Transaction API endpoints

Overview

This API provides access to POS transaction data. It is possible to retrieve past transactions and to receive push updates of any incoming transaction in real-time.

Transaction data is stored in GTC format. Please refer to GTC specification for details (you can find this document in the 'Getting Started' section under 'API documentation').

Important: if your/your client's business does not have GTC enabled, no data will be available for it. To enable GTC feature, please contact our support.

There is no way to write any data to the transaction API, as by definition only the POS can create transaction data. So this API provides read-only access. Other API's can be used to trigger actions on the POS which will result in new transactional data to be provided through this API (e.g. placing an order).

Version information

Version : v3.0

URI scheme

Hostname is the same as your POS Cloud.

BasePath : /api/transaction/v3.0

Headers:

`X-Token` : You can obtain this token in the Cloud
`X-Business-Units` : The business unit id is the cloud ID (sometimes also referred to as locationID). Transaction data is tied to each business unit id, so the correct id must be used. Make sure your token allows to access the given business unit, you can check this in the API configuration menu of the Cloud.

GET endpoints

GET endpoints provide access to past transactional data (pull mode). These are normal HTTP GET requests which deliver a result and close the connection afterwards.

Get business periods by date

Provides access to all business periods the service has data for. Business periods are uniquely identified by the `periodId` value. Business day is additional information and represents the day a particular period was started. There can be multiple periods in a single business day.

Method: `GET`

URL: `/business_periods`

Headers: `X-Token`, `X-Business-Units`

Request: `/business_periods`

Response:

HTTP Status: 200

```
{
  "businessPeriods": [
    {
      "periodId": 100,
      "businessDay": "2020-01-13",
      "startPeriodTimestamp": "2018-03-13T14:36:11Z",
      "finishPeriodTimestamp": "2018-03-14T02:10:51Z"
    },
    {
      "periodId": 101,
      "businessDay": "2020-01-15",
      "startPeriodTimestamp": "2018-03-15T17:20:43Z",
      "finishPeriodTimestamp": null
    }
  ]
}
```

Get transactions by periodId

Provide a GTC document containing all transactions for a given period. If a period is not yet closed, it provides all transactions that have been recorded by the POS so far.

Method: `GET`

URL: `/transactions/{periodId}`

If no `periodId` is provided, the currently open or, if there is no period open at this time, that last period is returned.

Request: `/transactions/100`

Headers: `X-Token`, `X-Business-Units`

Response:

HTTP Status: 200


```
{
  "type": "day",
  "version": "1.0",
  "tenantId": "LSTSDE",
  "locationId": 49810,
  "companyId": 1,
  "businessDay": "2020-01-23",
  "periodId": 100,
  "createdTimestamp": "2020-01-23T13:15:14Z",
  "transactions": [
    {
      "head": {
        "typeCode": "42",
        "startedTimestamp": "2020-01-23T13:15:14Z",
        "uuid": "f7761233-0305-43a7-86e2-d28eb0a8b055"
      },
      "lineItems": [
        {
          "sequenceNumber": 1,
          "typeCode": "17",
          "primary": {
            "transactionSequenceNumber": 30
          },
          "flags": {
            "isSuspendedFlag": false
          },
          "extras": {
            "appVersion": "2.0.4444"
          }
        }
      ]
    },
    {
      "head": {
        "typeCode": "43",
        "startedTimestamp": "2020-01-23T13:15:14Z",
        "uuid": "732d628a-ddc8-45c5-9eef-269a4ba11295"
      },
```

```
      "lineItems": [
        {
          "sequenceNumber": 1,
          "typeCode": "17",
          "primary": {
            "transactionSequenceNumber": 40
          },
          "flags": {
            "isSuspendedFlag": false
          },
          "extras": {
            "appVersion": "2.0.4444"
          }
        }
      ]
    }
  ],
  "masterData": null,
  "sequenceNumber": 1
}
```


Websocket endpoints (push API)

Websocket (WS) endpoints allow to establish an open connection through which live transaction updates are pushed to the client. Normally the connection stay open until the client closes it. Implementations must assure though, that the connection can also be closed by the server in rare cases (e.g. service restarts or network interruptions).

Websockets can be established with two modes: `new` and `all`.

For `new` connections, the server will start pushing any transaction updates that occur after the connection was established. This is useful if past data is not relevant and only new events are required.

For `all` connections, the server will first push out all transactions from the current period and then any subsequent updates. In this mode it is assured, that all transactions will be provided at any given time. This can be used by implementations that require the full data of the day and require subsequent updates, e.g. a real-time dashboard application showing values to date.

Important! Timeout for a websocket connection is 60 seconds. It is recommended that the websocket client sends a ping every 20 seconds to keep the connection alive.

Get transactions via transaction feed

Method: `WS`

URL: `/ws?type=[new|all]` Default type is `new`

Headers: `X-Token`, `X-Business-Units`

Request: `/ws`

Response:


```
{
  "head": {
    "typeCode": "00",
    "uuid": "942b8cf7-46db-4662-b5e4-033c2952a253",
    "startedTimestamp": "2018-03-09T11:25:16Z",
    "trainingFlag": false,
    "voidedTrUuid": null
  },
  "lineItems": [
    {
      "typeCode": "00",
      "sequenceNumber": 1,
      "primary": {
        "orderSequenceNumber": 1
      },
      "related": {
        ".....": "skipped to keep it brief"
      },
      "flags": {
        ".....": "skipped to keep it brief"
      },
      "amounts": {
        "quantity": 3000,
        "units": 1000,
        "regularUnitPrice": 3000,
        "regularAmount": 9000
      },
      "timestamps": {
        "recordedTimestamp": "2018-03-09T11:25:16Z"
      },
      "extras": {
        "voidedLineItemSequenceNumber": null,
        "triggeringLineItemSequenceNumber": null,
        "party": 1,
        "itemName": "Cola",
        "itemKind": 0,
        "divisionName": "Beverages",
        "groupName": "Soft Drink",
        "itemEntryMethod": "02",
```

```
        "priceEntryMethod": "00"
      }
    },
    {
      "typeCode": "17",
      "sequenceNumber": 2,
      "primary": {
        "transactionSequenceNumber": 10006,
        "dayDeviceTransactionSequenceNumber": 30,
        "invoiceNumber": null
      },
      "related": {
        ".....": "skipped to keep it brief"
      },
      "flags": {
        "isCancelFlag": false,
        "isBusinessCaseCompleteFlag": true,
        "isSuspendedFlag": true,
        "isOfflineFlag": false,
        "isRestoreFlag": false
      },
      "amounts": {
        "regularSubTotal": 9000,
        "actualSubTotal": 9000,
        "discountSubTotal": 0,
        "tipSubTotal": 0,
        "overAllSubTotal": 9000
      },
      "timestamps": {
        "updatedTimestamp": "2018-03-09T11:25:16Z"
      },
      "extras": {
        ".....": "skipped to keep it brief"
      }
    }
  ]
}
```


Error handling

200 - OK (record stored, changed or deleted, return value returned)

400 - Bad Request for any request that is not valid (missing mandatory parameters, incorrect format of values, ...)

403 - Unauthorized (check security token or other auth issues).

404 - Not Found (requested instance, business, periodId, tenant or URL is absent on the server side).

500 - Server error on any unexpected error (exception).

All errors have similar body-structure in response, for example:

```
{
  "result":
    {
      "status_code": 400,
      "details": "Missing required field: date"
    }
}
```