* dateAdded
* dateModified
* categoryName
* categoryId
* isbn
* upc
* withOffers
* page

These criteria have to be appended to the URL [http://api.marketplacefactory.com/products/?](http://api.marketplacefactory.com/products/?model=model&category=categorie), and separated by "&".

For instance, to obtain the list of the products belonging to category "Clothes" and containing "Tees" in their title, send a GET request to the address:

[http://api.marketplacefactory.com/products/?categoryName=*clothes&title=tees*](http://api.marketplacefactory.com/products/?categoryName=clothes&title=tees)

Every request includes a maximum of 50 products, and pagination is required to get the following products. Pagination is done using the "page" attribute.

Example: http://api.marketplacefactory.com/products/?page=2 will show the products from index 51 to index 100.

## Response format

XML

<products xmlns="http://www.marketplacefactory/api.xsd">

<pageIndex>1</pageIndex>

<pageSize>50</pageSize>

<recordsReturned>1</recordsReturned>

<startIndex>1</startIndex>

<totalRecords>1</totalRecords>

<products>

<product>

<brand>Diesel</brand>

<categoryId>1466</categoryId>

<categoryName>Vintage</categoryName>

<dateAdded>2008-12-03T02:41:00</dateAdded>

<description>La description longue obligatoire.</description>

<brandId>772</brandId>

<mpId>15590</mpId>

<partnerId>32</partnerId>

<shortDescription>Description courte</shortDescription>

<title>Pantalon jean wash</title>

<validation>2</validation>

</product>

<products>

<products>

JSON

[

{

"products": [

{

"mpid": 15590,

"partnerId": 32,

"title": "Pantalon jean wash",

"brand": "Diesel",

"manufacturerId": 772,

"description": "La description longue obligatoire.",

"shortDescription": "Description courte",

"dateAdded": "2008-12-03T02:41:00",

"categoryName": "Vintage",

"categoryId": 1466,

"validation": 2,

"source": 0

}

],

"recordsReturned": 1,

"totalRecords": 1,

"startIndex": 1,

"sort": 0,

"pageIndex": 1,

"pageSize": 50

}

# Categories

To add a product to the marketplace, you need the category identifier in order to categorize the product in the marketplace’s taxonomy. If one does not wish to use categories’ names, the CloudCommerceFactory API allows you to retrieve the hierarchic categories list with their identifiers. This is done sending a GET request to the URL <http://api.marketplacefactory.com/categories/>.

Examples of the response in JSON and XML formats

<categories>

<category>

<id>294</id>

<listSubCategories />

<name>Vêtements </name>

</category>

<category>

<id>333</id>

<listSubCategories>

<category>

<id>1475</id>

<listSubCategories />

<name>Intérieur </name>

</category>

<category>

<id>1250</id>

<listSubCategories />

<name>Extérieur </name>

</category>

</listSubCategories>

<name>Maison </name>

</category>

<category>

<id>582</id>

<listSubCategories />

<name>Bijoux </name>

</category>

</ categories >

[

{

"id":294,

"name":"Vêtements",

"listSubCategories":[]

},

{

"id:333,

"name":"Maison",

"listSubCategories":[

{

"id":1475,

"name":"Intérieur",

"listSubCategories":[]

},

{

"id":1250,

"name":"Extérieur",

"listSubCategories":[]

}

]

},

{

"id":582,

"name":"Bijoux",

"listSubCategories":[]

}

]

# Offers

Offers determine for a certain merchant and product the price, the shipping cost, and some others terms. (See section 4 *data model* to see information concerning the offers).

## Offers query by criterias

For a simple search of offers by criteria, one needs to send a GET query at the URL <http://api.marketplacefactory.com/offers>.

Example:

http://api.marketplacefactory.com/offers?brand=marque&priceProduct=15&page=3&sort=desc

The criteria are as follow:

* offerId: The offer of the ID
* title
* brand
* reference
* productId: The referring product of the offer
* price
* merchantUserId: Merchant Id of the proposing offer
* merchantName : Merchant name of the proposing offer
* statusOffer
* dateStart: Beginning of sales
* dateEnd: End of sales
* sku: SKU identifier of the offer
* promotion: Sales offer or not
* priceMin: Minimum price of the offers
* priceMax: Maximum price of the offers
* sortBy: Sorting criteria
* page: The desired search page
* sort: ascending or descending order

4 sorting optionss are available:

* date : sort by date
* merchantRate: sort by sell by merchant
* orderMerchantPerMonth: sort by average monthly sales by merchant
* price : sort by price

When you are sending a request to the offer API, it refers to some data flows with some information:

* The page number
* The number of result by page
* The number of result
* The first element of the searching page index
* The total number of result
* The offer list that satisfy the research

XML response example:

<offers xmlns=**"http://www.marketplacefactory/api.xsd"**>

<pageIndex>**1**</pageIndex>

<pageSize>**50**</pageSize>

<recordsReturned>**2**</recordsReturned>

<startIndex>**1**</startIndex>

<totalRecords>**2**</totalRecords>

<offers>

<offer>

<activated>**false**</activated>

<dateAdded>**2012-03-27T11:45:00**</dateAdded>

<id>**31381**</id>

<mpId>**31213**</mpId>

<price>**0.00**</price>

<userId>**f795f2fb-2da4-41b2-b037-9e001d193a2a**</userId>

</offer>

<offer>

<activated>**false**</activated>

<dateAdded>**2012-03-27T11:45:00**</dateAdded>

<dateModified>**2012-04-11T14:23:00**</dateModified>

<id>**31382**</id>

<mpId>**31214**</mpId>

<price>**0.00**</price>

<userId>**f795f2fb-2da4-41b2-b037-9e001d193a2a**</userId>

</offer>

</offers>

</offers>

JSON response example

{

"offers":

[

{

"id": 31381,

"mpId": 31213,

"priceIsPromo": false,

"price": 0,

"dateAdded": "2012-03-27T11:45:00",

"userId": "f795f2fb-2da4-41b2-b037-9e001d193a2a",

"activated": false,

"stock": 0,

"visible": false

},

{

"id": 31382,

"mpId": 31214,

"priceIsPromo": false,

"price": 0,

"dateAdded": "2012-03-27T11:45:00",

"dateModified": "2012-04-11T14:23:00",

"userId": "f795f2fb-2da4-41b2-b037-9e001d193a2a",

"activated": false,

"stock": 0,

"visible": false

}

],

"recordsReturned": 2,

"totalRecords": 2,

"startIndex": 1,

"sort": 0,

"pageIndex": 1,

"pageSize": 50

}

# Catalog

## Publishing a catalog of offers/products and asynchronous treatment

The publication of a catalog involves the following steps:

* **Receipt of the data:** The request is submitted to the APIs. At this point, data is only stored on the server.
* **Batch creation:** The API creates a new “batch”, asking for the asynchronous batch handler to manage the newly received catalog.
* **Processing of the batch**: The server collects information from the request, checks and registers them on the data basis.

The catalog is stored in the Market Place factory data stored only when all these steps are completed.

## Catalog submission

### Request

JSON request example (modification of a product):

{

"catalog":[{

"product":{

"ean":1234,

"brand":"Nouvelle Marque",

"categoryId":2,

"description":"Nouvelle Description",

"images":[ "http://s8.decofinder.com/0/0//vig3/624/624008/Mis\_En\_Demeure.jpg",

"http://s.decofinder.com/0/0//vig3/652/652980/Mis\_En\_Demeure.jpg"],

"shortDescription":"Descr2",

"title":"Fabuleux produit",

"partnerId":123

}

}]

}

XML request example:

<?xml version="1.0" encoding="UTF-8"?>

<requestCatalog xmlns="http://schemas.datacontract.org/2004/07/MPf.Web.API.Models.Catalog" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<catalog>

<catalogItem>

<product>

<brand>Nouvelle Marque</brand>

<categoryId>2</categoryId>

<description>Nouvelle Description</description>

<ean>1234</ean>

<images>

<url>

<http://s8.decofinder.com/0/0//vig3/624/624008/Mis_En_Demeure.jpg>

</url>

<url>

<http://s.decofinder.com/0/0//vig3/652/652980/Mis_En_Demeure.jpg>

</url>

</images>

<partnerId>123</partnerId>

<shortDescription>Descr2<shortDescription>

<title>Fabuleux produit</title>

</product>

</catalogItem>

</catalog>

</requestCatalog>

### Limitation

Each **catalog** item includes strictly **1product and/or 1 offer**.

In the case of a multiple add, each product or offer added are in a new catalog item.

To create or update an offer on an existing product, you will find the offer on the product part simply specifying a valuable identifier. (Example **annexe 12.1**)

### Response Code 1: Batch created (JSON)

{

‘responseCode’:1,

‘responseLabel’:’Stored for processing’,

‘batchId’:’5AEB837E-70A4-4EE2-9DE5-CE7DB2994994’

}

The flow has been stored and the generated batch (with the request information) is referenced in **batchId.**

### Response Code 2: Technical error with no details (JSON)

{

‘responseCode’:2,

‘responseLabel’:‘Error’,

‘errorCode’:1,

‘errorLabel’:‘Internal technical error’

}

This is the example of what would be the response if a technical error occurs. In this case, the catalog has to be submitted again.

## Batch interrogations

### Process

The batch previously created is queried doing a GET request.

URL example :

<http://api.marketplacefactory.com/catalog/5AEB837E-70A4-4EE2-9DE5-CE7DB2994994>

If no batch with the ID is found for the merchant, **a Code 404 status (Not found)** is sent back.

Reminder: A batch is identified by a Guid (16 bytes identifier)

### Response (JSON)

{

‘batchId’:’5AEB837E-70A4-4EE2-9DE5-CE7DB2994994’,

‘status’:‘Processing’

}

A processing state related to a global state is given to a batch

* Waiting
* In process
* Finish
* Error

The “error” case is global to the batch. It means the entire batch failed.

When the treatment is “finish”, a state of each line is returned. It let you know if a product and/or an offer has been taken into account.

### Limitations

A batch can only be consulted by the merchant at the beginning of the creation. Otherwise, a **Code 404 status (Not found)** appeared even if the batch exists.

# Merchants

A selling merchant account is an account that has the possibility to be connected to the platform. Then it can create offer on a product in order to make a sale. The seller will be able to add new products, depending on his privilege level

* Categories
* Catalog

# Clients

A client account is an account registered on the Market Place factory platform to buy products through the offers posted by the merchant account. They access to the products and offers through our partners' website only.

# Cart

## Presentation

The shopping cart has been designed to be plugged on any e-commerce/non e-commerce website by connecting to the cart API. The website will be referred to as the “partner”, and it is assumed that the partner already manages users’ accounts, and unique identification of users.

The cart API is designed to easily allow a partner to “instantiate” a new CloudCommerceFactory cart. Its particularity is its ability to handle products from different merchants, and once persisted as an order, dispatch the orders to every merchant concerned.

There are 2 use-cases at the partner level:

* The user is authenticated and its email is known
* The user is anonymous

Depending on the case, the partner will be provided with three possibilities:

* Ask for the cart of an authenticated user by email. The cart is persisted, and if it has not been previously validated (as an order), it will be returned by the API.
* Ask for an anonymous cart. A new cart ID, of the GUID type (16 bytes identification) is returned.
* Ask for an anonymous cart by cart ID (GUID)

## Add an offer to the cart

The add method requires the following information:

* OfferID [mandatory] : int
* Quantity (default 1 if not specified) : int
* Option : int
* Cart ID : GUID or email

## Cart querying

Query a cart by ID or email address, and returns the following information:

* Offer list with detail for each offer:
  + Title: string
  + Quantity : int
  + Image URL : string
  + Seller’s name (merchant) : string
  + Delivery rate : float
  + Available delivery methods, with names and rates:
    - Name : string
    - Rate (based on the offer quantity) : float
* Total price: float
* Total delivery rates: float

Delivery methods and rates depend on the country specified in the cart querying request.

## Cart validation, and creation of new order(s)

Cart validation requires the partner to provide the API with the customer data:

* First name [mandatory]
* Last name [mandatory]
* Address [mandatory]
* Post code [mandatory]
* Country [mandatory]
* Delivery address (if different)
* Email [mandatory]
* Password: auto-generated if blank

Once saved, the cart is persisted as orders(s) (one per merchant, with different order id each), and notifications and triggered, and order ids are sent back by the API. The cart is then emptied. The notifications are:

* Email(s) to customer
* Email(s) to merchant(s)

The orders then become available in the merchant and customer backoffice, and through the APIs.

# Orders

Orders can be retrieved and fully managed by merchants through the API.

## Query orders

You can download the list of orders associates with your merchant account with the CloudCommerceFactory APIs.

<http://api.marketplacefactory.com/orders/>

* Referred to chapter 4 (section 4.5.1) for the different filtration order options

You also need to specify in the http header “Accept” the answer format.

XML answer example:

<orders xmlns="http://www.marketplacefactory/api.xsd">

<pageIndex>1</pageIndex>

<pageSize>50</pageSize>

<recordsReturned>2</recordsReturned>

<startIndex>1</startIndex>

<totalRecords>2</totalRecords>

<orders>

<order>

<billingAddress>21</billingAddress>

<billingCity>Ecommoy</billingCity>

<billingName>Paul Saltz</billingName>

<billingPostcode>72220</billingPostcode>

<customerName>Paul Saltz</customerName>

<date>2012-10-14T12:49:00</date>

<deliveryAddress>21</deliveryAddress>

<deliveryCity>Ecommoy</deliveryCity>

<deliveryName>Paul Saltz</deliveryName>

<deliveryPostcode>72220</deliveryPostcode>

<orderId>3548</orderId>

<products>

<product>

<image>http://www.monsite.com/products\_no\_image\_Medium.jpg</image>

<price>10.00</ price >

<brand>Cool brand</brand>

<productId>1465</productId>

<productName>guitare</productName>

<quantity>1</quantity>

<sku>FR-89273</sku>

</product>

</products>

<status>Shipped</status>

<trackingNumber>Remplacer par le numero de suivi</trackingNumber>

</order>

## Order update

### Status update

With the CloudCommerceFactory API, you can change the status of one or several order(s) associate to your merchant account, in the following status:

* **Cancelled:** the order is cancelled.
* **Waiting:** The order is waiting for the merchant validation.
* **Accepted:** The order is validating and waiting to be shipped.
* **Shipped:** The order is shipped.
* **ShippedAndReceived:** The order is received by the client.

When an order changes its status, a comment can be inserted.

When the order gets the “shipped status”, a tracking number can be specified.

During the validation and shipment of an order, the order can be partially validated.

To do it, you just need to send a POST request at:

<http://api.marketplacefactory.com/orders/>

* Refered to Chapter 4 section 4.5.3.

### Request

<?xml version="1.0" encoding="utf-8"?>

<ordersRequest xmlns="http://www.marketplacefactory/api.xsd">

<orders>

<order>

<comment>Test</comment>

<products>

<product>

<productId>1446</productId>

<quantity>1</quantity>

</product>

</products>

<orderId>3548</orderId>

<status>Shipped</status>

<trackingNumber>TN126578</trackingNumber>

</order>

</orders>

</ordersRequest >

JSON request

{  
    requestupdate:{  
        orders:{  
            order:{  
                comment:'Test',  
                products:{  
                    product:{  
                        productid:1446,  
                        quantity:1  
                    }  
                },  
                orderid:3548,  
                status:'Shipped',  
                trackingnumber:'TN126578'  
            }  
        }  
    }  
}

### Response

XML response

<reponseUpdate xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.marketplacefactory/api.xsd">

<orders>

<order>

<code>SUCCESS</code>

<description>Successfully updated this order</description>

<orderId>3549</orderId>

<status>OK</status>

<products>

<product>

<productId>1446</productId>

<quantity>1</quantity>

</product>

</products>

</order>

</orders>

</reponseUpdate>

JSON answer:

{  
    reponseupdate:{  
        orders:{  
            order:{  
                code:'SUCCESS',  
                description:'Successfully updated this order',  
                orderid:3549,  
                status:'OK',  
                products:{  
                    product:{  
                        productid:1446,  
                        quantity:1  
                    }  
                }  
            }  
        }  
    }  
}

### Response in case of a mistake

XML

<reponseUpdate xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.marketplacefactory/api.xsd">

<orders>

<order>

<code>ERR\_04</code>

<description>Product(s) Update has failed</description>

<orderId>2974</orderId>

<productsError>

<product>

<baseQuantity>0</baseQuantity>

<error>Please specify a correct quantity.</error>

<productId>28</productId>

<productName>T-shirt rouge taille unique</productName>

</product>

</productsError>

<status>ERROR</status>

</order>

</orders>

</reponseUpdate>

JSON:

{  
    reponseupdate:{  
        orders:{  
            order:{  
                code:'ERR\_04',  
                description:'Product(s) Update has failed',  
                orderid:2974,  
                productserror:{  
                    product:{  
                        basequantity:0,  
                        error:'Please specify a correct quantity.',  
                        productid:28,  
                        productname:'T-shirt rouge taille unique'  
                    }  
                },  
                status:'ERROR'  
            }  
        }  
    }  
}

# Appendix

## Catalog: Request example

### Product and offer creation

JSON:

{

"catalog":[{

"product":{

"ean":1234,

"brand":"Marque",

"categoryId":1,

"description":"Description",

"images":[

"http://image.cloudcommercefactory.com/0/0//vig3/624/624008/Mis\_En\_Demeure.jpg",

"http://image.cloudcommercefactory.com/0/0//vig3/652/652980/Mis\_En\_Demeure.jpg"],

"shortDescription":"Descr",

"title":"Titre",

"partnerId":112233

},

"offer":{

"sku":"MP123L",

"price":20,

"pricePromo":15,

"promoDateEnd":"2012-12-24",

"promoDateStart":"2012-09-12 14:55:00",

"stock":100,

"deliveryModes":[

{

"name":"Chronopost",

"price":30

},

{

"name":"Colissimo",

"countries":[{"countryCode":"FR","priceFirst":3,"priceNext":2}]

}]

}

}]

}

XML :

<?xml version="1.0" encoding="UTF-8"?>

<catalogRequest xmlns="http://schemas.datacontract.org/2004/07/MPf.Web.API.Models.Catalog" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<catalog>

<catalogItem>

<offer>

<deliveryModes>

<deliveryMode>

<name>Chronopost</name>

<price>30</price>

</deliveryMode>

<deliveryMode>

<name>Colissimo</name>

<countries>

<country>

<countryCode>FR</countryCode>

<priceFirst>3</priceFirst>

<priceNext>2<priceNext>

</country>

</countries>

</deliveryMode>

</deliveryModes>

<price>20</price>

<pricePromo>15</pricePromo>

<promoDateEnd>2012-12-24</promoDateEnd>

<promoDateStart>2012-09-12 14:55:00</promoDateStart>

<sku>MP123L<sku>

<stock>100</stock>

</offer>

<product>

<brand>Marque</brand>

<categoryId>1</categoryId>

<description>Description</description>

<ean>1234</ean>

<images>

<url>

<http://image.cloudcommercefactory.com/0/0//vig3/624/624008/Mis_En_Demeure.jpg>

</url>

<url>

<http://image.cloudcommercefactory.com/0/0//vig3/652/652980/Mis_En_Demeure.jpg>

</url>

</images>

<partnerId>112233</partnerId>

<shortDescription>Descr<shortDescription>

<title>Titre</title>

</product>

</catalogItem>

</catalog>

</catalogRequest >

### Catalog: offer creation on existing product

JSON

{

"catalog":[{

"product":

{

"mpId":123

},

"offer":

{

"sku":"MP123L",

"price":20,

"pricePromo":15,

"promoDateEnd":"2012-12-24",

"promoDateStart":"2012-09-12 14:55:00",

"stock":100,

"deliveryModes":[

{

"name":"Chronopost",

"price":30

},

{

"name":"Colissimo",

"countries":[{"countryCode":"FR","priceFirst":3,"priceNext":2}]

}]

}

}]

}

XML:

<?xml version="1.0" encoding="UTF-8"?>

<requestCatalog xmlns="http://schemas.datacontract.org/2004/07/MPf.Web.API.Models.Catalog" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<catalog>

<catalogItem>

<offer>

<deliveryModes>

<deliveryMode>

<name>Chronopost</name>

<price>30</price>

</deliveryMode>

<deliveryMode>

<name>Colissimo</name>

<countries>

<country>

<countryCode>FR</countryCode>

<priceFirst>3</priceFirst>

<priceNext>2<priceNext>

</country>

</countries>

</deliveryMode>

</deliveryModes>

<price>20</price>

<pricePromo>15</pricePromo>

<promoDateEnd>2012-12-24</promoDateEnd>

<promoDateStart>2012-09-12 14:55:00</promoDateStart>

<sku>MP123L<sku>

<stock>100</stock>

</offer>

<product>

<mpId>123</mpId>

</product>

</catalogItem>

</catalog>

</requestCatalog>

### Catalog: update of an existing offer

JSON

{

"catalog":[{

"offer":{

"sku":"MP123L",

"price":10,

"pricePromo":8,

"promoDateEnd":"2012-12-24",

"promoDateStart":"2012-09-12 14:55:00",

"stock":150,

"deliveryModes":[

{

"name":"Chronopost",

"price":30

},

{

"name":"Colissimo",

"countries":[{"countryCode":"FR","priceFirst":3,"priceNext":2}]

}]

}

}]

}

XML:

<?xml version="1.0" encoding="UTF-8"?>

<requestCatalog xmlns="http://schemas.datacontract.org/2004/07/MPf.Web.API.Models.Catalog" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<catalog>

<catalogItem>

<offer>

<deliveryModes>

<deliveryMode>

<name>Chronopost</name>

<price>30</price>

</deliveryMode>

<deliveryMode>

<name>Colissimo</name>

<countries>

<country>

<countryCode>FR</countryCode>

<priceFirst>3</priceFirst>

<priceNext>2<priceNext>

</country>

</countries>

</deliveryMode>

</deliveryModes>

<price>20</price>

<pricePromo>15</pricePromo>

<promoDateEnd>2012-12-24</promoDateEnd>

<promoDateStart>2012-09-12 14:55:00</promoDateStart>

<sku>MP123L<sku>

<stock>100</stock>

</offer>

</catalogItem>

</catalog>

</requestCatalog>

### Catalog: offer creation with options

JSON

{

"catalog":[{

"product":

{

"mpId":123

},

"offer":

{

"sku":"MP123L",

"price":20,

"pricePromo":15,

"promoDateEnd":"2012-12-24",

"promoDateStart":"2012-09-12 14:55:00",

"deliveryModes":[

{

"name":"Chronopost",

"price":30

},

{

"name":"Colissimo",

"countries":[{"countryCode":"FR","priceFirst":3,"priceNext":2}]

}],

"options": [

{ "name":"S", "stock:50" }, { "name":"XL", "stock:15" },

],

"optionsName": "Size"

}

}]

}

XML:

<?xml version="1.0" encoding="UTF-8"?>

<requestCatalog xmlns="http://schemas.datacontract.org/2004/07/MPf.Web.API.Models.Catalog" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<catalog>

<catalogItem>

<offer>

<deliveryModes>

<deliveryMode>

<name>Chronopost</name>

<price>30</price>

</deliveryMode>

<deliveryMode>

<name>Colissimo</name>

<countries>

<country>

<countryCode>FR</countryCode>

<priceFirst>3</priceFirst>

<priceNext>2<priceNext>

</country>

</countries>

</deliveryMode>

</deliveryModes>

<options>

<option><name>S</name><stock>20</stock></option>

<option><name>L</name><stock>30</stock></option>

<option><name>XL</name><stock>40</stock></option>

<option><name>XXL</name><stock>40</stock></option>

</options>

<optionsName>Size</optionsName>

<price>20</price>

<pricePromo>15</pricePromo>

<promoDateEnd>2012-12-24</promoDateEnd>

<promoDateStart>2012-09-12 14:55:00</promoDateStart>

<sku>MP123L<sku>

</offer>

<product>

<mpId>123</mpId>

</product>

</catalogItem>

</catalog>

</requestCatalog>