

ORDERING SCRIPT

User Manual for Creodias

WRITTEN BY
Mateusz Miśkiewicz

TABLE OF CONTENTS

TABLE OF CONTENTS	2
1. INTRODUCTION	3
2. COMMANDS.....	4
2.1. <i>create-order-with-body</i>	4
2.1.1. Use case	4
2.1.2. Output.....	4
2.1.3. What does <i>new_orders</i> parameter do?.....	5
2.2. <i>create-order-with-query</i>	6
2.2.1. Use case	7
2.3. <i>create-order-with-query --hours=x</i>	10
3. CONFIGURATION.....	11
3.1. <i>keycloak_catalogue.json</i>	12
3.2. <i>keycloak_ordering.json</i>	12
3.3. <i>order_body.json</i>	16
3.4. <i>order_details.json</i>	17
3.5. <i>query_details.json</i>	18
4. FINAL REMARKS	20
TABLE OF FIGURES	21
PICTURES.....	21
TABLES	22

You can run the script via *GUI* or a command. In order to run the script with a *GUI*, simply type *python main.py gui* into the terminal. Just choose one of the options and the script will do its magic (ref. *Picture 1*).

Picture 1 – Running the script with a GUI

- `python main.py create-order-with-body`
- `python main.py create-order-with-query`
- `python main.py create-order-with-query --hours=x`

3

2. COMMANDS

In this chapter we're going to break down all the commands available in ordering script.

2.1. create-order-with-body

This option allows you to place order with a body, which means that you're hardcoding all products in *IdentifierList* parameter in *order_body.json*.

2.1.1. Use case

Let's say that you've provided these parameters into *order_body.json*, *order_details.json* and *keycloak_ordering.json* that you're seeing down below. Side note – remember to change *username*, *password* and *totp_code* credentials in *keycloak_ordering.json* to your own (more on that later).

```
{
  "WorkflowName": "card_bs",
  "Name": "your_card_bs_order",
  "IdentifierList": [
    "S1B_IW_GRDH_1SDV_20211223T043834_20211223T043859_030148_03998E_B01A",
    "S1B_IW_GRDH_1SDV_20211223T034414_20211223T034439_030147_039989_16DF",
    "S1B_EW_GRDM_1SDH_20211222T173128_20211222T173205_030141_039962_217D"
  ]
}
```

order_body.json (colors represent different orders)

```
{
  "parallel_quota": 2,
  "new_orders": true
}
```

order_details.json

```
{
  "client_id": "CLOUDFERRO_PUBLIC",
  "username": "username@cloudferro.com",
  "password": "mysecretpassword123",
  "client_secret": "",
  "keycloak_address": "https://identity.cloudferro.com/auth/realms/Creodias-new/protocol/openid-connect/token?",
  "host": "datahub-creodias-eu",
  "totp_code": "XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX"
}
```

keycloak_ordering.json

2.1.2. Output

Two orders will be placed. In this example we provided 3 products as an input of *IdentifierList* parameter in *order_body.json*, but we did set *parallel_quota* (which is a limit of products per one

order) in *order_details.json* to 2. What it means is that we will place an order for the first two products from *order_body.json* and once this order is completed, we will place another one, but this time around for only one product, because it's the only one left, which hasn't been ordered yet. Let's break it down once more. First we're creating an order for two products.

```
{
  "WorkflowName": "card_bs",
  "Name": "your_card_bs_order",
  "IdentifierList": [
    "51B_IW_GRDH_1SDV_20211223T043834_20211223T043859_030148_03998E_B01A",
    "51B_IW_GRDH_1SDV_20211223T034414_20211223T034439_030147_039989_16DF"
  ]
}
```

Body used to create first order

The script will wait and check if the order was already processed. If that's the case, then we're creating an order for the last product.

```
{
  "WorkflowName": "card_bs",
  "Name": "your_card_bs_order",
  "IdentifierList": [
    "51B_EW_GRDM_1SDH_20211222T173128_20211222T173205_030141_039962_217D"
  ]
}
```

Body used to create the second and final order

Naturally, you can adjust your *parallel_quota* parameter. If we would set it to 3, then we'd create one order with 3 products. You can adjust your quota to your needs.

2.1.3. What does *new_orders* parameter do?

I should also mention *new_orders* parameter in *order_details.json*. As you can see, right now it is set to true, but what does it mean? Well, essentially it means that new order will be placed once the old one was processed. Here's what will happen if we would set it to *false* - basically, only one order would be placed.

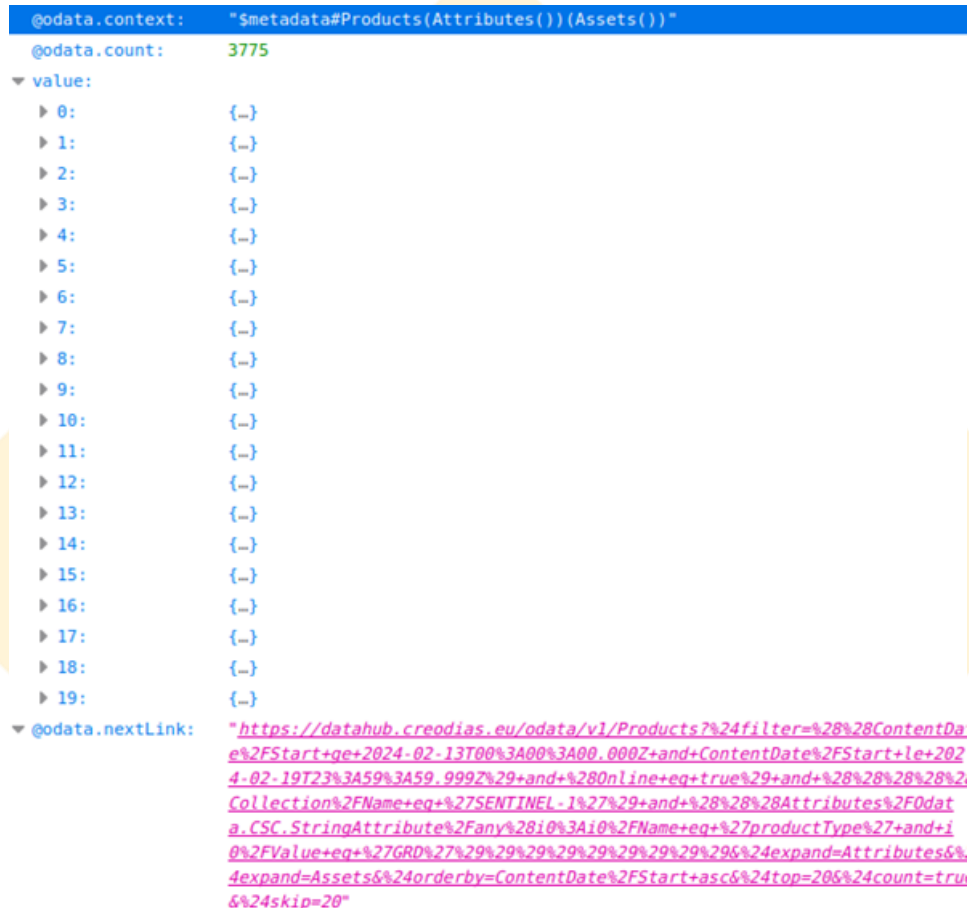
```
{
  "WorkflowName": "card_bs",
  "Name": "your_card_bs_order",
  "IdentifierList": [
    "51B_IW_GRDH_1SDV_20211223T043834_20211223T043859_030148_03998E_B01A",
    "51B_IW_GRDH_1SDV_20211223T034414_20211223T034439_030147_039989_16DF"
  ]
}
```

Body used to create an order

Because we did set *new_orders* parameter to *false*, we're done with creating new orders at this point. If we would set it to *true*, we would continue with creating orders till all products would be ordered.

2.2. create-order-with-query

In the second option you're creating an order by providing a query. Let's say that you've set *query_url* as this URL¹. Let's stop at this query for a minute (ref. Picture 2).



Picture 2 - query_url output

It has 3775 products (*@odata.count* parameter), which is quite a lot. Imagine if you had to hardcode all these products in *order_body.json* all by yourself. That would be a pain, right? Well, *create-order-with-query* gives you an option to avoid just that. All you need to do is provide a query and some other minor parameters. If you do that, the script will look for products in the query and

¹ [https://datahub.creodias.eu/odata/v1/Products?\\$filter=\(\(ContentDate/Start%20ge%202024-02-13T00:00:00.000Z%20and%20ContentDate/Start%20le%202024-02-19T23:59:59.999Z\)%20and%20\(Online%20eq%20true\)%20and%20\(\(\(\(Collection/Name%20eq%20SENTINEL-1%27\)%20and%20\(\(\(Attributes/Odata.CSC.StringAttribute/any\(i0:i0/Name%20eq%20productType%27%20and%20i0/Value%20eq%20GRD%27\)\)\)\)\)\)\)\)&\\$expand=Attributes&\\$expand=Assets&\\$orderby=ContentDate/Start%20asc&\\$top=20](https://datahub.creodias.eu/odata/v1/Products?$filter=((ContentDate/Start%20ge%202024-02-13T00:00:00.000Z%20and%20ContentDate/Start%20le%202024-02-19T23:59:59.999Z)%20and%20(Online%20eq%20true)%20and%20((((Collection/Name%20eq%20SENTINEL-1%27)%20and%20(((Attributes/Odata.CSC.StringAttribute/any(i0:i0/Name%20eq%20productType%27%20and%20i0/Value%20eq%20GRD%27))))))))&$expand=Attributes&$expand=Assets&$orderby=ContentDate/Start%20asc&$top=20), access: February 20th 2024

place orders till all of these products will be ordered (assuming you did set *new_orders* parameter to *true*).

2.2.1. Use case

Let's say that you provided these parameters into *order_body.json*, *query_details.json* and *keycloak_ordering.json*.

```
{
  "WorkflowName": "card_bs",
  "Name": "your_card_bs_order"
}
```

order_body.json

```
{
  "parallel_quota": 5,
  "query_url":
    "https://datahub.creodias.eu/odata/v1/Products?$filter=((ContentDate/Start%20ge%202024-02-13T00:00:00.000Z%20and%20ContentDate/Start%20le%202024-02-19T23:59:59.999Z)%20and%20(Online%20eq%20true)%20and%20((((Collection/Name%20eq%20%27SENTINEL-1%27)%20and%20(((Attributes/odata.CSC.StringAttribute/any(iO:iO/Name%20eq%20%27productType%27%20and%20iO/Value%20eq%20%27GRD%27))))))&$expand=Attributes&$expand=Assets&$orderby=ContentDate/Start%20asc&$top=20",
  "new_orders": true
}
```

query_details.json

```
{
  "client_id": "CLOUDFERRO_PUBLIC",
  "username": "username@cloudferro.com",
  "password": "mysecretpassword123",
  "client_secret": "",
  "keycloak_address": "https://identity.cloudferro.com/auth/realms/Creodias-new/protocol/openid-connect/token?",
  "host": "datahub.creodias.eu",
  "totp_code": "XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX"
}
```

keycloak_ordering.json

You may have noticed, that in ref. *Picture 2* we had 20 products listed on the page. More products do exist, but for performance reasons they are hidden in the subsequent pages. You'd have to take my word for it, but these products down below were listed on this particular page.

S1A_IW_GRDH_1SDV_20240213T001328_20240213T001357_052533_065AA7_B920·SAFE
S1A_IW_GRDH_1SDV_20240213T001357_20240213T001422_052533_065AA7_E26B·SAFE
S1A_IW_GRDH_1SDV_20240213T001422_20240213T001447_052533_065AA7_88DE·SAFE
S1A_IW_GRDH_1SDV_20240213T001447_20240213T001512_052533_065AA7_278F·SAFE
S1A_IW_GRDH_1SDV_20240213T001512_20240213T001537_052533_065AA7_7A3D·SAFE
S1A_IW_GRDH_1SDV_20240213T001537_20240213T001602_052533_065AA7_D42A·SAFE
S1A_IW_GRDH_1SDV_20240213T001602_20240213T001627_052533_065AA7_9C0C·SAFE
S1A_IW_GRDH_1SDV_20240213T001627_20240213T001652_052533_065AA7_178A·SAFE
S1A_IW_GRDH_1SDV_20240213T001652_20240213T001717_052533_065AA7_CA28·SAFE
S1A_IW_GRDH_1SDV_20240213T001717_20240213T001742_052533_065AA7_0E30·SAFE
S1A_IW_GRDH_1SDV_20240213T001742_20240213T001807_052533_065AA7_A7A6·SAFE
S1A_IW_GRDH_1SDV_20240213T001807_20240213T001832_052533_065AA7_C586·SAFE
S1A_IW_GRDH_1SDV_20240213T001832_20240213T001857_052533_065AA7_2722·SAFE
S1A_IW_GRDH_1SDV_20240213T001857_20240213T001922_052533_065AA7_8508·SAFE
S1A_IW_GRDH_1SDV_20240213T001922_20240213T001947_052533_065AA7_CB45·SAFE
S1A_IW_GRDH_1SDV_20240213T001947_20240213T002012_052533_065AA7_567F·SAFE
S1A_IW_GRDH_1SDV_20240213T002012_20240213T002037_052533_065AA7_3F55·SAFE
S1A_IW_GRDH_1SDV_20240213T002037_20240213T002102_052533_065AA7_45F3·SAFE
S1A_IW_GRDH_1SDV_20240213T002102_20240213T002127_052533_065AA7_57DE·SAFE
S1A_IW_GRDH_1SDV_20240213T002127_20240213T002152_052533_065AA7_C849·SAFE

Products listed in `query_url` (colors represent orders – batches of 5 products)

You might be wondering, where did these products come from? Well, they've been extracted from `[“value”][x][“Name”]` (ref. Picture 3).



Picture 3 - Random product from `query_url`

Now that we know where these products actually come from, let's see how the script will behave, when choosing `create-order-with-query` option. First, it goes to the `query_url` and looks there for products. Like we've noticed before, there are 3775 products in total, but 20 listed on this one page. The script will look for products on this specific page till all of them will be ordered and then go to the next page. We will be placing orders of 5 products each time, because we have `parallel_quota` parameter set to 5.

Ordering Script

```
{  
  "WorkflowName": "card_bs",  
  "Name": "your_card_bs_order",  
  "IdentifierList": [  
    "S1A_IW_GRDH_1SDV_20240213T001328_20240213T001357_052533_065AA7_B920·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001357_20240213T001422_052533_065AA7_E26B·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001422_20240213T001447_052533_065AA7_88DE·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001447_20240213T001512_052533_065AA7_278F·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001512_20240213T001537_052533_065AA7_7A3D·SAFE"  
  ]  
}
```

First order

```
{  
  "WorkflowName": "card_bs",  
  "Name": "your_card_bs_order",  
  "IdentifierList": [  
    "S1A_IW_GRDH_1SDV_20240213T001537_20240213T001602_052533_065AA7_D42A·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001602_20240213T001627_052533_065AA7_9C0C·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001627_20240213T001652_052533_065AA7_178A·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001652_20240213T001717_052533_065AA7_CA28·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001717_20240213T001742_052533_065AA7_0E30·SAFE"  
  ]  
}
```

Second order

```
{  
  "WorkflowName": "card_bs",  
  "Name": "your_card_bs_order",  
  "IdentifierList": [  
    "S1A_IW_GRDH_1SDV_20240213T001742_20240213T001807_052533_065AA7_A7A6·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001807_20240213T001832_052533_065AA7_C586·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001832_20240213T001857_052533_065AA7_2722·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001857_20240213T001922_052533_065AA7_8508·SAFE",  
    "S1A_IW_GRDH_1SDV_20240213T001922_20240213T001947_052533_065AA7_CB45·SAFE"  
  ]  
}
```

Third order

```
{
  "WorkflowName": "card_bs",
  "Name": "your_card_bs_order",
  "IdentifierList": [
    "S1A_IW_GRDH_1SDV_20240213T001947_20240213T002012_052533_065AA7_567F·SAFE",
    "S1A_IW_GRDH_1SDV_20240213T002012_20240213T002037_052533_065AA7_3F55·SAFE",
    "S1A_IW_GRDH_1SDV_20240213T002037_20240213T002102_052533_065AA7_45F3·SAFE",
    "S1A_IW_GRDH_1SDV_20240213T002102_20240213T002127_052533_065AA7_57DE·SAFE",
    "S1A_IW_GRDH_1SDV_20240213T002127_20240213T002152_052533_065AA7_C849·SAFE"
  ]
}
```

Fourth order

At this point, all products were ordered from the *query_url*, but there are much more. You might remember that there were 3775 products to begin with. So what happens when we order all products from the page? Well, we simply go to the next one (*@odata.nextLink* in this case) and repeat the cycle.



Picture 4 - URL to the next page

The script will go on and on till all products will be ordered.

2.3. create-order-with-query --hours=x

The third and final option looks pretty intriguing, doesn't it? You may be asking yourself, what exactly is `--hours`? Well, by providing this option you're specifying from which time frame you'd like to get the products from. Let's say that you've executed the script by `create-order-with-query --hours=12`

and it's currently *February 19th, 10:57 AM*. The script will adjust the time in the query accordingly. Let's assume that you've provided this query².

The script will adjust this fragment: *(ContentDate/Start%20ge%202024-02-13T00:00:00.000Z%20and%20ContentDate/Start%20le%202024-02-19T23:59:59.999Z)*. After executing the script by providing *create-order-with-query --hours=12*, you will get *(ContentDate/Start%20ge%202024-02-18T22:57:14.635804Z%20and%20ContentDate/Start%20le%202024-02-19T10:57:14.635804Z)*. As you can see, the last bit was changed to *2024-02-19T10:57:14.635804Z*, because at the time of executing the script, it was *February 19th, 10:57 AM*. Let's now look into this first fragment: *2024-02-18T22:57:14.635804Z*. Why does it say *February 18th, 11:57PM*? Well, it's simply because if you'd look back 12 hours from *February 19th 10:57AM*, you'd get *February 18th 11:57PM*. In other words *February 19th, 10:57AM – 12 hours = February 18th, 11:57PM*.

Ultimately, after executing the script by *create-order-with-query --hours=12*, the query will be changed to this one³. The rest looks the same as in 2.2 (*create-order-with-query* command).

3. CONFIGURATION

In this chapter we're going to focus on the configuration aspect of ordering script. In order to run the script, you have to do a little configuration by yourself. You don't have to configure every file in *jsons* folder. Depending on your needs, your configuration process may vary (ref. *Table 1*).

File Name	When to configure
keycloak_catalogue.json	Only if you'd like to create order with a query AND your provided <i>query_url</i> parameter requires authentication (e.g. <i>hda</i> catalogue)
keycloak_ordering.json	Always

² [https://datahub.creodias.eu/odata/v1/Products?\\$filter=\(\(ContentDate/Start%20ge%202024-02-13T00:00:00.000Z%20and%20ContentDate/Start%20le%202024-02-19T23:59:59.999Z\)%20and%20\(Online%20eq%20true\)%20and%20\(\(\(\(Collection/Name%20eq%20%27SENTINEL-1%27\)%20and%20\(\(\(Attributes/Odata.CSC.StringAttribute/any\(i0:i0/Name%20eq%20%27productType%27%20and%20i0/Value%20eq%20%27GRD%27\)\)\)\)\)\)\)\)&\\$expand=Attributes&\\$expand=Assets&\\$orderby=ContentDate/Start%20asc&\\$top=20](https://datahub.creodias.eu/odata/v1/Products?$filter=((ContentDate/Start%20ge%202024-02-13T00:00:00.000Z%20and%20ContentDate/Start%20le%202024-02-19T23:59:59.999Z)%20and%20(Online%20eq%20true)%20and%20((((Collection/Name%20eq%20%27SENTINEL-1%27)%20and%20(((Attributes/Odata.CSC.StringAttribute/any(i0:i0/Name%20eq%20%27productType%27%20and%20i0/Value%20eq%20%27GRD%27))))))))&$expand=Attributes&$expand=Assets&$orderby=ContentDate/Start%20asc&$top=20), access: February 20th 2024

³ [https://datahub.creodias.eu/odata/v1/Products?\\$filter=\(\(ContentDate/Start%20ge%202024-02-18T22:57:14.635804Z%20and%20ContentDate/Start%20le%202024-02-19T10:57:14.635804Z\)%20and%20\(Online%20eq%20true\)%20and%20\(\(\(\(Collection/Name%20eq%20%27SENTINEL-1%27\)%20and%20\(\(\(Attributes/Odata.CSC.StringAttribute/any\(i0:i0/Name%20eq%20%27productType%27%20and%20i0/Value%20eq%20%27GRD%27\)\)\)\)\)\)\)\)&\\$expand=Attributes&\\$expand=Assets&\\$orderby=ContentDate/Start%20asc&\\$top=20](https://datahub.creodias.eu/odata/v1/Products?$filter=((ContentDate/Start%20ge%202024-02-18T22:57:14.635804Z%20and%20ContentDate/Start%20le%202024-02-19T10:57:14.635804Z)%20and%20(Online%20eq%20true)%20and%20((((Collection/Name%20eq%20%27SENTINEL-1%27)%20and%20(((Attributes/Odata.CSC.StringAttribute/any(i0:i0/Name%20eq%20%27productType%27%20and%20i0/Value%20eq%20%27GRD%27))))))))&$expand=Attributes&$expand=Assets&$orderby=ContentDate/Start%20asc&$top=20), access: February 20th 2024

File Name	When to configure
order_body.json	Always
order_details.json	Only if you'd like to create order with a body
query_details.json	Only if you'd like to create order with a query

Table 1 - configuration of *.json files

Now, let's look into these files in more detail.

3.1. keycloak_catalogue.json

keycloak_catalogue.json takes 5 parameters as an input – *client_id*, *username*, *password*, *client_secret* and *keycloak_address*.

```
{
  "client_id": "string",
  "username": "string",
  "password": "string",
  "client_secret": "string",
  "keycloak_address": "string"
}
```

keycloak_catalogue.json

You don't have to provide any values in this file as long as you're not using a query, which requires authentication, such as *hda*. *Datahub* catalogue will work just fine without providing any parameters in this file, so for most cases you really shouldn't stress about this.

3.2. keycloak_ordering.json

This file takes 7 parameters as an input – *client_id*, *username*, *password*, *client_secret*, *keycloak_address*, *host* and *totp_code*.

```
{
  "client_id": "string",
  "username": "string",
  "password": "string",
  "client_secret": "string",
  "keycloak_address": "string",
  "host": "string",
  "totp_code": "string"
}
```

keycloak_ordering.json


Without providing proper values into this file, the script won't work. *totp_code* is an optional parameter, but if you're using *Creodias* it is required. Here's how this file looks like for *Creodias* use case:


```
{
  "client_id": "CLOUDFERRO_PUBLIC",
  "username": "username@cloudferro.com",
  "password": "mysecretpassword123",
  "client_secret": "",
  "keycloak_address": "https://identity.cloudferro.com/auth/realms/Creodias-new/protocol/openid-connect/token?",
  "host": "datahub-creodias.eu",
  "totp_code": "XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX"
}
```

keycloak_ordering.json for Creodias

It goes without saying, but you need to change *username* and *password* parameters with your own *Creodias* credentials and on top of that you also have to provide proper *totp_code*. It consists of 32 characters (or possibly 39 if you use space each time after 4 characters). You may be scratching your head right now and thinking how can you get this code? I'm going to assume that you're already an existing user of *Creodias*, but you don't know your *TOTP secret code* yet. The most straightforward way of going about this is to simply reset your code. All you have to do is navigate to the *TOTP management console*⁴ and remove your existing secret key. Once you've removed your secret key, click on *Set up authenticator application*. You should be prompted with this window down below.

⁴ <https://identity.cloudferro.com/auth/realms/Creodias-new/account/#/security/signingin>, access: February 20th 2024




 You need to set up Mobile Authenticator to activate your account.

1. Install one of the following applications on your mobile:

- FreeOTP
- Google Authenticator

2. Open the application and scan the barcode:



Unable to scan?

3. Enter the one-time code provided by the application and click Submit to finish the setup.

Provide a Device Name to help you manage your OTP devices.

One-time code *

Device Name

Submit

Picture 5 - Creodias authentication page

Click on *Unable to scan?*, save your secret key in a secure place and complete your form (ref. Picture 6).

CREODIAS
powered by CloudFerro

⚠ You need to set up Mobile Authenticator to activate your account.

1. Install one of the following applications on your mobile:
 - FreeOTP
 - Google Authenticator
2. Open the application and enter the key:
 O5XU KR2K K5KF E4TJ M5QW 2RLT JVDH O6TK
Scan barcode?
3. Use the following configuration values if the application allows setting them:
 - Type: Time-based
 - Algorithm: SHA1
 - Digits: 6
 - Interval: 30
4. Enter the one-time code provided by the application and click Submit to finish the setup.

 Provide a Device Name to help you manage your OTP devices.

One-time code *

Device Name

Submit

Picture 6 - Getting the secret key from Creodias

Do not scan your *QR code*, because it uses a different secret code to the one presented in the *Unable to scan?* page. It's really important, because once you start using it, it will be the one that works for your account. If you'd like to learn more about how *Creodias* handles 2FA, you might find this⁵ and that⁶ useful.

⁵ <https://creodias.docs.cloudferro.com/en/latest/eodata/Use-Python-to-automate-generating-API-tokens-for-accessing-and-downloading-EODATA-when-2FA-is-enabled-on-Creodias.html>, access: February 20th 2024

⁶ <https://creodias.docs.cloudferro.com/en/latest/eodata/Use-Python-to-automate-generating-API-tokens-for-accessing-and-downloading-EODATA-when-2FA-is-enabled-on-Creodias.html>, access: February 20th 2024

3.3. order_body.json

In this file you're specifying how your order body will look like. While this file takes all kinds of parameters as an input, only a few of them are required, namely *WorkflowName*, *Name* and *IdentifierList*. The last parameter is pretty special. Depending on your needs you may/may not want to provide *IdentifierList* parameter. It depends if you want to hardcode products by yourself or provide a query with products that you'd like to order. In this second case you can skip *IdentifierList* parameter, because the script will get products from your given query in *query_details.json* anyway. Some processors also require some special parameters, which you also have to provide e.g. private processors require putting s3 credentials into *WorkflowOptions*. Script will give you a run-down and inform you which parameters are missing.

```
{
  "Name": "string",
  "Priority": int,
  "WorkflowName": "string",
  "NotificationEndpoint": "string",
  "NotificationEpUsername": "string",
  "NotificationStatus": "string",
  "WorkflowOptions": [
    {
      "Name": "string",
      "Value": "string"
    }
  ],
  "IdentifierList": [
    "string"
  ],
  "BatchSize": int,
  "BatchVolume": int,
  "NotificationEpPassword": "string",
  "SendIntermediateNotifications": bool
}
```

order_body.json

Here's how order body could look like if you'd want to create *card_bs* order by hardcoding all products.


```
{
  "WorkflowName": "card_bs",
  "Name": "your_card_bs_order",
  "IdentifierList": [
    "51B_IW_GRDH_1SDV_20211223T043834_20211223T043859_030148_03998E_B01A",
    "51B_IW_GRDH_1SDV_20211223T034414_20211223T034439_030147_039989_16DF",
    "51B_EW_GRDM_1SDH_20211222T173128_20211222T173205_030141_039962_217D"
  ]
}
```

order_body.json with hardcoded card_bs products

Here's yet another example. Let's say that you'd want to create card_coh12_public order.

```
{
  "WorkflowName": "card_coh12_public",
  "Name": "your_card_coh12_order",
  "IdentifierList": [
    "51A_IW_SLC_1SDV_20240101T010035_20240101T010102_051906_064573_D8CB"
  ]
}
```

order_body.json with hardcoded card_coh12_public products

3.4. order_details.json

This file takes 2 parameters as an input, namely *parallel_quota* and *new_orders*. You're specifying values in here, if you'd like to create order by hardcoding products in a body. In this file you're giving details about your order.

```
{
  "parallel_quota": int,
  "new_orders": bool
}
```

order_details.json

parallel_quota is a limit of products per one order. E.g. if you provided 5 products in *IdentifierList*, but you set *parallel_quota* parameter to 2, then here's how order creation process would look like:

- first order – place an order for the first 2 products
- second order – place an order for the next 2 products
- third order – place an order for the last product

This would only look like this if you set *new_orders* parameter to *true*. If you'd set it to *false*, then you would create only one order with 2 products. *new_orders* parameter essentially determines if new orders will be placed once your parallel quota is reached. Here's how *order_details.json* looks like if you want to continue to create new orders with batches of 5 products.

```
{
  "parallel_quota": 5,
  "new_orders": true
}
```

order_details.json for the use case of creating new orders with batches of 5 products

3.5. query_details.json

query_details.json takes 3 parameters as an input, namely *parallel_quota*, *query_url* and *new_orders*. You're specifying values in here, if you'd like to create order by a query. This file follows similar principles to *order_details.json*, except for one additional parameter that doesn't exist in *order_details.json*, namely *query_url*, which is a *URL* to the catalogue.

```
{
  "parallel_quota": int,
  "query_url": "string",
  "new_orders": bool
}
```

query_details.json

Here's how that file could look like in practice. In this example, the script will look for products in a given query and continue with placing new orders with batches of 5 products each.

```
{
  "parallel_quota": 5,
  "query_url":
    "https://datahub.creodias.eu/odata/v1/Products?$filter=((ContentDate/Start%20ge%202024-02-10T00:00:00.000Z%20and%20ContentDate/Start%20le%202024-02-16T23:59:59.999Z)%20and%20(Online%20eq%20true)%20and%20((((Collection/Name%20eq%20%27SENTINEL-1%27)%20and%20(((Attributes/odata.CSC.StringAttribute/any(iO:iO/Name%20eq%20%27productType%27%20and%20iO/Value%20eq%20%27GRD%27))))))&$expand=Attributes&$expand=Assets&$orderby=ContentDate/Start%20asc&$top=20",
  "new_orders": true
}
```

query_details.json for the use case of creating new orders with batches of 5 products from the given query

You might be thinking, how can you generate a query? Well, you can do that via explore.creodias.eu (ref. *Picture 7*). Remember to filter by proper products, because some processors require special product types as an input.

Enter full product name...
~
=

Availability: ⓘ
☒ Immediate
☐ To order
☐ All data

Imagery types: ⓘ
☐ Optical
☐ Thermal
☐ Radar
☐ Altimetry
☐ DEM
☐ Land cover

Cloud cover:
Type cloud coverage... %

Type of dates:
Sensing dates

Date range:
14/02/2024 – 20/02/2024

Select mission: ⓘ

☐ SENTINEL-1

☐ SENTINEL-1-RTC

☒ SENTINEL-2

Search

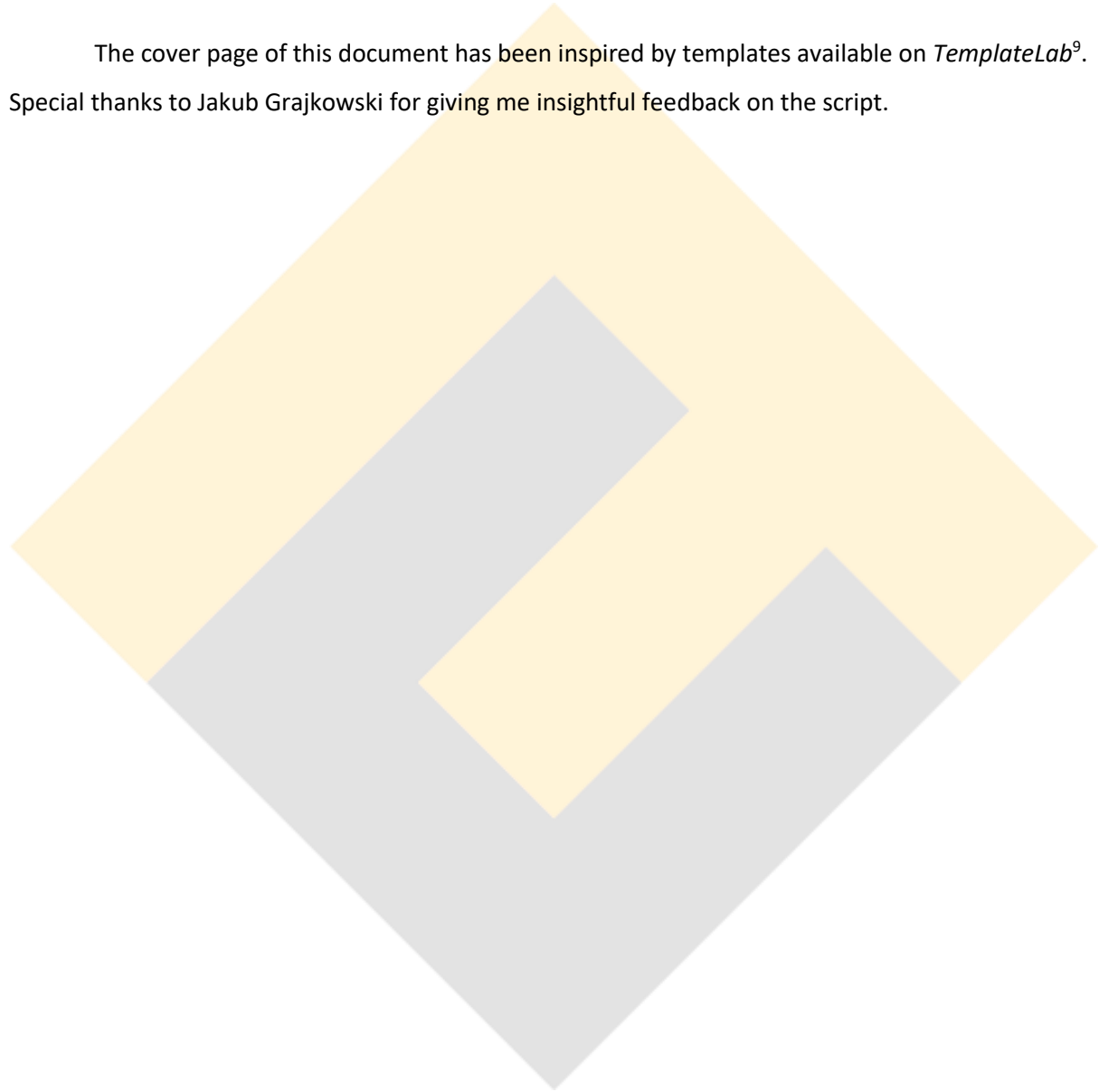
Copy query
Reset all

Picture 7 - Generating a query from explore.creodias.eu

4. FINAL REMARKS

If you have some doubts or question about the project, please visit documentation available on *Gitlab*⁷. There are a few things that I haven't covered in here, so you might find looking into docs helpful. You might not have authorization to visit this page. If that's the case, you may visit *CloudFerro's* official *Github* account⁸ and check if the script is available in there. As I'm writing this, there are plans on publishing this script on the official *Github* account in the future.

The cover page of this document has been inspired by templates available on *TemplateLab*⁹. Special thanks to Jakub Grajkowski for giving me insightful feedback on the script.



⁷ https://gitlab.cloudferro.com/processing/ordering_script, access: February 20th, 2024

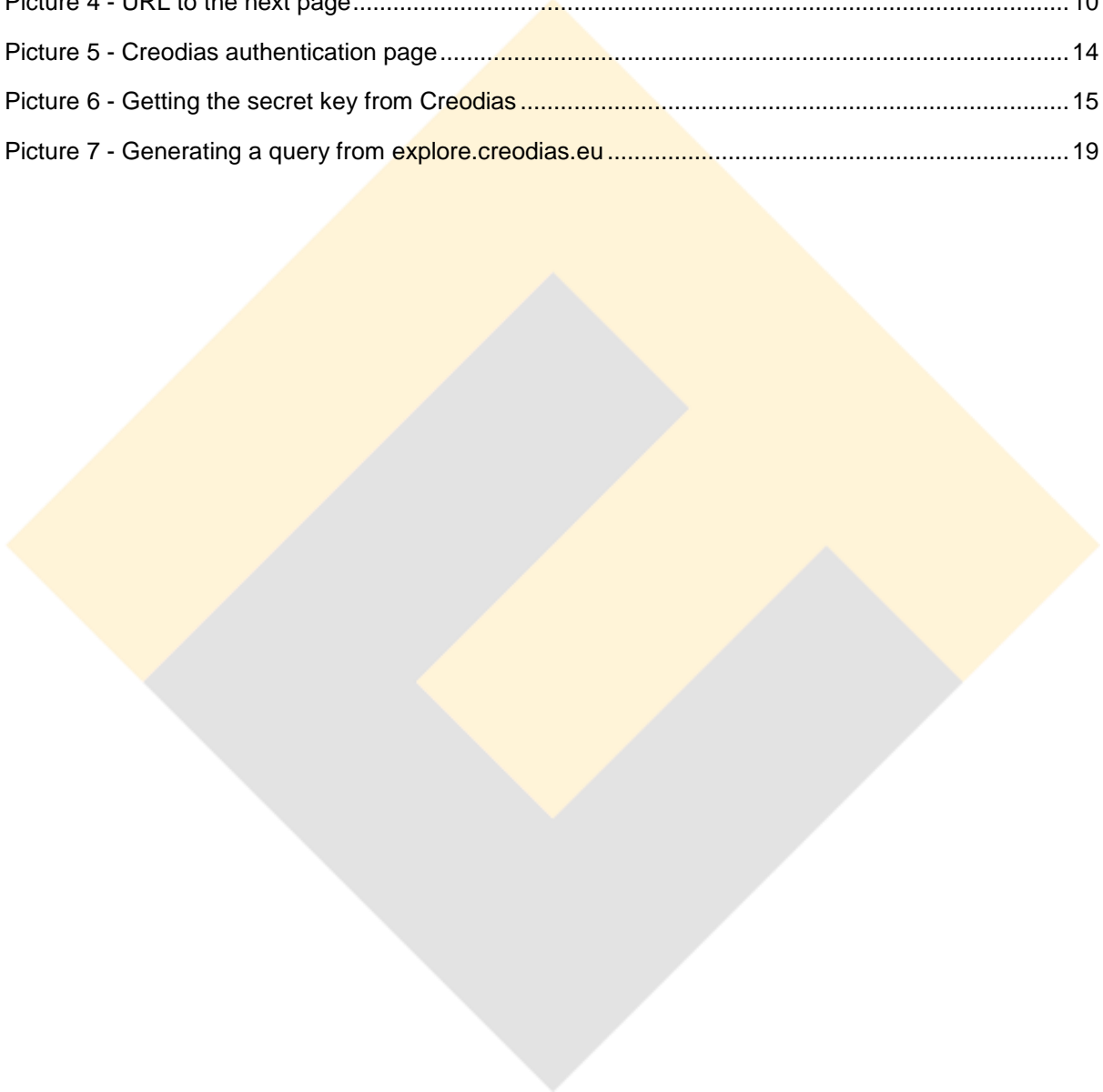
⁸ <https://github.com/CloudFerro>, access: February 20th, 2024

⁹ <https://templatelab.com/cover-page-templates/>, access: February 20th, 2024

TABLE OF FIGURES

PICTURES

Picture 1 – Running the script with a GUI	3
Picture 2 - query_url output.....	6
Picture 3 - Random product from query_url.....	8
Picture 4 - URL to the next page.....	10
Picture 5 - Creodias authentication page.....	14
Picture 6 - Getting the secret key from Creodias.....	15
Picture 7 - Generating a query from explore.creodias.eu	19



TABLES

Table 1 - configuration of *.json files 12





CloudFerro