convercus

Transaction Api Implementation Manual v1.3.0

convercus

POINT 4MORE

Table of Content

1 Transaction Api Implementation Manual

This manual describes how to implement an interface for sending digital receipts to Convercus. With this documentation, it should be more intuitive and straight-forward to map all relevant information of a digital receipt to Convercus standard transaction format. 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
v1.3.0	2020-11-24	• Adding Domain Search to description • Minor corrections
v1.2.0	2020-08-21	• Adding User / Account Identification to documentation • Minor corrections
v1.1.0	2020-05-08	• Adding BURN (PayWithPoints) to documentation • Minor corrections
v1.0.0	2020-04-30	• 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 and analogously in the chapters about earn- and burn-transactions.

4.1 Card identification

All connections of transactions, bookings, 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 General Questions

5.1 Which Bons should be sent?

If not communicated otherwise, all completed bons (with and without loyalty connection) should be sent to Convercus Interface. It is crucial to secure, that every purchase is only incentivated once.
Thus, processes like partial payments and commisions (which may produce a bon itself) should not be sent to Convercus. The bon (with all line-items) should be sent and incentivated only after (and only if) it has been paid completely.
It is not necessary to exclude certain line-items, as exclusion from incentivation on the level of purchases can be configured in Convercus system (given a constistent set of identifiers for the line-items). Always give all the line-items, which are printed on the bon itself.

6 Basic Call Setup: Transaction vs Account-Transaction

Digital receipts sent to Convercus may (or may not) have a Loyalty-Connection. Thus, Convercus Api differentiates between Transactions (no Loyalty-connection; Bon is only stored in database for later purposes) and Account-Transactions (Loyalty-connection). While for the same purchase, the body of both transaction-types will stay the same (as will be the subject of the next chapter), the headers and endpoints deviate from each other, as we will see in the next two subchapters.

6.1 Transaction

Transactions without Loyalty-connect can be sent with the following request

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

with the following 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 Bon-Information as explained in the next chapter.

6.2 Account-Transaction

If a transaction has to be connected to a loyalty-account, the following request should be sent:

```
curl --location --request POST '{{api_url}}/accounts/{{accountId}}/transactions' \
--header 'Authorization: {{jwt_token}}' \
--header 'interaction-id: {{interactionId}}' \
--header 'id-type: {{idType}}' \
--header 'Content-Type: application/json' \o
--data-raw '{{body}}'
```

Note the different endpoint (accounts instead of transactions) and the position of the Loyalty-information {{identifierCode}} and {{idType}}.

The used variables are the following:

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)
{{body}}	Body with Bon-Information as explained in the next chapter.

7 Transaction Request Body (Earn)

In this chapter, we will explain how to fill the {{body}} of transaction requests.

7.1 Minimal Transaction Body (Mandatory Fields)

The minimal set of receipt information (containing only mandatory fields) is the following:

```
{
        "transactionType": "EARNTRANSACTION",
        "amount": 99.90
}
```

with mandatory attributes

Attribute	Description
transactionType	Type of the transaction. For digital receipt, always use the value EARNTRANSACTION
amount	Total amount of the bon.

7.2 Maximal Transaction Body (Optional Fields)

The minimal setup is only providing the minimal functionality (i.e. store purchase-totals and connect them with loyalty-accounts). In order to benefit from a range of modules in the Convercus system, it may be necessary to add optional attributes.
The following example will give a maximal set of receipt information. The relevant fields for your implementation depend on the system and program setup and need to be coordinated with the program manager.

```
{
        "transactionType": "EARNTRANSACTION",
        "transactionTime": "2020-04-30T10:50:00+02:00",
        "valueTime": "2020-04-30T10:50:00+02:00",
        "externalId": "UniqueBonID",
```

```
    "amount": 99.90,
    "currencyCode": "EUR",
    "lineItems": [...],
    "tenderItems": [...],
    "linkedTransaction": {...}
}
```

The following attributes may be used (if not stated otherwise, every attribute may only be used once):

Attribute	Description	Relevance
transactionType	Type of the transaction. For digital receipt, also use the value EARNTRANSACTION	**Mandatory**
transactionTime	ISO 8601 datetime, when the transaction was created (i.e. datetime of the receipt). Either give the datetime in UTC (eg. 2020-04-08T10:50:00Z or in local-time with timezone (e.g. 2020-04-08T10:50:00+02:00 for CET with daylight-saving-time or 2020-01-08T10:50:00+01:00CET).	**Optional**
valueTime	ISO 8601 value datetime of the transaction (i.e. the datetime, when the booked transaction-points become active). **Note:** If this field is missing, the valueTime will be set to the time, when the transaction is processed in Convercus system. Do **not** fill this field unless you don't want to have this standard-situation.	**Optional**
externalId	Unique (over the program) Bon-ID. As a security measure (don't incentivate the same bon twice), reoccuring bons with the same externalID are rejected by the system, so make sure that every bon-id is unique.	**Optional**
amount	Total amount of the bon. **Important:** This amount should always be the same as the sum of all purchases and returns of the bon. Convercus-system rejects bons, where the amount is less the sum of lineItems. Any deviation should be checked before sending the bon as there may be an error in the bon-structure (e.g. missing positions).	**Mandatory**
currencyCode	Currency Code (e.g. EUR) of the total amount.	**Optional**
lineItems	List of purchased line items of the bon. This attribute will be explained in more Detail in the respective subchapter.	**Optional**
tenderItems	List of payment items of the bon. This attribute will be explained in more Detail in the respective subchapter.	**Optional**

linkedTransaction	Link to an existing transaction. This attribute will be explained in more Detail in the respective subchapter.	**Optional**

7.2.1 LINEITEMS ATTRIBUTE

7.2.1.1 Minimal lineItem

Every line-item (both purchases and returns) of the bon should be mapped to an item in the lineItems-list.

The minimal required set of information for a line-item is the following:

```
"lineItems": [
{
        "sequenceNumber": 1,
        "type": "SALE",
        "itemID": "2758221",
        "extendedAmount": 199.80
        },
...
]
```

with mandatory attributes:

Attribute	Description
sequenceNumber	Sequence number of the line-item. This value is incremented with every line-item, starting at 1.
type	Type of the transaction. The values SALE and RETURN are used to differentiate the processes of purchase and cancellation / return.
itemID	This is the unique identifier of the product. Usually this is the article code in an external system (e.g. barcode). **Note about Incentivation:** On the level of this ID, it is possible to define incentivation rules (i.e. exclusion or multiple points). Thus, the set of IDs for this should be consistent over the whole program.
extendedAmount	This is the actual amount, this item was sold for. Accordingly, this value has to incorporate all given discounts. **Note:** Depending on the type of line-item, one has to keep track of the signs of this amount: SALE-transactions have positive signs, RETURN-transactions have negative signs.

Like before, the minimal setup is only providing the minimal functionality, which may be extended by adding optional attributes. For example, in order to be able to make special incentivation for certain line-items (e.g. exclusion from incentivation, extra-points), it's important to provide a detailed and consistent set of information about the product itself. In the following we will show maximal sets of information, also stressing the difference between SALE- and RETURN-transactions a little more.

7.2.1.2 Maximal SALE-lineItem

The SALE is most common and most important line-item-type as this is classic purchase line-item. Every purchase, which is printed on the bon should also be added to the lineItem-list. Note, that Discounts should not produce a line-item for themselves. It is possible to track original pricing in the lineItem-element itself.
The following example will give a maximal set of SALE-line-item-information. The relevant fields for your implementation depend on the system and program setup and need to be coordinated with the program manager.

```
"lineItems": [
{
"sequenceNumber": 1,
"type": "SALE",
"itemID": "2758221",
"merchandiseGroupCode": "1201",
"merchandiseGroupName": "Shoes",
"merchandiseSubGroupCode": "120103",
"merchandiseSubGroupName": "Sneakers",
"description": "Test-Shoe",
"actualSalesUnitPrice": 99.90,
"quantity": 2,
"extendedAmount": 199.80,
"currencyCode": "EUR",
"taxRate": 19.00
}
]
```

T
he following attributes may be used (if not stated otherwise, every attribute may only be used once):

Attribute	Description	Relevance
sequenceNumber	Sequence number of the line-item. This value is incremented with every line-item, starting at 1.	**Mandatory**
type	Type of the transaction. For purchases, the value is SALE.	**Mandatory**
itemID	This is the unique identifier of the product. Usually this is the article code in an external system (e.g. barcode). **Note about Incentivation:** On the level of this ID, it is possible to define incentivation rules (i.e. exclusion or multiple points). Thus, the set of IDs for this should be consistent over the whole program.	**Mandatory**

merchandiseGroupCode	This is the unique identifier of the product's merchandise group. **Note about Incentivation:** On the level of the ID, it is possible to define incentivation rules (i.e. exclusion or multiple points). Thus, the set of IDs for this should be consistent over the whole program.	**Optional**
merchandiseGroupName	This is the name of the product’s merchandise group.	**Optional**
merchandiseSubGroupCode	This is the unique identifier of the product's merchandise subgroup. **Note about Incentivation:** On the level of the ID, it is possible to define incentivation rules (i.e. exclusion or multiple points). Thus, the set of IDs for this should be consistent over the whole program.	**Optional**
merchandiseSubGroupName	This is the name of the product’s merchandise subgroup.	**Optional**
description	This is the name or description of the product.	**Optional**
actualSalesUnitPrice	This is the amount, **a single item** was sold for. So this value is equal to the regular unit price - all discounts on the level of this product. **Note:** For SALE-transaction, this value is always positive.	**Optional**
quantity	This is the number of units, that were sold in this line-item.	**Optional**
extendedAmount	This is the actual amount, this item was sold for. Accordingly, this value has to incorporate all given discounts. **Note:** For SALE-transaction, this value is always positive.	**Mandatory**
currencyCode	Currency Code (e.g. EUR) of the extendedAmount.	**Optional**
taxRate	Percentage of tax applied for this line-item.	**Optional**

7.2.1.3 Maximal Return-lineItem

As it is also pretty common to have negative bookings in a bon, as products may be returned or cancelled altogether. Such line-items may be added as a RETURN-line-item. However, there are a few things, to be minded.

Note about the difference between Cancellation and Returns:

The Convercus system does not differentiate between cancellation and return as both do

basically the same: They refer to a previous, positive purchase / booking and negate it by adding a negativ purchase / booking.
Thus, all negative bookings (i.e. returns and cancellations) shall be mapped to a RETURN-line-item.

Note about the setup of Return line-items

RETURN-line-items have to have the exact same setup as the corresponding SALE-Line-item had. This secures the same treatment of the return while processing in order to rebook points. Therefore, especially all discounts given to the initial booking have to be added to the RETURN-item as well (so that the price is correct). Furthermore, the same product information has to be given (so that incentivation exclusion and extra-points are handled correctly).
Note however, that RETURN-line-amounts have to have negative signs.
The following example will give a maximal set of RETURN-line-item-information. The relevant fields for your implementation depend on the system and program setup and need to be coordinated with the program manager.

```
"lineItems": [
{
        "sequenceNumber": 1,
        "type": "RETURN",
        "itemID": "2758221",
        "merchandiseGroupCode": "1201",
        "merchandiseGroupName": "Shoes",
        "merchandiseSubGroupCode": "120103",
        "merchandiseSubGroupName": "Sneakers",
        "description": "Test-Shoe",
        "currencyCode": "EUR",
        "actualSalesUnitPrice": -99.90,
        "quantity": 1,
        "extendedAmount": -99.90,
        "taxRate": 19.00
        }
]
```

The following attributes may be used (if not stated otherwise, every attribute may only be used once):

Attribute	**Description**	**Relevance**
sequenceNumber	Sequence number of the line-item. This value is incremented with every line-item, starting at 1.	**Mandatory**
type	Type of the transaction. For cancellations and returns, the value is RETURN.	**Mandatory**
itemID	This is the unique identifier of the product. Usually this is the article code in an external system (e.g. barcode). **Note about Incentivation:** On the level of this ID, it is possible to define incentivation rules (i.e. exclusion or multiple points). Thus, the set of IDs for this should be consistent over the whole program.	**Mandatory**

merchandiseGroupCode	This is the unique identifier of the product's merchandise group. **Note about Incentivation:** On the level of the ID, it is possible to define incentivation rules (i.e. exclusion or multiple points). Thus, the set of IDs for this should be consistent over the whole program.	**Optional**
merchandiseGroupName	This is the name of the product's merchandise group.	**Optional**
merchandiseSubGroupCode	This is the unique identifier of the product's merchandise subgroup. **Note about Incentivation:** On the level of the ID, it is possible to define incentivation rules (i.e. exclusion or multiple points). Thus, the set of IDs for this should be consistent over the whole program.	**Optional**
merchandiseSubGroupName	This is the name of the product's merchandise subgroup.	**Optional**
description	This is the name or description of the product.	**Optional**
actualSalesUnitPrice	This is the amount, **a single item** was returned for. So this value is equal to the rebooked regular unit price - all discounts on the level of this product. **Note:** For RETURN-transaction, this value is always negative.	**Optional**
quantity	This is the number of units, that were returned in this line-item.	**Optional**
extendedAmount	This is the actual amount, this item was returned for. Accordingly, this value has to incorporate all given discounts. **Note:** For RETURN-transaction, this value is always negative.	**Mandatory**
currencyCode	Currency Code (e.g. EUR) of the extendedAmount.	**Optional**
taxRate	Percentage of tax applied for this line-item.	**Optional**

7.2.2 TENDERITEMS ATTRIBUTE

In addition to the purchased articles,it may also be reasonable to document payment-methods - especially, for the tracking of gift-cards and pay-with-points transactions. These payment methods may be added to a list of tenderItems.

7.2.2.1 Minimal tenderItem

The minimal required set of information for a tender-item is the following:

```
"tenderItems": [
```

```
{
        "sequenceNumber": 3,
        "amount": 10.00
        }
]
```

with mandatory attributes:

Attribute	Description
sequenceNumber	Sequence number of the line-item. This value is incremented with every item. It may start with 1 or continue the line-item sequence numbering.
amount	This is the amount, that was paid in the format of the respective payment method.

7.2.2.2 Maximal tenderItem

Again, the minimal setup is only providing the minimal functionality, which may be extended by adding optional attributes. For example, it is common to allow multiple payment-methods at the same time, which may be documented.
In the following, we will show maximal set of information, where also multiple payment methods were used. The relevant fields for your implementation depend on the system and program setup and need to be coordinated with the program manager.

```
"tenderItems": [
{
        "sequenceNumber": 3,
        "tenderType": "Loyalty",
        "tenderId": "PayWithPoints",
        "amount": 10.00,
        "currencyCode": "EUR",
        "taxRate": 19.00
        },
{
        "sequenceNumber": 4,
        "tenderType": "GiftCard",
        "tenderId": "Geschenkkarte50",
        "amount": 50.00,
        "currencyCode": "EUR",
        "taxRate": 19.00
        },
{
        "sequenceNumber": 5,
        "tenderType": "Cash",
        "tenderId": "Barzahlung",
        "amount": 39.90,
        "currencyCode": "EUR",
        "taxRate": 19.00
        }
]
```

The following attributes may be used (if not stated otherwise, every attribute may only be used once):

Attribute	Description	Relevance
sequenceNumber	Sequence number of the line-item. This value is incremented with every item. It may start with 1 or continue the line-item sequence numbering.	**Mandatory**
tenderType	This attribute defines the payment method description in the third party system. It should be secured, that these descriptions are consistent.	**Optional**
tenderId	This attribute defines the payment method ID in the third party system. It should be secured, that these IDs are consistent.	**Optional**
amount	This is the amount, that was paid in the format of the respective payment method.	**Mandatory**
currencyCode	Currency Code (e.g. EUR) of the amount. **Note:** Always use real currencies (or the equivalent of a pseudo-currency in a real currency) here. Example: If a amount of 5,00 EUR was paid with Loyalty points, give the amount of 5,00 EUR (and not the amount of burned points).	**Optional**
taxRate	Percentage of tax applied for this payment method.	**Optional**

Note:
Always keep in mind, that the sum of all payment-amounts should get the same value as the bon-total-amount.

7.2.3 LINKEDTRANSACTION

If a transaction has to be linked to an older transaction (e.g. a return, which is mapped to the original sale), the following attribute can be added:

```
"linkedTransaction": {
        "linkType": "EXTERNALID",
        "linkValue": "bon_id_123456"
        }
```

with mandatory attributes:

Attribute	Description

linkType	Identifier-Type (corresponding to the linkValue).
	Available values:
	• TRANSACTIONID
	  ○ Convercus-ID of original transaction
	  ○ e.g. db1234b5-67fd-8912-3c4e-56789ac57595
	• EXTERNALID
	  ○ Original-ID of the transaction
	  ○ e.g. bon_id_123456

linkValue	Value of the above linkType.

7.2.4 MAXIMAL EXAMPLE JSON

Summarzing everything, this is an example maximal JSON, with incorporates almost all previous settings (valueTime was omitted as here, the booking is to be expected after processing).

```
{
        "transactionType": "EARNTRANSACTION",
        "transactionTime": "2020-04-08T10:50:00+02:00",
        "externalId": "UniqueBonID",
        "amount": 99.90,
        "currencyCode": "EUR",
        "lineItems": [
        {
                "sequenceNumber": 1,
                "type": "SALE",
                "itemID": "2758221",
                "merchandiseGroupCode": "1201",
                "merchandiseGroupName": "Shoes",
                "merchandiseSubGroupCode": "120103",
                "merchandiseSubGroupName": "Sneakers",
                "description": "Test-Shoe",
                "actualSalesUnitPrice": 99.90,
                "quantity": 2,
                "extendedAmount": 199.80,
                "currencyCode": "EUR",
"taxRate": 19.00
                },
        {
                "sequenceNumber": 2,
                "type": "RETURN",
                "itemID": "2758221",
                "merchandiseGroupCode": "1201",
                "merchandiseGroupName": "Shoes",
                "merchandiseSubGroupCode": "120103",
                "merchandiseSubGroupName": "Sneakers",
                "description": "Test-Shoe",
                "actualSalesUnitPrice": -99.90,
                "quantity": 1,
```

```
                "extendedAmount": -99.90,
                "currencyCode": "EUR",
                "taxRate": 19.00
                }
        ],
        "tenderItems": [
        {
                "sequenceNumber": 3,
                "tenderType": "Loyalty",
                "tenderId": "PayWithPoints",
                "amount": 10.00,
                "currencyCode": "EUR",
                "taxRate": 19.00
                },
        {
                "sequenceNumber": 4,
                "tenderType": "GiftCard",
                "tenderId": "Geschenkkarte50",
                "amount": 50.00,
                "currencyCode": "EUR",
                "taxRate": 19.00
                },
        {
                "sequenceNumber": 5,
                "tenderType": "Cash",
                "tenderId": "Barzahlung",
                "amount": 39.90,
                "currencyCode": "EUR",
                "taxRate": 19.00
                }
        ],
        "linkedTransaction": {
                "linkType": "EXTERNALID",
                "linkValue": "bon_id_123456"
                }
}
```


8 Burn Interface (PayWithPoints)

The possibility to make (partial) payments with earned points is a common feature in Loyalty programs. In this chapter, we will explain how to implement this payment method using Convercus Api.

8.1 Getting current Account Balance

Before payments with points can be made, it is necessary to get the current account balance. There are two ways to get the balance.

The following request will respond the current amount of points on a certain account:

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

whereas the following request (note the difference in the url) will respond the value of those points in Euro as amount.

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

The ladder option offers a shortcut for payment implementation as the (possibly not static) calculation of the account balances financial value is already performed by the Convercus System.

The used variables in the requests are the following

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)

8.2 PayWithPoints-Transaction

Like all api-bookings, paying with points is also handled via account-transaction.

Analogously, the following request has to be sent to burn points by paying with them:

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

where again

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)
{{body}}	Body with PayWithPoints information as explained in the next subchapter.

8.2.1 PAYWITHPOINTSTRANSACTION-BODY

The minimal set of {{body}} information about payments with points is the following

```
{
        "transactionType": "PAYWITHPOINTSTRANSACTION",
        "amount": 5.00
}
```

with the following mandatory attributes

Attribute	Description
transactionType	Type of the transaction. For paying with points, always use the value PAYWITHPOINTSTRANSACTION.
amount	Total amount, that should be paid with the corresponding amount of points. **Important Note:** The value of this attribute is always given in terms of a currency (as this is a payment method). It is **not** an amount of points. Be sure to implement a way to convert points to financial value or get the account balance in your currency via GET accounts/balance/EUR.

Once again, the minimal setup is only providing the minimal functionality (i.e. burn the amount of points that's worth the amout you want to pay with).
The following example will give a maximal set of PayWithPoints information. The relevant fields for your implementation depend on the system and program setup and need to be coordinated with the program manager.

```
{
        "transactionType": "PAYWITHPOINTSTRANSACTION",
        "reason": "Paying with Points",
        "transactionTime": "2020-04-30T10:50:00+02:00",
        "valueTime": "2020-04-30T10:50:00+02:00",
        "externalId": "UniqueBonID",
        "amount": 5.00,
        "currencyCode": "EUR",
        "linkedTransaction": {
                "linkType": "EXTERNALID",
                "linkValue": "bon_id_123456"
                }
}
```

The following attributes may be used (if not stated otherwise, every attribute may only be used once):

Attribute	Description	Relevance
transactionType	Type of the transaction. For paying with points, always use the value PAYWITHPOINTSTRANSACTION.	**Mandatory**
reason	Additional booking text for the transaction. This value can be set to anything and may be used to transmit an internal description / wording of the process (e.g. PayWithPoints).	**Optional**
transactionTime	ISO 8601 datetime, when the payment was made (usually this is the same as the datetime of the corresponding receipt). Either give the datetime in UTC (eg. 2020-04-08T10:50:00Z or in local-time with timezone (e.g. 2020-04-08T10:50:00+02:00 for CET with daylight-saving-time or 2020-01-08T10:50:00+01:00CET).	**Optional**
valueTime	ISO 8601 value datetime of the transaction (i.e. the datetime, when the booked transaction-points become active). **Note:** If this field is missing, the valueTime will be set to the time, when the transaction is processed in Convercus system. Do **not** fill this field unless you don't want to have this standard-situation.	**Optional**

externalId	Unique (over the program) Bon-ID. As a security measure (don't incentivate the same bon twice), reoccuring bons with the same externalID are rejected by the system, so make sure that every bon-id is unique. **Note:** As an exception to the rule, PAYWITHPOINTSTRANSACTION can have the same externalId as the EARN-transaction they are connected to. This allows to use the same bon-ID for the same receipt, even if EARN und BURN are two separate requests.	**Optional**
amount	Total amount, that should be paid with the corresponding amount of points. **Important Note:** The value of this attribute is always given in terms of a currency (as this is a payment method). It is **not** an amount of points. Be sure to implement a way to convert points to financial value or get the account balance in your currency via GET accounts/balance/EUR.	**Mandatory**
currencyCode	Currency Code (e.g. EUR) of the total amount.	**Optional**
linkedTransaction	Link to an existing transaction, e.g. the original earn transaction where the payment was made.	**Optional**

linkedTransaction / linkType	Identifier-Type (corresponding to the linkValue).
	Available values:
	• TRANSACTIONID
	  ○ Convercus-ID of original transaction
	  ○ e.g. db1234b5-67fd-8912-3c4e-56789ac57595
	• EXTERNALID
	  ○ Original-ID of the transaction
	  ○ e.g. bon_id_123456
		Mandatory (if linkedTransaction is **used)**

linkedTransaction / linkValue	Value of the above linkType.	**Mandatory (if** linkedTransaction is **used)**

Note, that the basic setup is similar to the EARN-case above. However, the usage of lineItems and tenderItems is not meaningful here.

8.3 Connection between EARN and BURN transactions

It is very important to note, that receipts where a (partial) payment is made with points requires two transaction requests - one for the payments with points (BURN) and one for the bon as a whole (EARN). If payments are tracked in the EARN-bon, make sure to add the correct amount of the PAYWITHPOINTSTRANSACTION to the bon, e.g. add

```
"tenderItems": [
{
        "sequenceNumber": 3,
        "tenderType": "Loyalty",
        "tenderId": "PayWithPoints",
        "amount": 10.00,
        "currencyCode": "EUR",
        "taxRate": 19.00
        },
        ...
]
```