convercus

Coupon Api Implementation Manual v1.1.0

convercus

POINT 4MORE

Table of Content

1 COUPON API IMPLEMENTATION MANUAL 3

2 CHANGE LOG 3

3 AUTHENTICATION 4

4 USER / ACCOUNT IDENTIFICATION 4

4.1 Card identification 4

4.2 User Search 5

4.3 Getting Account Details 7

4.3.1 Getting basic Account Information 8

4.3.2 Getting Membership Details 9

4.3.3 Getting User Details 10

5 COUPONING 11

5.1 Getting Coupon Information 11

5.1.1 Get all account-coupons 11

5.1.2 Get information for a specific account-coupon 15

5.1.3 Get AccountCoupon Details 15

5.2 Coupon Activation / Deactivation 19

5.3 Coupon Redemption 20

5.3.1 Note about Extra-Incentivation Coupons 21

1 Coupon Api Implementation Manual

This manual describes how to implement an interface for couponing with Convercus. With this documentation, it should be more intuitive and straight-forward to map all relevant information and processes to calls of the Convercus Api.
Technical information about the Convercus Api can be found at the following sites

- https://staging.convercus.io/api-docs/swagger-ui.html (Staging Environment)
- https://api.convercus.io/api-docs/swagger-ui.html (Production Environment)

and on the Developer Page

- https://developer.convercus.io/

2 Change Log

Version	Change Date	Change Log
v1.1.0	2020-11-24	• Adding Domain Search to description • Minor corrections
v1.0.0	2020-06-10	• Initial document

3 Authentication

Every request requires a JWT-Token for authentication. The token can be obtained with the following request:

```
curl --location --request POST '{{api_url}}/auth/login' \
--header 'Content-Type: application/json' \
--data-raw '{
  "org": "{{org}}",
  "userName": "{{userName}}",
  "password": "{{password}}"
}'
```

where the following variables have been used:

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)
`{{org}}`	Organization code for correct mapping. This value will be given by Convercus.
`{{userName}}`	User-Name of the api-user. This value will be given by Convercus.
`{{password}}`	User-Password of the api-user. This value will be given by Convercus.

The JWT-Token can be found in the body of the response.
Note, that the token expires after 24 hours. To have a valid token at all times, it is necessary to generate the token on a regular basis.

4 User / Account Identification

For all loyalty-related processes it is important to identify the person the system is interacting with. The central object for this identification is the account you can collect points on. This account can be linked to user-data, transactions, bookings, etc. Every account has an unique identifier, the `account_id` (e.g. `7d123457-bfa1-4a83-8213-123456789763`), which is the technical ID all those connections are made with.
Additionally, every account can have multiple identifiers (i.e. card-codes, external identification-codes, etc.), which allow to make a connection to an account without the need to extract the account-id. We will explore both identifier-related api-calls and account-id-related api-calls in this chapter. The handling in later chapters in anlogue.

4.1 Card identification

All connections of transactions, bookings, account-coupons, etc. to accounts can be performed using the `account_id` (or alternatively the identifier codes). Thus, there has to be a mechanism to identify the account you want to connect to.

This is usually done, by scanning / typing / etc. the predefined identifier code (card number) of the customer. This card code can then be used in the api analogously to the account-id (see in the respective chapters for examples).
Note, that for this process, it is not necessary to get any user-information. It is possible to perform the whole purchase and payment process without requesting additional user-information (like name, birthdate, optins, etc.) as only the account-identification (account_id or identifier-code) is important for the relevant api processes. If there is no need to view user-data on the cash register, you can simplify and speed up the whole process by skipping it altogether.

4.2 User Search

If a registered customer forgot to bring his card, there is the option to search for him via search api. This api is performing an elastic search over various domain objects (including accounts) and responds an array with results. You can perform an elastic search in the following way

```
curl --location --request POST '{{api_url}}/search' \
--header 'Authorization: {{jwt_token}}' \
--header 'interaction-id: {{interactionId}}' \
--header 'Content-Type: application/json' \
--data-raw '{{body}}'
```

with variables

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)
`{{jwt_token}}`	JWT-token, which has been generated by authentication.
`{{interactionId}}`	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.
`{{body}}`	Body with search parameters, specified in the following.

The body with search parameters looks like this:

```
{
  "searchTerm": "Example",
  "type": "ACCOUNT"
}
```

with

Attribute	Description	Relevance
`searchTerm`	Text to be matched with indexed search fields. Multiple values can be separated by spaces (e.g. `FirstName LastName`). The following fields are indexed • `account.identifier` • `user.givenName` • `user.familyName` • `user.city` • `user.zipCode` • `user.emailAddress` • `user.birthdate` (`YYYY-MM-DD`), not compatible with multiple search values	**Mandatory**
`type`	Domain filtering.	**Optional** Filtering to `ACCOUNT` is strongly recommended.

In general, this api may search over various domains, giving results like this


```
{
    "searchTerm": "Example",
    "nrOfResults": 3,
    "results": {
        "COUPON": [
            {
                "preview": "Title: Title of Coupon,Type: REWARD",
                "refId": "a1e52b7a-5cd4-4580-a0f7-7f602b27ba6e",
                "type": "COUPON"
            },
            {
                "preview": "Title: Title of Coupon,Type: DISCOUNT",
                "refId": "abad53e5-bd86-48df-858c-1bcce3af41fb",
                "type": "COUPON"
            }
        ],
        "ACCOUNT": [
            {
                "preview": "name: Name of Person,email: E-Mail of Person",
                "refId": "2f081e91-1346-4ef6-82b2-fcdecd3c190b",
                "type": "ACCOUNT"
            }
        ]
    }
}
```


Important Note about Filtering to Domains:

Usually, the `ACCOUNT` domain search is the weapon of choice here (as we are normally not interested in information about backend settings here). Applying the filtering on `ACCOUNT` responds the following result (same setup as before, but filtered):


```
{
    "searchTerm": "Example",
```

```
    "nrOfResults": 1,
    "results": {
        "ACCOUNT": [
            {
                "preview": "name: Name of Person,email: E-Mail of Person",
                "refId": "2f081e91-1346-4ef6-82b2-fcdecd3c190b",
                "type": "ACCOUNT"
            }
        ]
    }
}
```


After identifying the user, the `refId` of the correct `ACCOUNT`-result can be used as `accountId` for all further processes.

4.3 Getting Account Details

If you are interested in more detail about the person standing at the POS, you can use an identifier code or `accountid` to get more information about the account like its bookings, transactions, current balance, membership information or user-data. All those options are explained in Swagger Documentation (spec: account). We will focus here on personal user-data.
Note, that due to structure of the platform, there are essentially three objects which contain information about the user, his account and membership.

- **account:**
  - This is the central element of the loyalty system.
  - Every account has its unique `accountId`. All relevant loyalty-processes can be linked to this id.
  - N identifiers of different `id-type` may serve as additional external identifiers for an account. Usually, these identifiers are Cardcodes or external numbers (like a online-shop-id).
  - Accounts can be anonymous, if they don't have user-information connected (see membership).
- **user:**
  - This is the object which contains personal user-data like name, address, etc.
  - User data may be created independently from an account. Without the connection to an account (see membership) however, there is no way to interact with this data in a loyalty context (e.g. you cannot earn points on a user, but an account.
- **membership:**
  - This object connects an account to a user object.
  - The creation of a membership is typically the result of a completed registration.
  - Optins for the program are related to this object.

Depending on the set of information you are interested in, you may need to get all of them. We will explain a straight-forward way to do this in the following.

4.3.1 GETTING BASIC ACCOUNT INFORMATION

As stated out before, the account is the center of the whole loyalty system. The `accountId` can be used as common connection id for practically all loyalty connections.
You can get the basic account information with the following request:

```
curl --location --request GET '{{api_url}}/accounts/{{accountId}}' \
--header 'Authorization: {{jwt_token}}' \
--header 'interaction-id: {{interactionId}}' \
--header 'Content-Type: application/json' \
--header 'id-type: {{idType}}' \
```

with variables

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)
`{{accountId}}`	ID of the account that will receive the transaction. The ID has to be given in the format, that is dictated by the `{{idType}}`.
`{{jwt_token}}`	The JWT-token, which has been generated by authentication.
`{{interactionId}}`	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.
`{{idType}}`	Identifier-Type (corresponding to the `{{accountId}}`). **Available values:** • `APPCODE` (e.g. `A1B2C3D4E5`) • `CARDCODE` (e.g. `V1W2X3Y4Z5`) • `EXTERNALCODE` (e.g. `123456780123`) • `ID` (account-identifier, e.g. `7d123457-bfa1-4a83-8213-123456789763`)

Note, that it's possible to get the account-object using the identifier code (e.g. `{{idType}}=CARDCODE`, `{{accountId}}=V1W2X3Y4Z5`) or the `accountId` itself (e.g. `{{idType}}=ID`, `{{accountId}}=7d123457-bfa1-4a83-8213-123456789763`). Thus, the request may also be used to extract an `accountId` from a given Identifier.

The account-object per se is rather slender, only containing an id (`accountId`), a program reference and a status of the account.

```
{
  "id": "550e8400-e29b-11d4-a716-446655440000",
  "program": "Pgr-A",
  "status": "ACTIVE"
}
```

If your program allows deactivation or locking of accounts, you should make sure, that accounts that don't have the status `ACTIVE` cannot proceed with the following earn- and burn-processes. Deleted Accounts will not be responded at all.

4.3.2 GETTING MEMBERSHIP DETAILS

If you are interested in optins and / or user data of a given account, you need to check if the account has an active membership by requesting

```
curl --location --request GET
'{{api_url}}/accounts/{{accountId}}/memberships' \
--header 'Authorization: {{jwt_token}}' \
--header 'interaction-id: {{interactionId}}' \
--header 'Content-Type: application/json' \
--header 'id-type: {{idType}}' \
```

with variables

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)
`{{accountId}}`	ID of the account that will receive the transaction. The ID has to be given in the format, that is dictated by the`{{idType}}`,
`{{jwt_token}}`	The JWT-token, which has been generated by authentication.
`{{interactionId}}`	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.
`{{idType}}`	Identifier-Type (corresponding to the `{{accountId}}`). **Available values:** • `APPCODE` (e.g. `A1B2C3D4E5`) • `CARDCODE` (e.g. `V1W2X3Y4Z5`) • `EXTERNALCODE` (e.g. `123456780123`) • `ID` (account-identifier, e.g. `7d123457-bfa1-4a83-8213-123456789763`)

Note again, that you can use the identifier code directly (e.g. `{{idType}}=CARDCODE`, `{{accountId}}=V1W2X3Y4Z5`) at this point.

The returned object looks like this:

```
[
  {
    "accountId": "550e8400-e29b-11d4-a716-446655440000",
    "memberRole": "OWNER",
    "membershipId": "550e8400-e29b-11d4-a716-446655440000",
    "optins": [
      {
        "flag": true,
        "type": "email"
      }
    ],
    "partnerId": "550e8400-e29b-11d4-a716-446655440000",
    "userId": "550e8400-e29b-11d4-a716-446655440000"
  }
]
```

where

Attribute	Description
`accountId`	Technical account-Id connected to this membership.
`memberRole`	Role of the membership. In general, it is possible to have multiple memberships, one account owner (`"memberRole": "OWNER"`) and several `COOWNERs` or `COLLECTORs`. In the standard setup however, there is only one membership (with `"memberRole": "OWNER"`), connecting one account to one user.
`membershipId`	Technical ID of the membership.
`optins`	List of optins that this membership has. Optin names are denoted by `type`, optin values are denoted by `flag`.
`partnerId`	Technical ID of the partner, the membership is belonging to.
`userId`	Technical ID of the user connected to an account via the membership.

If no membership is returned, the returned array is empty `[]`. Depending on the specific program setup, these users may not be allowed to earn and/or burn their points if there are not registered. This logic has to be adopted here if that's the case.

4.3.3 GETTING USER DETAILS

Given the `userId` of the membership, user data can be received by requesting:

```
curl --location --request GET '{{api_url}}/users/{{userId}}' \
--header 'Authorization: {{jwt_token}}' \
--header 'interaction-id: {{interactionId}}' \
--header 'Content-Type: application/json' \
```

with

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)

{{userId}}	Technical User-ID of the user (usually given by the membership connection).
{{jwt_token}}	The JWT-token, which has been generated by authentication.
{{interactionId}}	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.

An exemplary response looks like this

```
{
  "birthDate": "1965-03-05",
  "city": "München",
  "countryCode": "DE",
  "customProperties": [
    {
      "name": "string",
      "value": "string"
    }
  ],
  "emailAddress": "member1@convercus.de",
  "familyName": "Mustermann",
  "genderCode": "MALE",
  "givenName": "Max",
  "phone": 654324563,
  "streetHouseNo": "Bahnhofstraße 1",
  "userId": "550e8400-e29b-11d4-a716-446655440000",
  "zipCode": 80469
}
```

Note, that the content of this response may differ with the program. There may be multiple `customProperties`, which are completely program-specific. Furthermore, it is possible that in future versions, the response will be expanded by more fields, so you should make sure to be able to accept more output.

5 Couponing

A very common process in cash registers and online shops is the basic couponing. There are various options to get and redeem coupons that certain users have connected with their accounts.

5.1 Getting Coupon Information

5.1.1 GET ALL ACCOUNT-COUPONS

To view all coupons a customer has access to, the following request must be sent:

```
curl --location --request GET
'{{api_url}}/v2/accounts/{{accountId}}/accountcoupons' \
--header 'interaction-id: {{interactionId}}' \
--header 'Authorization: {{jwt_token}}' \
--header 'id-type: {{idType}}' \
--header 'Content-Type: application/json' \
```

with

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)
`{{accountId}}`	ID of the account that will receive the transaction. The ID has to be given in the format, that is dictated by the `{{idType}}`,
`{{jwt_token}}`	The JWT-token, which has been generated by authentication.
`{{interactionId}}`	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.
`{{idType}}`	Identifier-Type (corresponding to the `{{accountId}}`). **Available values:** • `APPCODE` (e.g. `A1B2C3D4E5`) • `CARDCODE` (e.g. `V1W2X3Y4Z5`) • `EXTERNALCODE` (e.g. `123456780123`) • `ID` (account-identifier, e.g. `7d123457-bfa1-4a83-8213-123456789763`)

This will return an array of all coupons accessable for the given `accountId`, e.g.


```
[
        {
        "accountCouponId":
"YmZlOGNjNzgtMDIxZC00NTI2LThjMGMtZjhhYzUwYzE3MmIxfGExNDVlMTJkLTk4NjItNDQxOC
1hM2RkLWM3NjJjNzk3ZGY2Nw",
        "accountId": "a145e12d-9862-4418-a3dd-c762c797df67",
        "couponId": "bfe8cc78-021d-4526-8c0c-f8ac50c172b1",
        "usageType": "REWARD",
        "externalReference": "ExternalReferenceExample",
        "externalCode": {
            "value": "EXCODE",
            "type": "TEXT"
        },
        "maxRedeemCount": 1,
        "maxRedeemCountGlobal": 1,
        "redeemCount": 0,
        "redeemCountGlobal": 0,
        "pointsOfRedemption": [
            {
                "unitType": "INTERACTIONPOINT",
                "value": "app"
            }
        ],
```

```
        "validFrom": "2020-11-01T00:00:00",
        "validTo": "2020-11-30T23:59:00",
        "stateLevel": null,
        "couponValue": 100,
        "customProperties": [
            {
                "name": "CustomProperty1",
                "value": "Example"
            }
        ],
        "images": [
            {
                "id": 552a1c66-45e8-442c-8fdc-29eb175f3a01,
                "name": exampleimage.jpg,
                "tags": [
                    "contentPicture"
                ],
                "path": "https://image-
repository.point4more.com/1605623285936-exampleimage.jpg"
            }
        ],
        "i18nFields": {
            "de": {
                "title": "Example Coupon",
                "subTitle": "Example subtitle",
                "description": "Example description",
                "bookingText": "Example booking text"
            }
        },
        "activation": {
            "activatable": false,
            "activated": null,
            "lastActivationDate": null
        }
    }
]
```


with

Attribute	Description
`accountCouponId`	Unique identification string generated from the `accountId` and the individual `couponId`.
`accountId`	Technical account-Id connected to this membership.
`couponId`	Unique identifier for every coupon.
`usageType`	Specifies the type of coupon. `usageType` can be one of four types: • `DISCOUNT` • `MULTIPLIER` • `EXTRAPOINT` • `REWARD`

`externalReference`	Unique Reference of the coupon, that may be interpreted by the cash register to execute predefined actions. **Note:** This value has to be stored before redemption as it will not be part of the redemption response.
`maxRedeemCount`	Maximum number of times a coupon can be redeemed for the given `accountId`.
`maxRedeemCountGlobal`	Maximum number of times a coupon can be redeemed for all accounts.
`redeemCount`	Number of times a coupon has been redeemed for given `accountId`.
`redeemCountGlobal`	Number of times a coupon has been redeemed for all accounts.
`pointsOfRedemption`	Partners and interaction points the coupon can be redeemed at.
`validFrom`	Starting date the coupon will be valid from. **Note:** If `validFrom` and `validTo` are empty, the coupon will be valid unconditionally.
`validTo`	End date the coupon will be valid until. **Note:** If `validFrom` and `validTo` are empty, the coupon will be valid unconditionally.
`couponValue`	States the registered value of the coupon. For the different types of coupons that specifies: • `DISCOUNT`: The discount in percent. • `REWARD`: Amount of points a customer will burn for claiming the reward coupon. • `EXTRA-POINTS`: The additional points that will be added after the transaction. • `MULTIPLIER`: Factor, the points will be multiplied with.
`customProperties`	Custom parameters with their respective values.
`i18nFields`	Descriptive texts of the coupon, partitioned in all generated langueages with 4 subcategories, see below for more details.
`activation`	Detailed information about the coupon's activation, partitioned in 3 subcategories, see below for more details.

The detailed attributes of `i18nFields` are (one set per language (e.g. `de`):

Attribute	Description
`title`	Title of the coupon.
`subTitle`	Additional title of the coupon.

`description`	A more expressive description of the coupon.
`bookingText`	Booking text of the coupon redemption.

The attributes of `activation` can be described as:

Attribute	Description
`activatable`	Describes if a coupon can be manually activated, where `true` indicates a possible activation and `false` a permanent coupon. **Note:** Activatable coupons are only redeemable if they have been activated.
`activated`	Indicates, if an activatable coupon is currently activated on given `accountId`. This parameter will be `null`, if the coupon is not activatable.
`lastActivationDate`	Indicates the last date the coupon was activated for given `accountId`. This parameter will be `null`, if the coupon is not activatable.

5.1.2 GET INFORMATION FOR A SPECIFIC ACCOUNT-COUPON

By encoding the `accountCouponId` as QR-Code (or similar), the required account information can be retrieved without requesting all account coupons again. By inputting the Code at the cash register, either through scanning the code or manual input, the coupon can be used directly without further requests, since the `accountCouponId` is a unique ID for the combination between `accountId` and `couponId`.

5.1.3 GET ACCOUNTCOUPON DETAILS

By using the unique `accountCouponId`, the coupons properties can be observed in greater detail.

```
curl --location --request GET
'{{api_url}}/v2/accountcoupons/{{accountCouponId}}' \
--header 'interaction-id: {{interactionId}}' \
--header 'Authorization: {{jwt_token}}' \
--header 'Content-Type: application/json' \
--header 'id-type: {{idType}}' \
```

again with

Attribute	Description
`{{accountCouponId}}`	Unique identification string generated from the `accountId` and the individual `couponId`.
`{{accountId}}`	Technical account-Id connected to this membership.
`{{jwt_token}}`	The JWT-token, which has been generated by authentication.
`{{interactionId}}`	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.

This will return the coupon properties (equivalent to the array elements of the full account-coupon view) in detail:


```
{
    "accountCouponId":
"YmZlOGNjNzgtMDIxZC00NTI2LThjMGMtZjhhYzUwYzE3MmIxfGExNDVlMTJkLTk4NjItNDQxOC1hM2RkLWM3NjJjNzk3ZGY2Nw",
    "accountId": "a145e12d-9862-4418-a3dd-c762c797df67",
    "couponId": "bfe8cc78-021d-4526-8c0c-f8ac50c172b1",
    "usageType": "REWARD",
    "externalReference": "ExternalReferenceExample",
    "externalCode": {
        "value": "EXCODE",
        "type": "TEXT"
    },
    "maxRedeemCount": 1,
    "maxRedeemCountGlobal": 1,
    "redeemCount": 0,
    "redeemCountGlobal": 0,
    "pointsOfRedemption": [
        {
            "unitType": "INTERACTIONPOINT",
            "value": "app"
        }
    ],
    "validFrom": "2020-11-01T00:00:00",
    "validTo": "2020-11-30T23:59:00",
    "stateLevel": null,
    "couponValue": 100,
    "customProperties": [
        {
            "name": "CustomProperty1",
            "value": "Example"
        }
    ],
    "images": [
        {
            "id": 552a1c66-45e8-442c-8fdc-29eb175f3a01,
            "name": exampleimage.jpg,
            "tags": [
                "contentPicture"
            ],
            "path": "https://image-repository.point4more.com/1605623285936-
exampleimage.jpg"
        }
    ],
    "i18nFields": {
        "de": {
            "title": "Example Coupon",
            "subTitle": "Example subtitle",
            "description": "Example description",
            "bookingText": "Example booking text"
        }
    },
    "activation": {
        "activatable": false,
        "activated": null,
        "lastActivationDate": null
    }
```

}

with the same attributes as before:

Attribute	Description
`accountCouponId`	Unique identification string generated from the `accountId` and the individual `couponId`.
`accountId`	Technical account-Id connected to this membership.
`couponId`	Unique identifier for every coupon.
`usageType`	Specifies the type of coupon. `usageType` can be one of four types: • `DISCOUNT` • `MULTIPLIER` • `EXTRAPOINT` • `REWARD`
`externalReference`	Unique Reference of the coupon, that can be interpreted by the cash register to execute predefined actions. **Note:** This value has to be stored before redemption as it will not be part of the redemption response.
`maxRedeemCount`	Maximum number of times a coupon can be redeemed for the given `accountId`.
`maxRedeemCountGlobal`	Maximum number of times a coupon can be redeemed for all accounts.
`redeemCount`	Number of times a coupon has been redeemed for given `accountId`.
`redeemCountGlobal`	Number of times a coupon has been redeemed for all accounts.
`pointsOfRedemption`	Partners and interaction points the coupon can be redeemed at.
`validFrom`	Starting date the coupon will be valid from. Note: If `validFrom` and `validTo` are empty, the coupon will be valid unconditionally.
`validTo`	End date the coupon will be valid until. Note: If `validFrom` and `validTo` are empty, the coupon will be valid unconditionally.

`couponValue`	States the registered value of the coupon. For the different types of coupons that specifies: • `DISCOUNT`: The discount in percent. • `REWARD`: Amount of points a customer will burn for claiming the reward coupon. • `EXTRA-POINTS`: The additional points that will be added after the transaction. • `MULTIPLIER`: Factor, the points will be multiplied with.
`customProperties`	Custom parameters with their registered values.
`i18nFields`	Descriptive texts of the coupon, partitioned in all generated langueages with 4 subcategories, see below for more details.
`activation`	Detailed information about the coupon's activation, partitioned in 3 subcategories, see below for more details.

The detailed attributes of `i18nFields` are (one set per language (e.g. `de`):

Attribute	Description
`title`	Title of the coupon.
`subTitle`	Additional title of the coupon.
`description`	A more expressive description of the coupon.
`bookingText`	Booking text of the coupon redemption.

Similarly, the attributes of `activation` can be described as:

Attribute	Description
`activatable`	Describes if a coupon can be manually activated, where `true` indicates a possible activation and `false` a permanent coupon. **Note:** Activatable coupons are only redeemable if they have been activated.
`activated`	Indicates, if an activatable coupon is currently activated on given `accountId`. This parameter will be `null`, if the coupon is not activatable.
`lastActivationDate`	Indicates the last date the coupon was activated for given `accountId`. This parameter will be `null`, if the coupon is not activatable.

5.2 Coupon Activation / Deactivation

With a unique `accountCouponId`, a specific coupon available for given account can be activated or deactivated manually. This only works for activatable coupons, marked by the property
`"activatable": true` in the accountCouponId Details (see above).

```
curl --location --request PATCH
'{{api_url}}/v2/accountcoupons/{{accountCouponId}}/activation' \
--header 'interaction-id: {{interactionId}}' \
--header 'Authorization: {{jwt_token}}' \
--header 'Content-Type: application/json' \
--header 'id-type: {{idType}}' \
--data-raw '{
  "activated": true
}'
```

with

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)
`{{accountId}}`	ID of the account that will receive the transaction. The ID has to be given in the format, that is dictated by the `{{idType}}`,
`{{jwt_token}}`	The JWT-token, which has been generated by authentication.
`{{interactionId}}`	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.
`{{idType}}`	Identifier-Type (corresponding to the `{{accountId}}`). **Available values:** • `APPCODE` (e.g. `A1B2C3D4E5`) • `CARDCODE` (e.g. `V1W2X3Y4Z5`) • `EXTERNALCODE` (e.g. `123456780123`) • `ID` (account-identifier, e.g. `7d123457-bfa1-4a83-8213-123456789763`)
`{{accountCouponId}}`	Unique identifier generated by combining a coupon-ID and a unique user-ID, to which the coupon is accessable to.
`activated`	This boolean requests the coupon to be activated (`true`) or deactivated (`false`) for given account

This will return no body, but will activate or deactivate the coupon for given user. Activatable coupons have to be activated for them to be redeemable.
Note that this is not relevant if the coupon is permanently valid. Activatable coupons are automatically redeemed when redemption conditions are met.

5.3 Coupon Redemption

To redeem a coupon, the unique `accountCouponId` has to be sent using

```
curl --location --request PATCH
'{{api_url}}/v2/accountcoupons/{{accountCouponId}}' \
--header 'interaction-id: {{interactionId}}' \
--header 'Authorization: {{jwt_token}}' \
--header 'Content-Type: application/json' \
--header 'id-type: {{idType}}' \
```

with

Variable	Description
`{{api_url}}`	Endpoint of the api. • https://staging.convercus.io (Staging) • https://api.convercus.io (Production)
`{{accountId}}`	ID of the account that will receive the transaction. The ID has to be given in the format, that is dictated by the`{{idType}}`,
`{{jwt_token}}`	The JWT-token, which has been generated by authentication.
`{{interactionId}}`	Unique Identifier of the cash machine (or virtual equivalent, e.g. online-shop), which produced the receipt. This ID has to be listed in the Convercus System as with this ID, the connection bon-to-store is made.
`{{idType}}`	Identifier-Type (corresponding to the `{{accountId}}`). **Available values:** • `APPCODE` (e.g. `A1B2C3D4E5`) • `CARDCODE` (e.g. `V1W2X3Y4Z5`) • `EXTERNALCODE` (e.g. `123456780123`) • `ID` (account-identifier, e.g. `7d123457-bfa1-4a83-8213-123456789763`)
`{{accountCouponId}}`	Unique identifier generated by combining a coupon-ID and a unique user-ID, to which the coupon is accessable to.

In the current version of the interface, the response does not contain any body information, thus any information about the coupon (especially the `externalReference`) has to be stored before the redemption (they cannot be reproduced).

After the coupon has been redeemed sucessfully, the respective action / pricing rule / ect. (for which the original `externalReference` may be needed) may be applied.

5.3.1 NOTE ABOUT EXTRA-INCENTIVATION COUPONS

The coupon categories `EXTRAPOINT` and `MULTIPLIER` will be triggered automatically by the transaction itself (if all conditions are fulfilled) and therefore cannot be redeemed manually. Keep in mind however, that if the coupons are activatable, they need to be activated by the customer first.
A manual redemption of coupons is only necessary / possible for manual coupon categories `DISCOUNT` and `REWARD`.