General Transaction Container

Version 1.0

March 3, 2020

Contents

List of Figures

List of Tables

1 Introduction

1.1 GTC - General Transaction Container

The document describes and defines the exchange format that is used by the POS-System to provide transaction data to 3rd parties. The exchange format is called GTC which stands for General Transaction Container.

One of the most important aspects of a modern POS-System is interoperability. A POS-System must provide an open and standardized interface so that any 3rd party system can connect to the POS-System and process transaction data on behalf of the business owner who operates the POS-System. The most obvious reason for that is of course that a POS vendor can not provide solutions for all possible business requirements. It is almost impossible to provide the expertise for very different business requirements like POS, ERP, Accounting, etc at the same time. It is also almost impossible to facilitate the necessary resources to implement these business requirements.

Therefore, the POS-System follows the "Best of Bread" approach, meaning it concentrates on its core business of providing a POS-System and then let 3rd parties connect which provide the best solutions in their respective field. This approach makes it also possible for a business owner who run a POS-System to pick a specific 3rd party system that fits his business needs and let them integrate with the POS-System using GTC transaction data.

1.2 GTC - Fundamentals

GTC is designed around some fundamental rules how POS-Systems create transactions:

1. Transactions that were finalized are never modified afterwards. If an update to a transaction is necessary, then a reverse posting / transaction is created (e.g. Voids).
2. Even while an incomplete transaction is registered, already created line item are never deleted. Here also reverse postings are required to modify the transaction.
3. A transaction always reflects exactly what happened at the POS while the transaction was registered. So, there will be no aggregation of sold items or payment line items in a transaction.
4. Everything that happens at a POS is captured using transactions.

GTC documents come in two flavors but they contain the data using the same GTC structure as defined in this specification. They just differ in how much data is in the document that is provided to the 3rd party (aka Consumer of a GTC Document).

Log-Document

A Log-Document represents an update to one specific transaction. It just contains the data for that transaction only. It always contains the complete content of the transaction. An update is always terminated using a Status Update Line Item. The latest update to a transaction can be easily identified, it consists of all the line items between the last two ***Status Update Line Items***. The POS never sends two updates in one Log-Document. If events happen in short sequence, the POS will send two Log-Documents for the affected transaction. The last ***Status Update Line Item*** of the document also indicates if the transaction is finalized with that Log-Document (meaning, there won't be any further updates regarding the transaction).

A typical consumer of a Log-Document might be a Reservation-System or a Loyalty-System or a Property-Management-System. So, systems that are focused on one transaction only.

Day-Document

A Day-Document contains all transactions of a specific business day. The first transaction in the documents will be a Period Open Transaction and the last transaction will be a Period Close Transaction. All transactions in a Day-Document are finalized. The ***Period Close Transaction*** contains the Z-Report data (cash statement) of the business day.

A typical consumer of a Day-Document might be an ERP-System or an Accounting-System. So, systems that are focused on all transactions of a specific business day.

How to configure the POS-System to send GTC Documents to 3rd parties / consumers is not part of the document, please refer to the cloud manual here.

1.3 JSON

The upcoming chapter ***GTC Container Structure*** will define the structure of a GTC document in detail, but it might be a good idea to point out some fundamental properties of a GTC document first.

A GTC document is represented as a JSON object. A JSON schema for validating the structure might be provided in the future.

The description of the GTC structure is presented in form of a dictionary that defines which fields / keys are available for a specific JSON structure together with their exact meaning.

There are also some additional rules that always apply to the JSON representation. Since a schema is not available at the moment, the rules must be stated verbally. The POS-System which generates the GTC documents will follow those rules. A consumer might use these rules together with a future schema to validate documents.

The rules are:

- All fields of a (sub-)structure / entity should always be present, even if they are empty.
- An empty field always has the value **null**.
- An empty subsection is always represented as **{}**.
- A field is empty, if the functionality which is connected with a field was not used or does not apply to a specific transaction / business case.
- A subsection is empty, if the data is not relevant for a specific business case.
- All timestamps are based on UTC. They are represented using ISO 8601 and timezone Z with a precision of 1 second (e.g. "2019-07-22T15:32:52Z").
- All float values are represented as integers based on the divisor 1000. Please have a look at the following sections for a more detailed discussion of why this is the case.
- All numbers are always represented as positive integers. A GTC document does not contain negative numbers. A reverse posting is not indicated using a negative number, but by a type code which implies that it is a reverse posting. There is one exception to this rule, negative numbers might show up in total amounts of the ***payments*** or ***salesSummaries*** section of the ***Period Close Line Item*** if more was spent then was received.

1.4 Floats as Integers

A GTC container represents floats as integers based on a divisor 1000, e.g. 12.37 is represented as 12370.

Why is it done this way?

- One major reason is that floats and doubles (which are used by most programming languages) are imprecise. They are imprecise because they use the IEEE 754 Standard internally which can not represent many floats exactly. Usually, you might use a fix point library, but since GTC can not assume any particular programming language that is used to implement a consumer which then might or might not support a specific fixed point library, GTC must relay on the bare minimum. As a side note, floats and doubles might be even hiding unnoticed in a JSON parser which is used to convert a GTC document to an internal map or object, here rounding issue might be introduced unnoticed.
- Secondly, floats are not required to represent monetary values. Monetary values are exact if you represent them using the smallest denomination, e.g. Cents for Euro or Dollar. Therefore, monetary value can be easily and naturally represented using integers.
- But why is the divisor 1000 used and not 100 as you would expect for Euro and Dollar. Funny enough, there are currencies using 3 decimal places, e.g. "Tunisian Dinar", "Libyan Dinar" and "Jordanian Dinar" (actually, most of the Dinar family). Also, it is easier to use one fixed divisor and not connect the divisor to the currency. If you have to check for problems in GTC documents and you know the rules, it is quite easy to read the GTC documents without checking for the currency. That makes it easier to debug GTC documents in a multi currency environment.
- The divisor 1000 has another advantage. It is very natural to represent (metric) units in it, e.g. 1kg = 1000g. Units are included in GTC with respect to ***Regular Sales Line Items***.

Putting this all together, it was decided to represent all floats as integers using the divisor 1000 in GTC documents.

Which fields are affected?

It is very easy to identify fields that contain such a value. The names of fields follow a naming convention. All fields which names end with one of the following suffixes are represent as (float) integers:

. . . Amount
The field represents some form of monetary amount. Example: 12,27€ = 12270

. . . Quantity
The field represents a quantity based on units. Example: 0.223kg = 223

. . . Units
The field represents a unit which measures quantities. Example: 1kg = 1000

. . . Price
The field represents a price. Example: 2,99€ = 2990

. . . Percent
The field represents a percentage (tax or discount). Example: 7.275% = 7275

. . . Rate
The field represents a percentage (usually a discount). Example: 20% = 20000

. . . Total
The field represents a monetary amount (usually a transaction total). Example: 1357,24 = 1357240

. . . Count
The field represents a count. Example: 36 = 36000

1.5 Versions

Every GTC document includes information about the version of the schema that the document follows. In essence, the version defines the expected structure of the GTC document. The structure determines which entities should be present, under which conditions they might be present and the fields for each entity together with their exact meaning.

The version number will only increase whenever there were structural changes. Changes will always be backward compatible. Therefore, in principle, only new entities or fields will be added to the schema. Changes to the schema usually occur when new or extended functionality was implemented in the POS which adds new information stored together with transactions. That implies that a consumer will always be able to process documents which are sent by the latest POS version even if the consumer software was not adapted (but, of course, the new functionality is then not available to the consumer).

The specification also includes certain enumerations (usually called type codes), which are presented in tables and define the range of values that are applicable to specific fields. Whenever new POS functionality is implemented or existing functionality is extended, then new values in those enumerations might be available. In that case the document is updated but no new version is issued. New version numbers are issued only, if structural changes happen. Therefore, consumers should always be implemented in a way so that they can handle also unknown enumeration values in a safe way or update the software in a timely manner.

2 GTC Container Structure

This chapter defines the principle structure and the content of a GTC document. It also defines the data which is stored with a transaction for all available business cases.

2.1 Top Level Structure

The following JSON structure shows the top level layout of a GTC container.

Figure 1: Top Level GTC JSON Structure

```
{
    "version": "1.0",
    "type": "day",
    "tenantId": "0001",
    "companyId": 897,
    "locationId": 5315,
    "periodId": 100,
    "businessDay": "2018-03-09",
    "sequenceNumber": 1,
    "createdTimestamp": "2018-03-09T11:25:16Z",
    "transactions": [{}]
}
```

version
: declares the GTC version the document follows. At the moment the field has the constant value "1.0".

type
: specifies the type of the container. If the value is log, then it contains a Log-Document. If the value is day, then it contains a Day-Document. The type is actually not ultimately necessary, but the POS-System provides it so that a consumer is able to perform a validity check.

tenantId
: contains the id of the tenant which hosts the particular company. The identifier is assigned by the POS-System.

companyId
: contains the id of the company which created the container. This is the id of the operator in the POS-System.

locationId
: contains the id of the location within the company which created the container. This is the id of the restaurant in the POS-System.

periodId
: contains the id of the period of the GTC document. A (reporting) period represents a virtual business day. This virtual business day must not coincide with a calendar day. Such a virtual business day (period) can span multiple calendar days or one calendar day might have multiple periods. The period id is sometimes also called "Z-Report Counter" and the id is used when the Z-Report is printed to enumerate all Z-Reports. The period id is unique and is incremented whenever a period (business day) is opened. The ***periodId*** can be used by a consumer to check if a Day-Document hasn't been received or got retransmitted because of possible transport layer issues. If a consumer also wants to run similar validity checks on Log-Documents, the ***sequenceNumber*** needs to be checked.

businessDay
: contains the calendar day when the period started. Despite what was said for the ***periodId***, in reality most of the time there is one period (virtual business day) happening per calendar day and this field gives the date of it.

sequenceNumber
: contains the sequence number of a GTC Log-Document. The field enumerates all Log-Documents that were created by the POS in the course of a period (business day). The ***sequenceNumber*** will always start with 1 for the first document of the period (which is always a ***Period Open Transaction***). By following the progress of the ***sequenceNumber***, a consumer can easily verify if a GTC Log-Document hasn't been received or got retransmitted because of possible transport layer issues.

  The ***sequenceNumber*** of a Day-Document contains the ***sequenceNumber*** of the last Log-Document that was created by the POS for the period.

createdTimestamp
: shows when the document was created.

transactions
: contains the list of transactions. A Log-Document contains one transaction only, a Day-Document can contain multiple transactions.

2.2 Transactions

The section ***transactions*** contains the transaction data of the document. It is itself divided into the following two subsections which are detailed in the next sections:

Figure 2: Top Level - Transaction JSON Structure

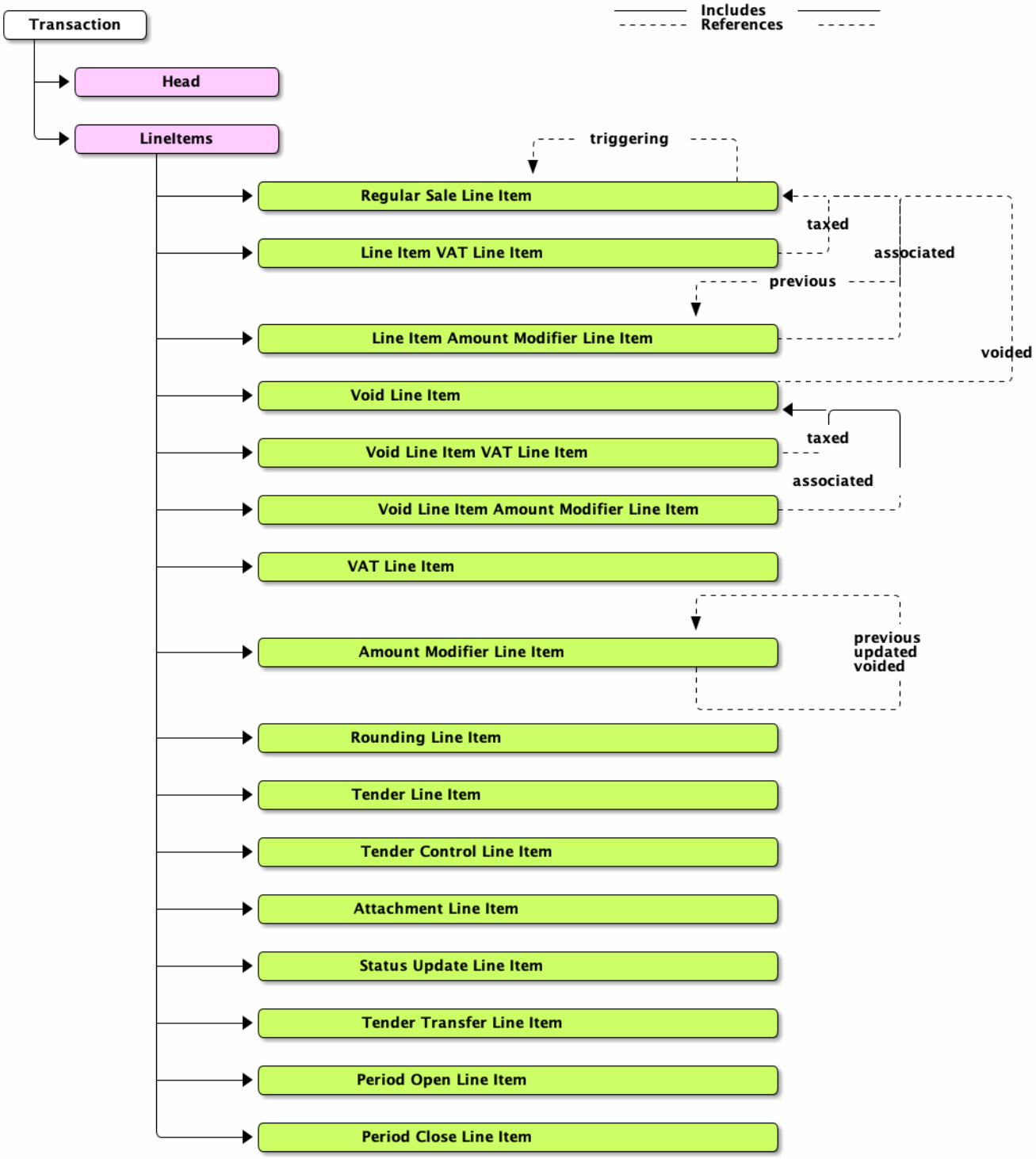
```
{
    ...
    "transactions": [{
        "head": {},
        "lineItems": [{}]
    }]
}
```

head
: contains the fundamental information of a transaction. They will never change and they are created together with the transaction.

lineItems
: contains the list of line items which constitute a transaction. The set of line items may describe a finalized or an incomplete transaction. A document of type day contains finalized transactions only. In contrast, a document of type log may contain a finalized or an incomplete transaction. The status is determined using the last ***Status Update Line Item*** of the Log-Document.

The next diagram provides a top-level view of the transaction structure. This jumps a bit ahead since it already shows all the available line items without describing them, but it might help to better understand the later sections when the details are discussed and you also might want to refer back to it while reading those sections.

Figure 3: Diagram - Transaction Structure


Includes
References
Transaction
Head
LineItems
triggering
Regular Sale Line Item
taxed
Line Item VAT Line Item
associated
previous
Line Item Amount Modifier Line Item
voided
Void Line Item
taxed
Void Line Item VAT Line Item
associated
Void Line Item Amount Modifier Line Item
VAT Line Item
previous
updated
voided
Amount Modifier Line Item
Rounding Line Item
Tender Line Item
Tender Control Line Item
Attachment Line Item
Status Update Line Item
Tender Transfer Line Item
Period Open Line Item
Period Close Line Item

2.2.1 Transaction Head

The transaction head contains information that will never change during the lifetime of a transaction. This information is created together with the transaction at the very first moment in time when a transaction object is needed to record data. That point in time could be the first order or just the moment when a party is seated at a table or when the POS-System needs to record some event (Sign-On, Sign-Off, Errors). Transactions that record events are a bit special in the sense that they usually contain only one line item which is a ***Status Update Line Item*** and they are also finalized immediately.

Figure 4: Transactions - Head JSON Structure

```
{
    "transactions": [{
        "head": {
            "typeCode": "00",
            "uuid": "a1e05418-9cfb-401e-b861-7611f413b647",
            "startedTimestamp": "2018-03-09T10:25:16Z",
            "trainingFlag": false,
            "voidedTrUuid": null,
            "referenceTrUuid": null
        },
    }]
}
```

typeCode

is a discriminator and denotes the type of a transaction. It indicates that a particular transaction is a ***Sale Transaction***, a ***Void Transaction***, a ***Control Transaction***, a ***Tender Control Transaction*** and so on. The following types are already available:

Table 1: Transaction Type Codes

Type Code	Transaction Type
’00’	Sale Transaction
’01’	Void Transaction
’20’	Funds Receipt Transaction
’21’	Disbursement Transaction
’22’	Tender Loan Transaction
’23’	Tender Pickup Transaction
’24’	Tender Loan Transaction On Behalf
’25’	Tender Pickup Transaction On Behalf
’26’	≪ RESERVED ≫
’27’	Tender Transfer Transaction
’30’	Amendment Transaction
’40’	Sign on Transaction
’41’	Sign off Transaction
’42’	Period Open Transaction
’43’	Period Close Transaction
’44’	No Sale / Drawer Opening
’60’	Error Log Transaction
’61’	Fiscal Initial Operation Transaction
’62’	Fiscal Shutdown Transaction
’63’	Firmware Update Transaction
’64’	TSE Log Upload Transaction
’65’	TSE Offline Log Transaction
’66’	Device Registration Transaction
..	See comment about new enumeration values in section 1.5.

uuid

uniquely identifies a transaction. A random UUID Version 4 as defined in RFC 4122 Section 4.4.

startedTimestamp
contains the timestamp when the transaction was started.

trainingFlag
indicates that the transaction was created while the POS was running in training mode.

voidedTrUuid
contains the UUID of the transaction which is voided by the transaction. Transactions can only be voided in the same period (business day). This field is null for a regular transaction, so it can be used to distinguish void and regular transactions.

referenceTrUuid
contains the UUID of a transaction. It creates a reference from on transaction to another. The exact meaning of the reference depends on the type of the transaction. The following table provides an overview when a reference is provided and what the reference is pointing to.

Table 2: Transaction Type Codes - Reference Transaction

Transaction Type Code	Transaction Reference
’60’	The transaction that caused the creation of an Error Log Transaction
..	See comment about new enumeration values in section 1.5.

2.2.2 Transaction Line Items

The section ***lineItems*** contains a list of line items that constitute a transaction. A line item represents a sale, a void, a payment, a status update or a multitude of other data which was entered while a transaction is conducted. Each line item is represented by a JSON dictionary object. The list may describe a finalized or an incomplete transaction. A document of type day contains finalized transactions only. In contrast, a document of type log may contain a finalized or an incomplete transaction. The set of line items which represent a transaction differ for the various transaction types.

Figure 5: Transactions - Line Items JSON Structure

```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "00",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {},
            "amounts": {},
            "timestamps": {},
            "extras": {}
        }]
    }]
}
```

typeCode
specifies the sub-type of the line item. The following table defines the currently supported types:

Table 3: Line Item Type Codes

Type Code	Transaction Line Item sub-types
’00’	Regular Sale Line Item
’01’ - ’05’	≪ RESERVED ≫
’06’	Void Line Item
’07’	VAT Line Item
’08’	Line Item VAT Line Item
’09’	Void Line Item VAT Line Item
’10’	Amount Modifier Line Item
’11’	Line Item Amount Modifier Line Item
’12’	Void Line Item Amount Modifier Line Item
’13’	Rounding Line Item
’14’	Tender Line Item
’15’	Tender Control Line Item
’16’	Attachment Line Item
’17’	Status Update Line Item
’18’	Tender Transfer Line Item
’19’	Table Party Lock Line Item
’20’	Period Open Line Item
’21’	Period Close Line Item
..	See comment about new enumeration values in section 1.5.

sequenceNumber
contains the sequence number of the line item within the confines of the transaction. The sequence number starts with 1 for the first line item of a transaction and is increased for each new line item. In essence, the field enumerates all line items of a transaction uniquely in the order in which they were created by the POS.

primary
contains the primary data of the line item. Usually it consists of identifying or vital information about the line item. The exact structure depends on the type of the line item and will be presented in the following sections which cover the different types.

related
contains ids to related entities connected to the line item. Usually it consists of ids to waiters, tables or other

entities that were involved while the line item was entered. The exact structure depends on the type of the line item and will be presented in the following sections which cover the different types.

flags

contains Boolean flags connected to the line item. Usually they indicate some states that were relevant while the line item was entered. The exact structure depends on the type of the line item and will be presented in the following sections which cover the different types.

amounts

contains amounts or totals connected to the line item. The exact structure depends on the type of the line item and will be presented in the following sections which cover the different types.

timestamps

contains timestamps connected to the line item. The exact structure depends on the type of the line item and will be presented in the following sections which cover the different types.

extras

contains additional/optional data connected to the line item. Usually the data is only entered in certain scenarios or variants of a business case. The exact structure depends on the type of the line item and will be presented in the following sections which cover the different types.

2.2.3 Regular Sale Line Item

The line item records the sale of a number of (measured) items based on a unit price.

Some general information:

- The line represents one entered position (Item, Price and Quantity).
- A line item is not created by combining multiple entered line items into one. Every line item is recorded as it happened at the POS.
- Not all fields are always set, but they are always present. The structure of a regular sale line item is fixed and empty fields still show up. This asserts that no data is missing and a consumer does not need to assume values for missing fields or assume a bug in the POS.
- Please also note how measured items are represented. They also use the field ***quantity***. The ***units*** play a crucial role here. If you sell single units, the ***units*** are set to 1 and the regular unit price is set to the item price. If for instance you sell a measured item and the base is 1000g, then ***units*** is set to 1000 and the regular unit price is set to the price for 1000g and the quantity is then e.g. 250 when 250g were bought.
- All line items are directly linked to the transaction. There is no special order entity, since there is no business process at the POS which requires access to a single order. Orders are just used to group line items which were ordered together. This is achieved by adding a orderSequenceNumber which contains the same sequence number for all line items which were ordered together. Using the orderSequenceNumber a consumer can regroup them if required.

Figure 6: Line Items - Sale Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "00",
            "sequenceNumber": 1,
            "primary": {
                "orderSequenceNumber": 1
            },
            "related": {
                "itemSku": 18776,
                "divisionId": 27,
                "groupId": 112,
                "vatId": 1,
                "courseId": 2,
                "articleTypeId": 7,
                "deviceId": 17,
                "waiterId": 19,
                "tableId": 101,
                "costCenterId": 73,
                "roomId": 99,
                "unitId": 18,
                "servicePeriodId": null,
                "priceLevelId": null
            },
            "flags": {
                "isVoidFlag": false,
                "isSplitFlag": false,
                "isInfoFlag": false,
                "isToGoFlag": false,
                "isTransitoryFlag": false,
                "isNonDiscountableFlag": false,
                "isNegativeItemFlag": false
            },
            "amounts": {
                "quantity": 8000,
                "units": 1000,
                "regularUnitPrice": 2990,
                "regularAmount": 23920,
                "taxPercent": 19000
            },
            "timestamps": {
                "recordedTimestamp": "2018-03-09T14:36:11Z"
            },
            "extras": {
                "voidedLineItemSequenceNumber": null,
                "triggeringLineItemSequenceNumber": null,
                "itemName": "Flour",
```

```
                "itemShortName": "Flour",
                "itemKind": 0,
                "itemBarcode": "4045348090241",
                "divisionName": "Food",
                "groupName": "Corn",
                "unitName": "kg",
                "priceLevelName": null
                "itemEntryMethod": "02",
                "priceEntryMethod": "00",
                "waiterName": "Herr Mustermann",
                "deviceUdid": "c82032a815ec4a54928922ac7ccac7c593088833"
            }
        }]
    }]
}
```


orderSequenceNumber
contains the id of the order where the line item was added. Every order gets a unique id. The ***orderSequenceNumber*** starts with 1 for the first order of a transaction. It is increased by 1 for each new order in the context of the transaction. All line items with the same order sequence number were entered during the same ordering process.

itemSku
contains the sku (stock keeping unit - item id) of the sold item.

divisionId
contains the id of the top-level group in the merchandise hierarchy of the sold item. A typical merchandise hierarchy has the following structure:

```
Division --> Department --> Class --> (Product) Group --> Item (Article)
```

groupId
contains the id of the merchandise (article) group of the sold item.

vatId
contains the id of the VAT group which is assigned to the sold item.

courseId
contains the id of the course associated with the line item.

articleTypeId
contains the id of the article type of the sold item.

deviceId
contains the id of the device which was used to create the line item.

waiterId
contains the id of the employee who created the line item.

tableId
contains the id of the table for which the order and therefore the line item was created.

costCenterId
contains the id of the cost center for which the order and therefore the line item was created.

roomId
contains the id of the room for which the order and therefore the line item was created.

unitId
contains the id of the unit which was used to calculate the retail price based on the measured quantity.

servicePeriodId
contains the id of the service period in which the line item was created.

priceLevelId
contains the id of the price level that was used to calculate the retail price.

isVoidFlag
indicates that a line item voids/reverses another line item in the transaction. If the flag is set, ***voidedLineItemSequenceNumber*** contains the ***sequenceNumber*** of the line item which is voided by this line. The flag is always set to false for a regular sale line item. The void line item which is described later is a variation of

the sale line item. A void line item contains the same information like a regular sale line item except that the flag is set. They are closely related, so please also refer to the chapter covering the void line item. A regular sale line item and a void line item differ only in this flag and also the type code of the line item.

isSplitFlag
indicates if the line item was created during a table split.

isInfoFlag
indicates that the line was created because of an info item.

isToGoFlag
indicates if the line item was ordered to be consumed to-go, otherwise it is in-house.

isTransitoryFlag
indicates if the line item represents a transitory item. They are sometimes also called expenses or non-turnover items.

isNonDiscountableFlag
indicates if the line can not be discounted.

isNegativeItemFlag
indicates that the ***regularAmount*** is actually a negative amount. All amounts or amount-like values in a GTC document are always positive integer values. The Cloud Component of the POS System still allows that the user can provide negative prices for items. This is used by some users to implement functionality which is not yet supported by the POS System (like Deposit/Redemption). In case of a negative item (negative regular amount of a sale line item) this flag is set. The ***regularAmount*** is still positive.

quantity
contains the initial number of retail selling units sold to the customer (e.g. 1 piece, 235g, ...). What does initial mean? It is the number of units that were entered using a quantity preselection when the line was entered. Quantity preselection is the process of entering the quantity together with the item when the line is registered (e.g. 8x Shirts). If no quantity was entered initially, then the preselection quantity is set to the default 1 by the POS. The initial quantity can then later be modified using ***Line Item Amount Modifier Line Item*** in order to track correction/additions to the number of quantities and to update the line amount.

units
contains the number of units sold, when selling bulk merchandise (1 piece, 1000g, ...). That's what the regular price is based on.

regularUnitPrice
contains the regular per-unit price for the item before any discounts have been applied.

regularAmount
contains the regular total amount of the line item which is calculated by multiplying the quantity with the regular unit price and then divide it by the units.

```
regularAmount = quantity * regularUnitPrice / units
```

taxPercent
contains the tax percentage of the VAT group which is assigned to the item.

recordedTimestamp
contains the timestamp when the line item was sold / ordered.

voidedLineItemSequenceNumber
contains the ***sequenceNumber*** of the line item which is voided by this line item, otherwise it is null (remark: all sequence number start at 1).

triggeringLineItemSequenceNumber
contains the ***sequenceNumber*** of the line item which triggered the creation of this line item, otherwise it is null (remark: all sequence number start at 1). These kind of line items are created when a constraint or set item is ordered / sold.

itemName
contains the name of the sold item.

itemShortName
contains the short name of the sold item.

itemKind

defines the kind of the sold item. The kind shows if the item is a regular item or a composite item or a transitory item (expense item, voucher item, and so on). If the item kind is a transitory item, then the ***isTransitoryFlag*** needs to be set also. The following table shows all currently valid values:

Table 4: Item Kind Values

Value	Item Kind
0	Regular Item
1	Constrained Item (Composite Item)
2	Aggregated Item (Composite Item) (Hide Components)
3	Aggregated Item (Composite Item) (Show Components)
4	Voucher Item (Transitory Item)
5	Expense Item (Transitory Item)
6	EU Multi-Purpose-Voucher (EU-MPV)
7	EU Single-Purpose-Voucher (EU-SPV)
..	See comment about new enumeration values in section 1.5.

itemBarcode

contains the barcode (aka string) which was used to look up the item. The field is only set, if the ***itemEntryMethod*** is either "Keyed" or "Scanned", otherwise it is set to null.

divisionName

contains the name of the top-level merchandise (article) group of the sold item (see remarks on ***divisionId***).

groupName

contains the name of the merchandise (article) group of the sold item.

unitName

contains the textual description of the unit which is used to measure the item (e.g. kg or g). If the item is not a measurement item (sold by piece), then the field is set to null.

priceLevelName

contains the name of the price level that was used to calculate the retail price.

itemEntryMethod

denotes how the item/barcode was entered (e.g. Keyed, Scanned, . . .). The following table shows all codes:

Table 5: Item Entry Method Type Codes

Type Code	Entry Method
'00'	Lookup (Database)
'01'	Keyed (Entered using keypad)
'02'	Touched (Entered by pressing short access button)
'03'	Scanned (Barcode)
..	See comment about new enumeration values in section 1.5.

priceEntryMethod

denotes how the price was entered (e.g. Keyed, Scanned, Lookup). The following table shows all codes:

Table 6: Price Entry Method Type Codes

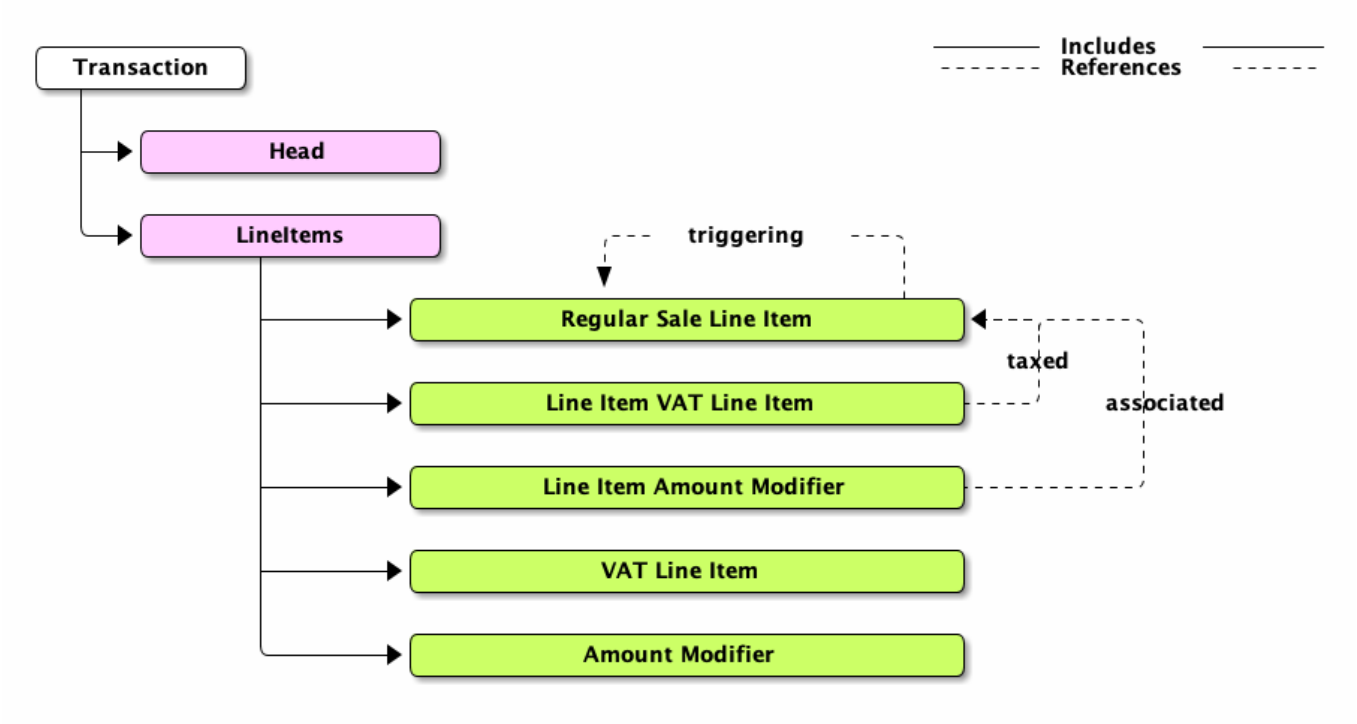
Type Code	Entry Method
'00'	Lookup (Database)
'01'	Keyed (Entered using keypad)
'02'	Touched (Entered by pressing short access button)
'03'	Scanned (Barcode)
..	See comment about new enumeration values in section 1.5.

waiterName
contains the name of the user/waiter who registered the line.

deviceUdid
contains the udid of the device which was used to create the line item.

The next diagram shows how the ***Regular Sale Line Item*** is linked with the other parts of a transaction.

Figure 7: Diagram - Regular Sale Line Item Structure



2.2.4 Void Line Item

A line item that represents the void of a previously registered ***Regular Sale Line Item***. It serves as the reverse posting of a sale line item. The ***Void Line Item*** and the ***Regular Sale Line Item*** are closely related. They bear the same structure, therefore please take a look at the sale line item to understand the fields and values. The void differs from the sale line item mainly in the ***typeCode*** and in the ***isVoidFlag*** fields. The ***typeCode*** tells that it is not a sale line item but a void line item ('06' instead of '00'). The ***isVoidFlag*** is kind of redundant since the ***typeCode*** and the ***isVoidFlag*** are actually coupled ('00' ⇔ false, '06' ⇔ true), but we keep it as denormalized information since a flag is more telling than just the ***typeCode***.

Some general information:

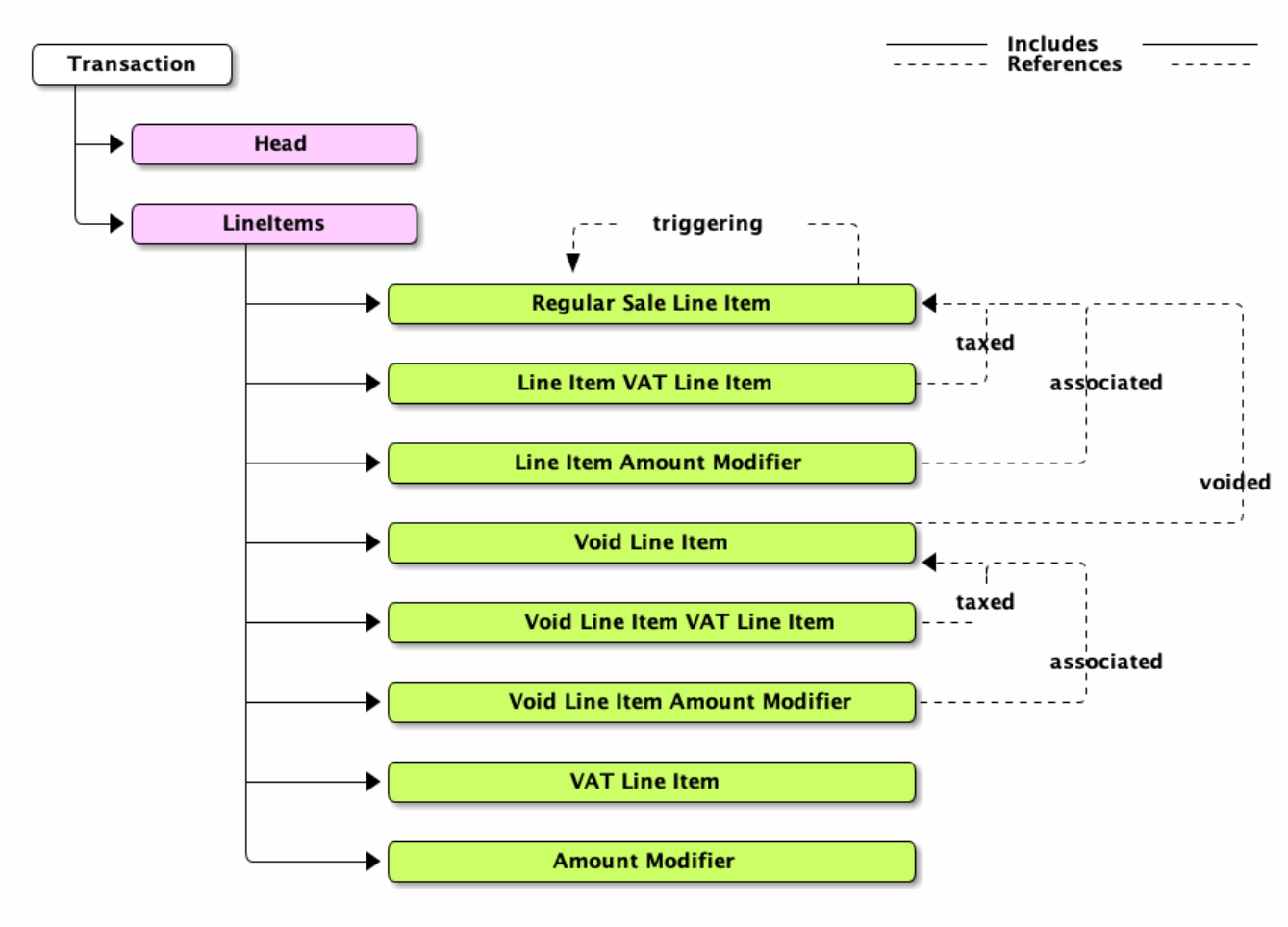
- A ***Void Line Item*** represents a reverse posting of a previously created ***Regular Sale Line Item***.
- The voided sale is referenced using the ***voidedLineItemSequenceNumber***.
- A ***Void Line Item*** contains the same sales data as the original ***Regular Sale Line Item***.
- A ***Void Line Item*** has no negative value. The void indicates that it is a reverse posting and therefore it needs to be handled properly by the consumer (which might require subtracting some values on its side).
- A ***Void Line Item*** always completely voids the original ***Regular Sale Line Item***. In other words, there are no partial voids on the line level. For instance, if the POS sold 5 beers and then voids 3, the POS will completely void the sale with the initial 5 beers and create a new ***Regular Sale Line Item*** which represents the remaining 2 beers, so there will be 3 line items in total in that void scenario.
- A ***Void Line Item*** also has linked Void Line Item VAT Line Items and Void Line Item Amount Modifier Line Items. They reverse the VAT and amount modifiers that are linked to the original sale line item.
- A void is done as a separate order with a unique ***orderSequenceNumber***.

Figure 8: Line Items - Void Line Item JSON Structure

```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "06",
            "sequenceNumber": 2,
            "primary": {
                "orderSequenceNumber": 2
            },
            "related": {
                ".....": "same a original sale line item"
            },
            "flags": {
                "isVoidFlag": true,
                ".....": "same a original sale line item"
            },
            "amounts": {
                ".....": "same a original sale line item"
            },
            "timestamps": {
                "recordedTimestamp": "2018-03-09T14:36:11Z"
            },
            "extras": {
                "voidedLineItemSequenceNumber": 1,
                ".....": "same a original sale line item"
            }
        }]
    }]
}
```

The next diagram shows how the ***Void Line Item*** is linked with the other parts of a transaction.

Figure 9: Diagram - Void Line Item Structure



2.2.5 VAT Line Item

A ***VAT Line Item*** represents the tax amount, net merchandise value and gross amount of a particular VAT group that are included in a transaction.

Some general information:

- VAT is paid on the subtotal of a VAT group with respect to a whole transaction. The subtotal is the sum of all ***Regular Sale Line Items*** of the transaction which contain an item that is categorized in that particular VAT group.
- A ***VAT Line Item*** of the VAT group represents the legally binding VAT values connected with the transaction.

Figure 10: Line Items - VAT Line Item JSON Structure

```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "07",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {},
            "amounts": {
                "taxAmount": 15970,
                "taxSalesNetAmount": 84030,
                "taxSalesGrossAmount": 100000,
                "taxPercent": 19000
            },
            "timestamps": {},
            "extras": {}
        }]
    }]
}
```

taxAmount
contains the total tax amount calculated for the VAT group based on the transaction.

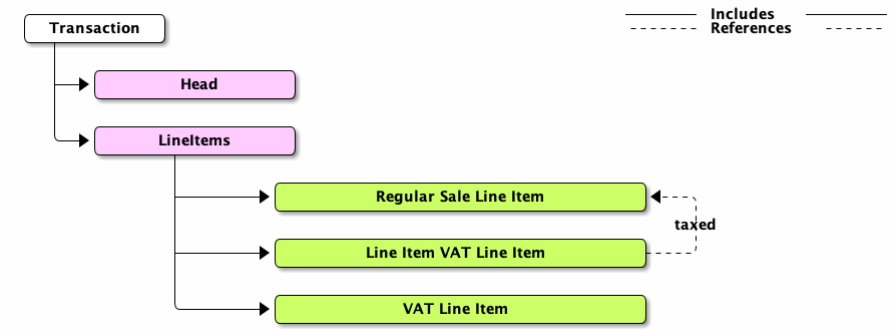
taxSalesNetAmount
contains the net merchandise value calculated for the VAT group based on the ***taxSalesGrossAmount***.

taxSalesGrossAmount
contains the cumulative sum of all sales affected by the VAT group.

taxPercent
contains the tax percentage of the VAT group.

primary, related, flags, timestamps and extras
are always empty.

Figure 11: Diagram - VAT Line Item Structure



2.2.6 Line Item VAT Line Item

A ***Line Item VAT Line Item*** represents the tax amount, net merchandise value and gross amount that are associated with one particular line item (position) of the transaction.

Some general information:

- VAT is paid on the subtotal of a VAT group with respect to a whole transaction. The subtotal is the sum of all ***Regular Sale Line Items*** of the transaction which contain an item that is categorized in that particular VAT group.
- A ***VAT Line Item*** of the VAT group represents the legally binding VAT values connected with the transaction.
- A ***Line Item VAT Line Item*** is in fact a convenience line item for a consumer, so that the consumer can see the estimated VAT connected with a ***Regular Sale Line Item***.
- The values in a ***Line Item VAT Line Item*** are derived from the ***VAT Line Item*** using a reconciliation calculation. The values in the ***Line Item VAT Line Item*** are therefore not exact. The reconciliation calculation assigns the values proportionately, based on the ratio of the amount of the ***Regular Sale Line Item*** to the subtotal of the VAT group.
- The VAT reconciliation calculation ensures that the sum of all ***Line Item VAT Line Items*** of a VAT group is equal to the legally binding values presented in the ***VAT Line Item*** of that VAT group.
- The VAT reconciliation calculation is done when the transaction is paid, since only then the transaction won't change anymore. Therefore the ***Line Item VAT Line Items*** are created at that point in time and a link to the connected ***Regular Sale Line Item*** is established.
- A ***Line Item VAT Line Item*** is connected to one particular ***Regular Sale Line Item*** and represents the VAT values for that line item. The ***taxedLineItemSequenceNumber*** identifies the connected ***Regular Sale Line Item***.

Figure 12: Line Items - Line Item VAT Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "08",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {},
            "amounts": {
                "taxAmount": 15970,
                "taxSalesNetAmount": 84030,
                "taxSalesGrossAmount": 100000,
                "taxPercent": 19000
            },
            "timestamps": {},
            "extras": {
                "taxedLineItemSequenceNumber": 1
            }
        }]
    }]
}
```


taxAmount
: contains the calculated tax amount of the connected ***Regular Sale Line Item*** as determined by the reconciliation calculation (details above).

taxSalesNetAmount
: contains the calculated net merchandise value of the connected ***Regular Sale Line Item*** as determined by the reconciliation calculation.

taxSalesGrossAmount
: contains the amount of the connected ***Regular Sale Line Item***. The amount is used as the basis for the reconciliation calculation.

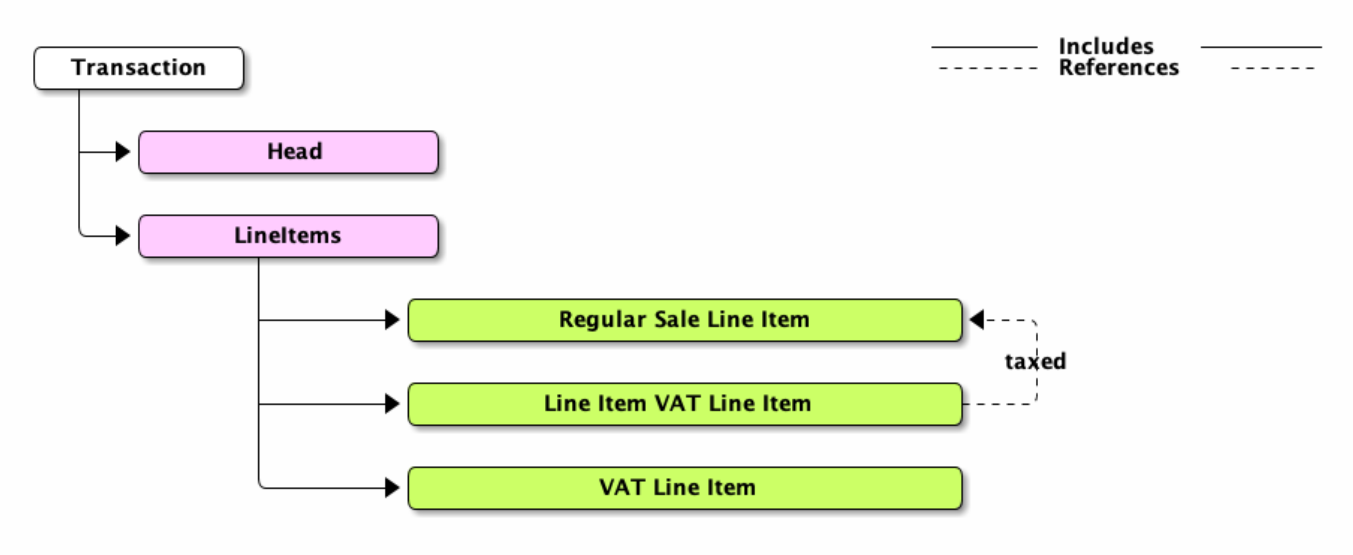
taxPercent
: The tax percentage of the VAT group.

taxedLineItemSequenceNumber
: links the line item to a ***Regular Sale Line Item***.

primary, related, flags and timestamps
: are always empty.

The next diagram shows how the ***Line Item VAT Line Item*** is linked with the other parts of a transaction.

Figure 13: Diagram - Line Item VAT Line Item Structure



2.2.7 Void Line Item VAT Line Item

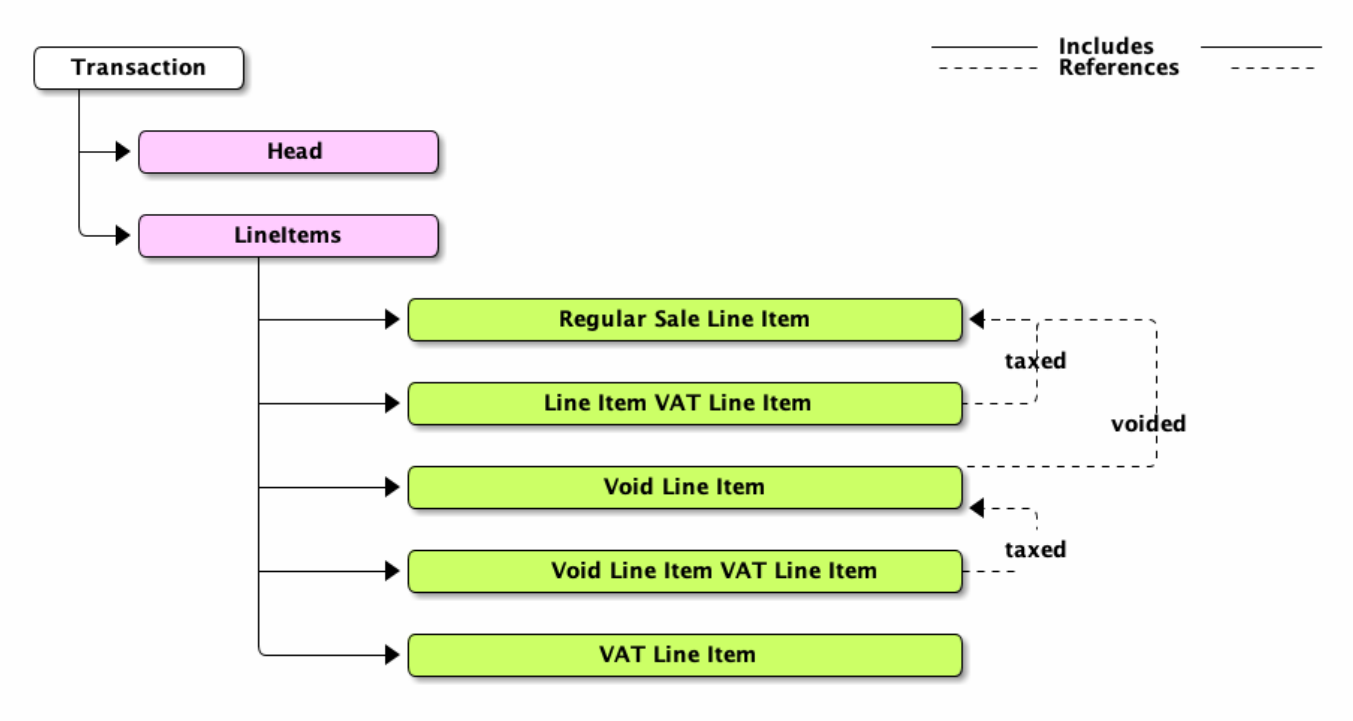
The ***Void Line Item VAT Line Item*** represents a reverse posting of a VAT previously captured with a Line Item VAT Line Item. In contrast to the ***Line Item VAT Line Item*** which is linked to a ***Regular Sale Line Item***, this line item type is always linked to a ***Void Line Item***. A detailed description of voids is available here.

Some general information:

- The ***Void Line Item VAT Line Item*** has the same structure as the ***Line Item VAT Line Item***.
- It also contains the same tax information and amounts.
- All values are positive. The reverse posting is indicated by the line type and the consumer must take care of the proper calculations.
- The ***Void Line Item VAT Line Item*** can only be linked to a ***Void Line Item***. So, the ***taxedLineItemSequenceNumber*** points to a ***Void Line Item***.

The next diagram shows how the ***Void Line Item VAT Line Item*** is linked with the other parts of a transaction.

Figure 14: Diagram - Void Line Item VAT Line Item



2.2.8 Amount Modifier Line Item

An ***Amount Modifier Line Item*** tracks transaction discounts.

Some general information:

- A discount might be created / recorded before the transaction is finalized, then it sticks with the transaction and is applied during payment. Therefore, it also needs to be updated whenever new sale line items are added.
- A modifier line item records what discount was or will be applied (see above) to the transaction. A transaction discount can either be a percent discount or a fixed amount. There might be multiple transaction discounts applied to one transaction, if the business rules allow that at all. In that case, there are multiple ***Amount Modifier Line Items*** in sequence.
- A discount is always applied to the discountable amount of the transaction. The discountable amount is the sum of all discountable position amounts (***Regular Sale Line Items*** which are discountable). For every ***Regular Sale Line Item*** the discountable position amount is given by the amount which can still be discounted.
- The POS system only applies a partial discount, if the given discount is greater than the discountable amount.
- When a discount was applied, an additional discount is never applied to the original position amount, but to the already discounted position amount.
- In case of a sticky discount (the transaction is not finalized yet), it will also be applied to future line items whenever they are added. Therefore, the discountable amount of the transaction will change when new line items are added. Therefore a new ***Amount Modifier Line Items*** must be created which updates the already issued modifier. The previously issued modifier can then be discarded because it is obsolete.
- The discount on a transaction which is not finalized yet can also be deleted by the user. In that case an ***Amount Modifier Line Item*** is created which voids the previously issued modifier.
- The ***Amount Modifier Line Item*** is an entity that records what happened. It records that a transaction discount was applied and also which one. It is usually not used for postings or calculations by a consumer directly, since discounts can also be applied on the line level directly. Therefore, it is not necessary for a consumer to distinguish between line discounts and prorated transaction discounts (both which are represented by ***Line Item Amount Modifier Line Items***, see next section). So the posting of transaction discounts by a consumer should always be based on ***Line Item Amount Modifier Line Items*** attached to the sale line items. ***Line Item Amount Modifier Line Items*** for transaction discounts are created by the POS during discount reconciliation calculation when the transaction is finalized / paid.

Figure 15: Line Items - Amount Modifier Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "10",
            "sequenceNumber": 1,
            "primary": {},
            "related": {
                "discountId": 1,
                "reasonId": 1,
                "reasonClassId": 1
            },
            "flags": {
                "isUpdateFlag": false,
                "isVoidFlag": false
            },
            "amounts": {
                "previousAmount": 12500,
                "appliedPercent": 10000,
                "appliedAmount": 1250,
                "newAmount": 11250,
            },
            "timestamps": {},
            "extras": {
                "discountName": "VIP Discount",
                "reasonName": "Tainted",
                "previousLineItemSequenceNumber": 1
                "updatedLineItemSequenceNumber": 1
                "voidedLineItemSequenceNumber": 1
            }
        }]
```

```
    }]
}
```

discountId
contains the id of the discount which was applied.

reasonId
contains the id of the reason which specifies why the discount was applied. A reason always belongs to a class. The class is given by the ***reasonClassId***. A reason is not mandatory. If no reason is provided, then the field is empty.

reasonClassId
contains the id of the reason class where the reason is grouped in. If no reason is connected with the amount modifier, then it is empty. There are 10 fixed general purpose discount classes available currently. They are used to categorize reasons into classes to better structure them. The following table shows the possible values:

Table 7: Class Ids

Id	Description
13	Discount Class 0
..	. . .
22	Discount Class 9
..	See comment about new enumeration values in section 1.5.

previousAmount
contains the amount the transaction had before applying the discount.

appliedPercent
contains the percentage if a percent discount was applied. It is 0, if a fixed discount amount was applied.

appliedAmount
contains the applied (discount) amount. It is always set even for a percent discount.

newAmount
contains the new amount of the transaction after applying the discount.

discountName
contains the name (description) of the applied discount.

reasonName
contains the name (description) of the reason.

previousLineItemSequenceNumber
contains the sequence number of a previously applied transaction discount ***Amount Modifier Line Item***. The field is only set, if multiple transaction discounts were applied, otherwise it is empty.

isUpdateFlag
indicates that the ***Amount Modifier Line Item*** updates a previously issued modifier line item. The previously issued ***Amount Modifier Line Item*** is then obsolete.

updatedLineItemSequenceNumber
contains the sequence number of the line item which is updated by the ***Amount Modifier Line Item*** and can be discarded since this new line item represents the correct and updated values.

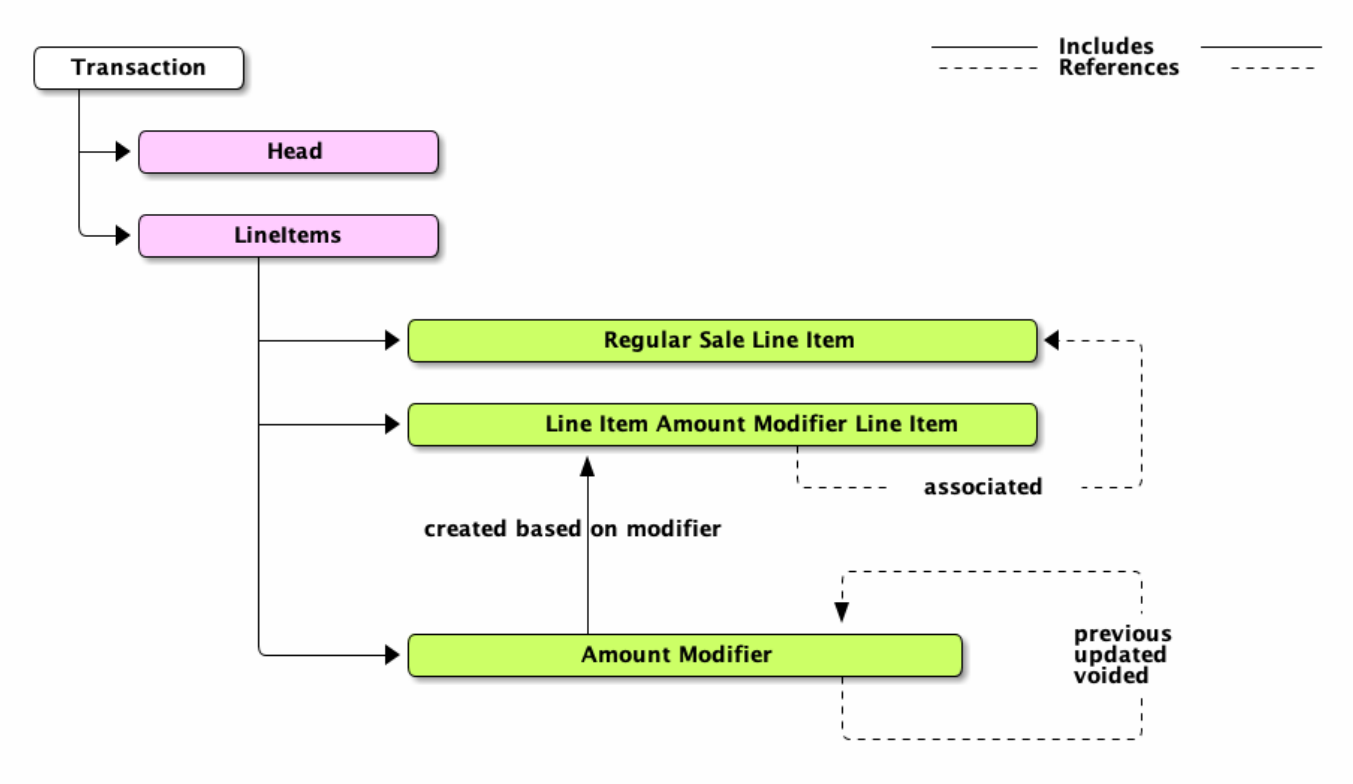
isVoidFlag
indicates that the ***Amount Modifier Line Item*** voids a previously issued ***Amount Modifier Line Item***. The referenced line item is then obsolete.

voidedLineItemSequenceNumber
contains the sequence number of the line item which was voided and can be discarded.

primary and timestamps
are always empty.

The next diagram shows how the ***Amount Modifier Line Item*** is linked with the other parts of a transaction.

Figure 16: Diagram - Amount Modifier Line Item



2.2.9 Line Item Amount Modifier Line Item

A ***Line Item Amount Modifier Line Item*** represents a change to the ***regularAmount*** of a ***Regular Sale Line Item***. The ***regularAmount*** can change for the following two reasons:

1. The quantity of sold items has changed which then results in a new total amount of the sale line item. The quantity is changed either by an addition (+) or correction (-) before the order is done. Once an order is completed (order button pressed) and the ***log*** document is issued, the quantity of a ***Regular Sale Line Item*** can not be changed anymore. In that case the original ***Regular Sale Line Item*** must be voided using a ***Void Line Item*** and then a new ***Regular Sale Line Item*** with the new quantity must be issued.
2. A discounts was applied to a ***Regular Sale Line Item***. The modifier then might represent a line discount which applies to just one particular sale line item directly, or it might represent a transaction discount which was prorated on the line by running the discount reconciliation calculation as described in this section.

Some general information:

- ***Line Item Amount Modifier Line Items*** are always associated with one ***Regular Sale Line Item***. Each modifier represents a change to the amount of that sale line item. The amount changes either because a discount was applied or the quantity changed. A modifier is always based on a previous modifier (if there is one). If there is no previous modifier, so it is the first one, then it modifies the ***regularAmount*** or ***quantity*** of the ***Regular Sale Line Item***. All modifiers that are associated with a sale line item constitute a chain. The last modifier in that chain then gives the final/actual values for the amount and the quantity.
- The chain of ***Line Item Amount Modifier Line Items*** documents which changes to the ***regularAmount*** or ***quantity*** of the ***Regular Sale Line Item*** have happened and in what order.
- The final amount of a ***Regular Sale Line Item*** is given by the ***regularAmount*** of the sale line item if no amount modifiers are present, otherwise by the last amount modifier in the chain.
- The final quantity of a ***Regular Sale Line Item*** is given by the ***quantity*** of the sale line item if no amount modifiers are present, otherwise by the last amount modifier in the chain.
- The quantity can only be changed before the order is done (the order button is pressed). Otherwise a void operation is required.
- A modifier represents a discount, if the field ***appliedPercent*** is set (not null) (note: it's 0 for a fixed discount amount). Otherwise it is set to null and represents a change to the quantity. Those two cases are exclusive. A modifier can not represent a discount and a change to the quantity simultaneously.
- A line discount can be either a percent discount or a discount with a fixed amount.
- A ***Line Item Amount Modifier Line Item*** is always associated with one particular ***Regular Sale Line Item***. The field ***associatedLineItemSequenceNumber*** links the modifier with the sale line item.
- The ***Line Item Amount Modifier Line Items*** are linked using the ***previousLineItemSequenceNumber***. It points to the modifier which the current modifier is based on. If the modifier modifies the ***Regular Sale Line Item*** directly, then this field is null / empty.
- It is possible to have multiple line discounts (modifiers) applied to one ***Regular Sale Line Item***, but only if the business rules allow it. They could even be mixed with some other modifiers like a direct line discounts or changes to the quantity.
- The field ***isTransactionDiscountFlag*** distinguishes between line discounts and prorated transaction discounts. It is only valid of course, if the modifier represents a discount and not a change to the quantity.
- A discount is always applied to the discountable amount of the line only. The discountable amount is the amount which can still be discounted. If multiple discounts were already applied, the discount is never applied to the ***regularAmount*** of the ***Regular Sale Line Item***, but to the amount with the previous discounts already applied (which is given by the last modifier in the chain). The section Amount Modifier Line Item also discusses the topic "discountable amount".
- The ***Line Item Amount Modifiers Line Items*** associated with a transaction discount are created during payment. Please refer to the section Amount Modifier Line Item for a detailed discussion of the topic and how transaction discounts are handled.
- A ***Line Item Amount Modifier Line Item*** can not be voided directly unlike the Amount Modifier Line Item. It can only be voided by voiding the whole ***Regular Sale Line Item***.

Figure 17: Line Items - Line Item Amount Modifier Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "11",
            "sequenceNumber": 1,
            "primary": {},
            "related": {
                "discountId": 1,
                "reasonId": 3,
                "reasonClassId": 5
            },
            "flags": {
                "isTransactionDiscountFlag": false
            },
            "amounts": {
                "previousAmount": 12500,
                "previousQuantity": 8000,
                "appliedPercent": 10000,
                "appliedAmount": 1250,
                "appliedQuantity": null,
                "newAmount": 11250,
                "newQuantity": 8000
            },
            "timestamps": {},
            "extras": {
                "discountName": "VIP Discount",
                "reasonName": "Tainted",
                "associatedLineItemSequenceNumber": 1,
                "previousLineItemSequenceNumber": 1
            }
        }]
    }]
}
```


discountId
contains the id of the discount if one was applied. If the modifier represents a change to the quantity, the field is empty.

reasonId
contains the id of the reason which specifies why the discount was applied. A reason always belongs to a class. The class is given by the ***reasonClassId***. A reason is not mandatory. If no reason is provided, then the field is empty.

reasonClassId
contains the id of the reason class where the reason is grouped in. If no reason is connected with the amount modifier, then it is empty. There are 10 fixed general purpose discount classes available currently. They are used to categorize reasons into classes to better structure them. The following table shows the possible values:

Table 8: Class Ids

Id	Description
13	Discount Class 0
..	. . .
22	Discount Class 9
..	See comment about new enumeration values in section 1.5.

isTransactionDiscountFlag
if true, indicates that the modifier represents a prorated transaction discount, otherwise it represents a line discount. If the modifier represents a change to the quantity, the field is empty.

previousAmount
contains the amount the sale line item had before applying the discount or the change to the quantity.

previousQuantity
contains the quantity of items the sale line item had before applying the modifier. This field is always set, even when the modifier represents a discount. In that case the quantity does not change.

appliedPercent
contains the percentage if a percent discount was applied. It is 0, if a fixed discount amount was applied. If the modifier represents a change to the quantity, the field is empty.

appliedAmount
contains the applied (discount) amount. It is always set even for a percent discount. If the modifier represents a change to the quantity, the field is empty.

appliedQuantity
contains the applied quantity. It defines the change to the quantity. It is a positive value for additions and a negative value for corrections. If the modifier represents a discount, the field is empty.

newAmount
contains the new amount of the sale line item after applying the discount or the change to the quantity.

newQuantity
contains the new quantity of items sold with the sale line item. The field is always set, even when the modifier represents a discount. It that case it is equal to the ***previousQuantity***. Therefore, the last modifier in the chain can be used to get the actual amount and quantity of a sale line item.

discountName
contains the name (description) of the applied discount.

reasonName
contains the name (description) of the reason.

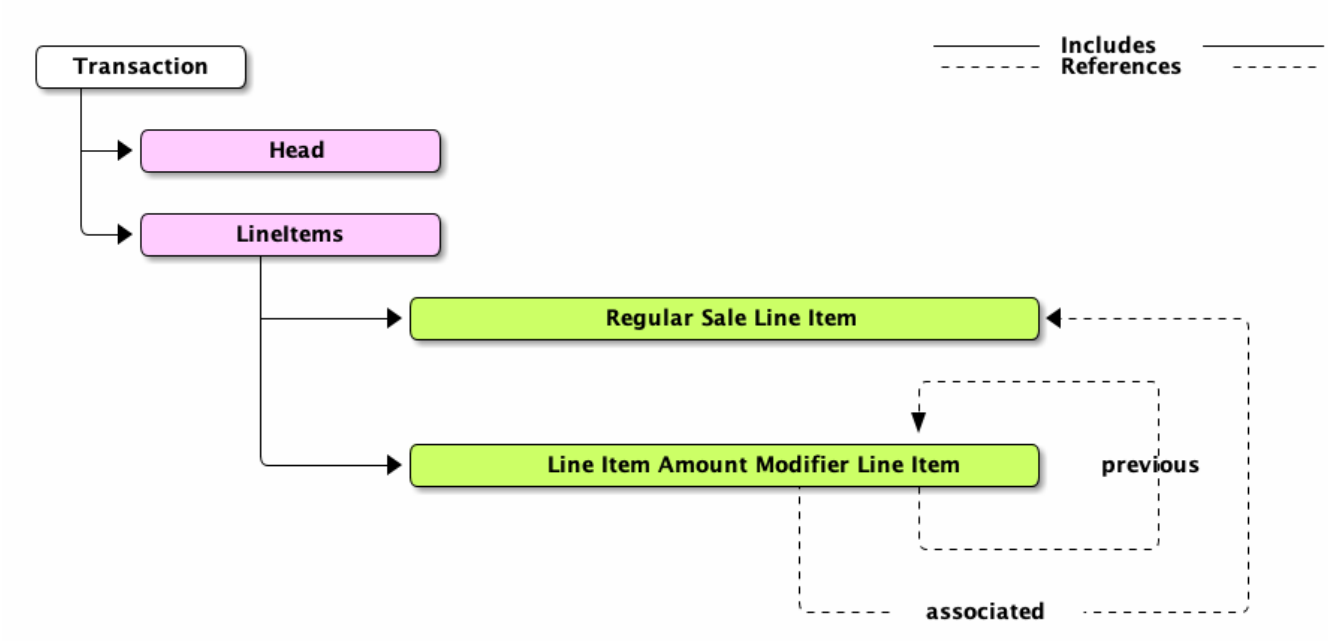
associatedLineItemSequenceNumber
associates the line item to a ***Regular Sale Line Item***. The discount or the change to the quantity applies to the associated line item.

previousLineItemSequenceNumber
contains the sequence number of the ***Line Item Amount Modifier Line Item*** which was applied before this modifier. The field is empty, if the modifier is based on the values in the ***Regular Sale Line Item***.

primary and timestamps
are always empty.

The next diagram shows how the ***Line Item Amount Modifier Line Item*** is linked with the other parts of a transaction.

Figure 18: Diagram - Line Item Amount Modifier Line Item



2.2.10 Void Line Item Amount Modifier Line Item

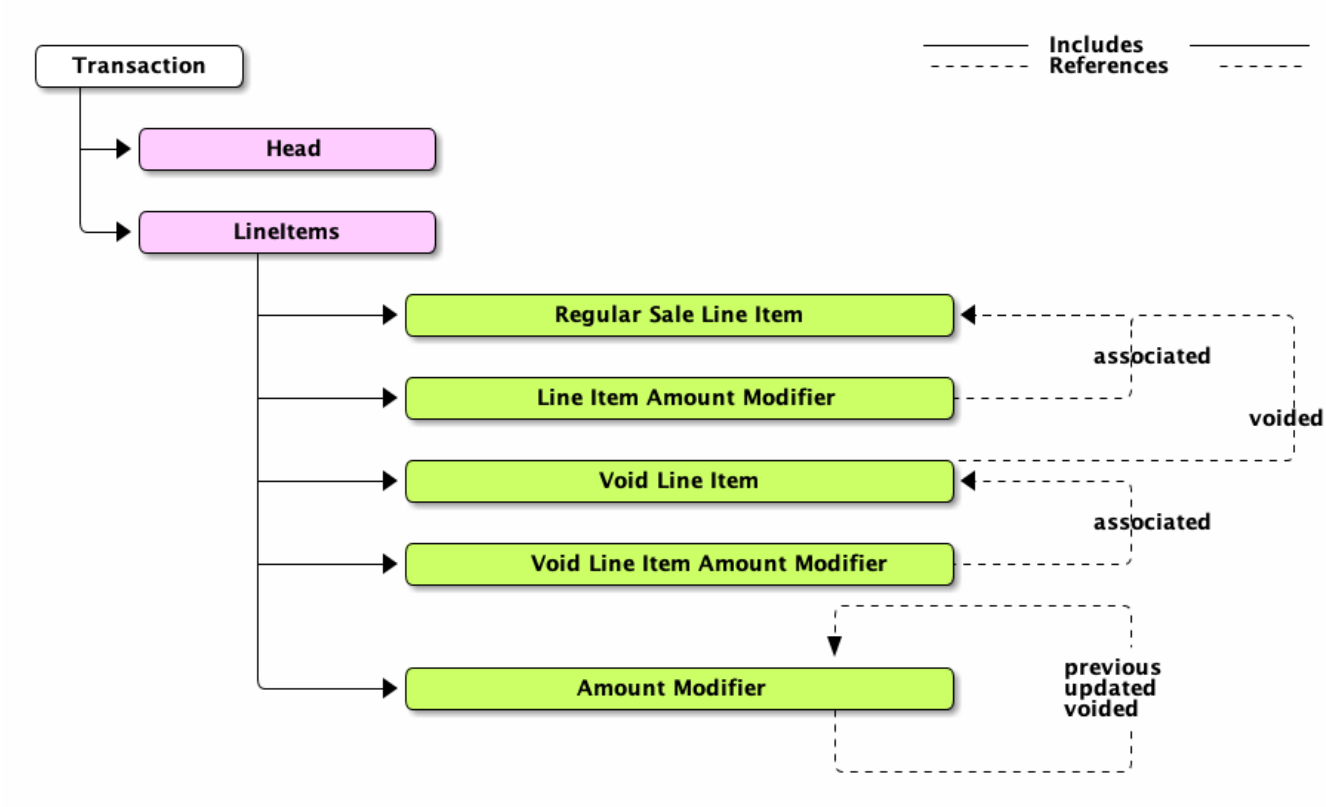
The ***Void Line Item Amount Modifier Line Item*** represents a reverse posting of a discount previously captured with a Line Item Amount Modifier Line Item. In contrast to the ***Line Item Amount Modifier Line Item*** which is linked to a ***Regular Sale Line Item***, this line item type is always linked to a ***Void Line Item***. A detailed description of voids is available here.

Some general information:

- The ***Void Line Item Amount Modifier Line Item*** has the same structure as the ***Line Item Amount Modifier Line Item***.
- It also contains the same discount information and amounts.
- All values are positive. The reverse posting is indicated by the line type and the consumer must take care of the proper calculations.
- The ***Void Line Item Amount Modifier Line Item*** can only be linked to a ***Void Line Item***. So, the ***associatedLineItemSequenceNumber*** points to a ***Void Line Item***.

The next diagram shows how the ***Void Line Item Amount Modifier Line Item*** is linked with other the parts of a transaction.

Figure 19: Diagram - Void Line Item Amount Modifier Line Item



2.2.11 Rounding Line Item

A ***Rounding Line Item*** records the adjustment of the transaction total to match the available denominations of the currency used to pay for the transaction. For instance, Switzerland supports 5 Rappen as the smallest cash denomination only, therefore they round up or down to the nearest 5 Rappen, when cash is used. If a transaction is paid with a non-cash tender, the transaction amount is not rounded.

Figure 20: Line Items - Rounding Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "13",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {
                "isRoundedUpFlag": true
            },
            "amounts": {
                "amount": 30
            },
            "timestamps": {},
            "extras": {}
        }]
    }]
}
```


isRoundedUpFlag
: indicates that the transaction total was rounded up by the given amount so that it matched the available denominations of the currency. It is set to false, if it was rounded down.

amount
: contains the amount by which the transaction total was rounded so that it matched the available denominations of the currency.

primary, related, timestamps and extras
: are always empty.

2.2.12 Tender Line Item

A ***Tender Line Item*** represents one payment item that was used to settle a transaction.

Some general information:

- A transaction can have multiple tender line items when multiple payment items were used to settle a transaction.
- A foreign currency can be used to (partially) settle a transaction. A foreign currency is a tender and not just an exchange rate.
- If a customer overpays a transaction, then the transaction also includes a ***Tender Line Item*** which represents the change given back to the customer. That ***Tender Line Item*** can be identified using the ***isChangeFlag***.
- The sum of all tender line items (in base currency) minus the change must be equal to the transaction total amount.
- The transaction total amount includes tip.

Figure 21: Line Items - Tender Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "14",
            "sequenceNumber": 1,
            "primary": {},
            "related": {
                "tenderId": 1,
                "customerUuid": "19c909c2-2177-494e-ad26-7889cf86f9a4"
            },
            "flags": {
                "isChangeFlag": false,
                "isNonTurnoverFlag": false
            },
            "amounts": {
                "tenderAmount": 50000,
                "foreignCurrencyAmount": 50000,
                "exchangeRate": 1000,
                "tipAmount": 3500,
                "tipForeignCurrencyAmount": 0,
                "appliedToTransactionAmount": 36500
            },
            "timestamps": {},
            "extras": {
                "tenderTypeCode": "00",
                "tenderName": "cash",
                "currencyCode": "EUR",
                "cardType": "",
                "cardLastFour": "",
                "cardExpiryDate": "",
                "cardHolderName": "",
                "cardFingerPrint": "",
                "cardTerminalId": "",
                "cardMerchantId": "",
                "cardApplicationId": "",
                "cardTerminalReferenceId": "",
                "cardAcquirerReferenceId": "",
                "cardAuthorizationCode": "",
                "cardVerificationMethod": "",
                "cardEntryMethodCode": "",
                "voucherSerial": "",
                "voucherProviderCode": "",
                "mobilePaymentProviderCode": "",
                "hotelPaymentProviderCode": "",
                "hotelReservationId": "",
                "hotelRoomId": "",
                "hotelRoomName": "",
                "hotelGuestName": ""
            }
        }]
    }]
}
```

tenderId
contains the id of the tender used to (partially) settle the transaction.

customerUuid
contains the uuid of the customer account which was used to pay for the transaction. The balance of the account is reduced. The account must have had a sufficient balance to start the payment process.

isChangeFlag
indicates that the line item represents a tender that is returned as change to the customer. This can also be understood that the amount left the drawer or wallet. This is an important information for the POS-System since it tracks how much of what tender is where. Essentially, this indicates to the POS-System that the balance of the affected tender needs to be reduced. This is also true for ***Tender Transfer Transaction***, therefore the flag needs to be set for the from ***Tender Line Item*** of a ***Tender Transfer Transaction*** for the same reason.

isNonTurnoverFlag
indicates that the line item actually uses a "fake" tender that represents a write-off. Every tender which is configured as a "non-turnover" tender in the POS-System is actually a write-off. The POS only allows to use one such tender to pay for the transaction completely.

tenderAmount
contains the monetary value submitted by the Customer in base currency.

foreignCurrencyAmount
contains the monetary value submitted by the Customer in the foreign currency. The amount is also set when the base currency is used, then it has the same value as ***tenderAmount***.

exchangeRate
contains the exchange rate used to convert the foreign currency amount into base currency. It is set to 1.00 when the base currency is used.

tipAmount
contains the monetary amount being collected as tip in base currency.

tipForeignCurrencyAmount
contains the monetary amount being collected as tip in the foreign currency. The amount is also set when the base currency is used, then it has the same value as ***tipAmount***.

appliedToTransactionAmount
contains the monetary amount in base currency applied to the transaction total (does not include tip).

tenderTypeCode
indicates which specific tender sub-type was used. The type code classifies the tender. The following table shows the available codes.

Table 9: Transaction Tender Line Item Type Codes

Type Code	Tender Type which was used during payment
'00'	Cash Base Currency
'01'	Cash Foreign Currency
'02'	Credit/Debit Card
'03'	Gift Certificate (Voucher)
'04'	Customer Account (Creditor, Balance based)
'05'	Customer Account (Debtor Payment, Invoice based)
'06'	Mobile Payment (Paypal, TWINT, . . .)
'07'	Hotel Payment (Ibelsa, Protel, Oracle)
'08'	Internal Payment (Only supported for backward compatibility)
'09'	EU Multi-Purpose-Voucher (EU-MPV)
'10'	EU Single-Purpose-Voucher (EU-SPV)
..	See comment about new enumeration values in section 1.5.

tenderName
contains the name of the tender.

currencyCode
contains the ISO 4217 code of the used currency e.g. USD or CHF.

cardType
contains a code which indicates the kind of card that was used to settle the transaction, e.g. Visa, Mastercard, see the following list for all the codes:

Table 10: Card Type Codes

Type Code	Type of Credit/Debit Card.
'00'	American Express
'01'	Diners
'02'	Discover
'03'	JCB
'04'	Mastercard
'05'	Visa
'06'	Maestro
'07'	V-Pay
..	See comment about new enumeration values in section 1.5.

cardLastFour
contains the last four digits of the credit card number if available.

cardExpiryDate
contains the expiry date of the card using format YYYYMM if available.

cardHolderName
contains the name of the card holder if available.

cardFingerPrint
contains the finger print of the card if available.

cardTerminalId
contains the id of the terminal reading the card data if available.

cardMerchantId
contains the merchant id of the card transaction if available.

cardApplicationId
contains the application id of the card transaction if available.

cardTerminalReferenceId
contains the reference number of the terminal for the card transaction if available.

cardAcquirerReferenceId
contains the reference number of the acquirer for the card transaction if available.

cardAuthorizationCode
contains the authorization code of the card transaction if available.

cardVerificationMethod
contains the code that indicates how the transaction was verified. Please see the following list:

Table 11: Card Entry Verification Methods

Type Code	Type of method.
'00'	PIN
'01'	PIN + Signature
'02'	Signature
'03'	No Verification required
..	See comment about new enumeration values in section 1.5.

cardEntryMethodCode
contains a code that indicates how the credit or debit card information was entered. Please see the following list:

Table 12: Card Entry Method Codes

Type Code	Type of entry methods.
'00'	Swiped (Magnetic Strip Reader)
'01'	NFC
'02'	Chip
'03'	Keyed (manual)
..	See comment about new enumeration values in section 1.5.

voucherSerial
contains the serial number that uniquely identifies a voucher.

voucherProviderCode
contains the type code that defines the voucher provider. Please see the following list:

Table 13: Voucher Provider Codes

Type Code	Entry Method
'00'	Lightspeed
'01'	Yovite
..	See comment about new enumeration values in section 1.5.

mobilePaymentProviderCode
contains the type code that defines the used mobile payment provider. Please see the following list:

Table 14: Mobile Payment Provider Codes

Type Code	Entry Method
'00'	Paypal
'01'	TWINT
'02'	Zapper
..	See comment about new enumeration values in section 1.5.

hotelPaymentProviderCode
contains the type code that defines the used PMS (hotel) system payment provider. Please see the following list:

Table 15: PMS / Hotel Payment Provider Codes

Type Code	Entry Method
'00'	Generic
..	See comment about new enumeration values in section 1.5.

hotelReservationId
contains the reservation id which is connected to the hotel payment.

hotelRoomId
contains the id of the room which is connected to the hotel payment.

hotelRoomName
contains the name of the room which is connected to the hotel payment.

hotelGuestName
contains the name of the guest. A room might host multiple guests, so it is important to also record who did

the payment.

primary and timestamps
are always empty.

2.2.13 Tender Control Line Item

A ***Tender Control Line Item*** records the movement of a certain amount of a tender in or out of a cash container. The more general term cash container is used here to make the description more general. A cash container stands for one of the following places where tenders are held.

Safe
: is a secure lockable box used for securing cash tenders when they are not needed right now. Cash is usually stored in the safe during the night and during closing hours and, of course, if there is excess cash.

Drawer
: is usually associated with a specific POS and contains cash tenders as well as slips for non-cash payments, e.g. credit cards or vouchers.

Waiter's Wallet
: is essentially a drawer which a waiter is carrying around.

A POS-System must track how much of what tender is in which cash container (where). This is required by many tax regulations and checked during tax audits. But it is also necessary for the business to monitor the payments and also to monitor employees who handle money. There are two principle ways a POS can work and track tenders:

1. A cash container is assigned to a POS directly.
2. A cash container is assigned to a waiter/user of the POS system.

The POS-System solely operates according to the second mode. That mode is actually more flexible since you can even emulate mode 1 by creating a dedicated waiter/user for every POS. Therefore, the terms "wallet" and "drawer" are just synonyms in the context of the POS-System. They might be used equally and are both used in the GTC-Specification.

As discussed, the POS-System tracks tenders per user/waiter. In order to achieve this, it maintains a virtual cash container for each user/waiter. Whenever the user/waiter performs a business case, the POS-System updates that virtual cash container immediately and accordingly in order to always know how much of what tender is where.

Since the POS tracks tenders per waiter, it needs a waiter for the safe as well. The safe user is automatically created by the POS-System when the corresponding feature is activated. There are also certain rules and restrictions that apply to the safe user. Please refer to the manual for further information.

Based on what is discussed above it can be stated, that the POS-System tracks tenders per waiters using virtual cash containers which are updated as a result of transactions. Every virtual cash container matches with one of the cash containers mentioned at the beginning of the section. Therefore, the POS-System uses the ***waiterId*** to address a cash container in a ***Tender Control Line Item***.

Tenders are moved in/out of a cash container as a result of one of the following business cases:

Loan
: A certain amount of a tender is moved from the safe to a waiter. Loans are used to fill a cash container with some initial coins and bills so that the user/waiter can give change to customers when the shift is started. Usually, this applies to the base currency only. The ***waiterId*** in the ***Tender Control Line Item*** defines the cash container which receives the loan. The balance of the safe user is reduced accordingly.

Pickup
: A certain amount of a tender is moved from the drawer or wallet of a waiter to the safe. It is performed to limit the loss in case of theft. Employees should just not have to much money in their drawers or wallets. The ***waiterId*** in the ***Tender Control Line Item*** defines the cash container which is picked. The balance of the safe user is increased accordingly.

Funds Receipt
: Money is (moved in) received in a way that is not connected to regular business operations (sales) from outside the business. But nevertheless it increases the amount of the affected tender in the receiving cash container. Typical reasons for receiving money in such a way are ***Rent***, ***Kid Fun Ride*** or ***Safe Deposit***. ***Safe Deposits*** move tenders from the bank to the safe and are represented as ***Funds Receipts*** where the safe is the receiving cash container. The ***waiterId*** in the ***Tender Control Line Item*** defines the receiving cash container (which might be the safe user). The balance of the receiving cash container is adjusted accordingly.

Disbursement

Money is (moved out) spent by the restaurant/store for reasons like ***Cleaning Supplies***, ***Office Supplies***, ***Waiter Tip Outs*** or ***Bank Deposit***. ***Bank Deposits*** move tenders from the safe to the bank and are represent as ***Disbursements*** where the safe is the spending cash container. The ***waiterId*** in the ***Tender Control Line Item*** defines the spending cash container (which might be the safe user). The balance of the spending cash container is adjusted accordingly.

The POS-System offers to use an input page to enter an amount when one of the above business cases is performed. The input page is essentially a simple calculator which helps the user to enter the amount by providing a page with input fields based on the denomination of the currency and the user simply enters the counted values per denomination unit. The user might also enter the amount directly and not use the input page.

If the user uses the input page, the POS creates multiple ***Tender Control Line Items*** for the transaction. Each line item records what the user has entered for one of the fields. Since transactions are supposed to record what happened, all entered value are captured in that case.

Some general information:

- A ***Tender Control Line Item*** is only valid in either of a ***Funds Receipt Transaction***, ***Disbursement Transaction***, ***Tender Loan (On Behalf) Transaction*** and ***Tender Pickup (On Behalf) Transaction***.
- ***Funds Receipt Transactions*** and ***Disbursement Transactions*** imply that the cash container of the ***safe user*** is adjusted as well.
- A Transaction can have many ***Tender Control Line Items***.
- All ***Tender Control Line Items*** of a transaction have the same ***waiterId***.
- Not all ***Tender Control Line Items*** of a transaction necessarily have the same ***tenderId***. The POS might pick up multiple tenders in one transaction.
- ***Tender Control Line Item*** might be structured according to the denomination, if the input page was used.

Figure 22: Line Items - Tender Control Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "15",
            "sequenceNumber": 1,
            "primary": {},
            "related": {
                "tenderId": 1,
                "waiterId": 1
            },
            "flags": {},
            "amounts": {
                "amount": 140000,
                "baseCurrencyAmount": 140000,
                "denomination": 20000,
                "count": 7000
            },
            "timestamps": {},
            "extras": {
                "tenderTypeCode": "00",
                "tenderName": "cash",
                "currencyCode": "EUR",
                "waiterName": "Herr Mustermann"
            }
        }]
    }]
}
```


tenderId

contains the id of the tender being moved.

waiterId

contains the id of the waiter whose cash container is updated.

amount
: contains the amount being moved. The value is given in the respective currency. The following statement always holds:

```
amount = denomination * count
```

baseCurrencyAmount
: contains ***amount*** converted to base currency. If the tender actually is in base currency, then ***baseCurrencyAmount*** equals ***amount***. If the tender represents a foreign currency, then the current exchange rate is used to convert ***amount*** to base currency.

denomination
: contains the denomination factor that was used to calculate ***amount***. If the amount was not entered using the input page but was entered directly, then the field is set to the smallest denomination of the currency, e.g. 10 for EUR or 50 for CHF. If the input page was used, then the value is set to the proper factor that corresponds to the input field, e.g. 20000 for 20€ bills.

count
: contains the number of tender units being moved, e.g. 20 x 100€ bills. If the amount was not entered using the input page but was entered directly, then ***count*** equals ***amount***. If the input page was used, then the value is set to the entered value of an input field.

tenderTypeCode
: identifies the type of tender. A table with the possible codes is available here where the tender type code of the tender line item is documented.

tenderName
: contains the display name of the tender.

currencyCode
: contains the ISO 4217 code of the currency e.g. USD or CHF.

waiterName
: contains the display name of the waiter whose cash container is updated.

2.2.14 Tender Transfer Line Item

A ***Tender Transfer Line Item*** records the transfer of a certain amount from one tender to another tender. Since tenders can be seen as accounts that the POS maintains to track the balance which then might be used for the XZ-Reports and so on, the statement could also have been phrased that the line item records the transfer of money from one account to another. A ***Tender Transfer Line Item*** can **only** appear in the context of a ***Tender Transfer Transaction***. A ***Tender Transfer Transaction*** has **usually** three line items as shown in the figure below. The transaction has two ***Tender Line Items*** and one ***Tender Transfer Line Item***. The two ***Tender Line Items*** contain the details about the amount that needs to be transferred. The ***Tender Transfer Line Item*** defines which ***Tender Line Item*** defines the outgoing tender account and which one is the receiving tender account. In essence, a ***Tender Transfer Transaction*** represents a correction that adjusts the amount of two tender accounts that are tracked by the POS. These kind of corrections are typically necessary if a wrong tender was used to finalize a transaction or if tip needs to be added/changed (see scenarios below). Since transactions are not modified at all after they have been finalized, an offsetting entry is needed that documents the correction. A ***Tender Transfer Transaction*** with its line items represents that correction.

Some general information:

- ***Tender Transfer Line Items*** are only valid in a ***Tender Transfer Transaction***.
- A ***Tender Transfer Transaction*** has usually two ***Tender Line Items*** and one ***Tender Transfer Line Item***.
- ***Tender Transfer Transactions*** are corrections.
- A correction can happen for one of the following reasons.

    1. A transaction was finalized with the wrong tender. This can happen by accident if the user is just to fast and presses the wrong button (e.g. cash instead of credit card). Or it might happen on purpose that a user/waiter closes the receipt with cash in order to give it to the customer and when the customer has paid a proper adjustment to the actual tender is needed. In these scenarios the ***Tender Transfer Transaction*** has two ***Tender Line Items*** and one ***Tender Transfer Line Item***. Since the transfer is connected to a particular transaction, the field ***referencedTrUuid*** of the ***Tender Transfer Line Item*** is set to document this connection. The reference can be used by any receiver or the POS to reconstruct the last status of the transaction.

    2. While counting the tenders at the end of the shift users might realize that they are over on one tender and short on another tender but the amounts offset. That typically means that the wrong tender has been picked accidentally during payments without noticing. In order to correct the X-Report a ***Tender Transfer Transaction*** would be performed. The POS would then show the correctly counted amounts on the X-Report. Here the ***Tender Transfer Transaction*** looks almost like case 1), except that the ***Tender Transfer Line Item*** has no reference to a transaction since it is not connected to an identifiable previous sale.

    3. The user receives an additional tip after the transaction is finalized. It needs to be noted that the tip can be given in a different tender than which was used to settle the transaction. It is not uncommon that customers pay the transaction with credit card and leave the tip in cash. Actually, the whole process of giving tip might also be seen as a Funds Receipt as implemented in the cash book, since the user receives money which is not connected to a sale. But it is seen as a tender transfer in this context and the reason "Tip" is provided by the ***transferTypeCode***. In this scenario the ***Tender Transfer Transaction*** has only one ***Tender Line Item*** and one ***Tender Transfer Line Item***. The transfer happens kinda out of thin air to one particular tender account (you can also see it as a transfer from an unnamed tender account to the particular tender account). Therefore, there exists no "from" tender in the transaction. The ***Tender Transfer Transaction*** has only one "to" ***Tender Line Item*** which is referenced in the ***Tender Transfer Line Item***. To be precise, the field ***fromLineItemSequenceNumber*** is null and ***toLineItemSequenceNumber*** references the only ***Tender Line Item*** and the ***referencedTrUuid*** contains the id of the transaction which receives the tip.

    4. The user needs to modify the given tip amount for some reason. This is actually almost equal to case 1), but in order to better distinguish what happens at the POS this scenario gets a different type code.

- In case that a transfer is done from Cash to Mastercard, the POS will require that the holder is present and that the authorization happens with the EFT terminal. Authorization might also be necessary for other "to" tenders, if the tender type requires it (e.g. Voucher or Hotel). These rules depend on the actual type of the "to" tender and maybe even on the region or country.

Figure 23: Line Items - Tender Transfer Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "18",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {},
            "amounts": {},
            "timestamps": {},
            "extras": {
                "transferTypeCode": "00",
                "fromLineItemSequenceNumber": 1,
                "toLineItemSequenceNumber": 2,
                "referencedTrUuid": "d60f72b0-1bc1-42a6-b445-722ccfeaca79"
            }
        }]
    }]
}
```


typeCode
defines which kind of correction is documented with the ***Tender Transfer Line Item*** (see above)

Table 16: Tender Transfer Type Codes

Type Code	Tender Transfer Type
'00'	Case 1) Wrong tender during payment.
'01'	Case 2) Transfer posting without knowing the corresponding transactions.
'02'	Case 3) Tip received after a transaction is finalized.
'03'	Case 4) Tip is changed after a transaction is finalized.
..	See comment about new enumeration values in section 1.5.

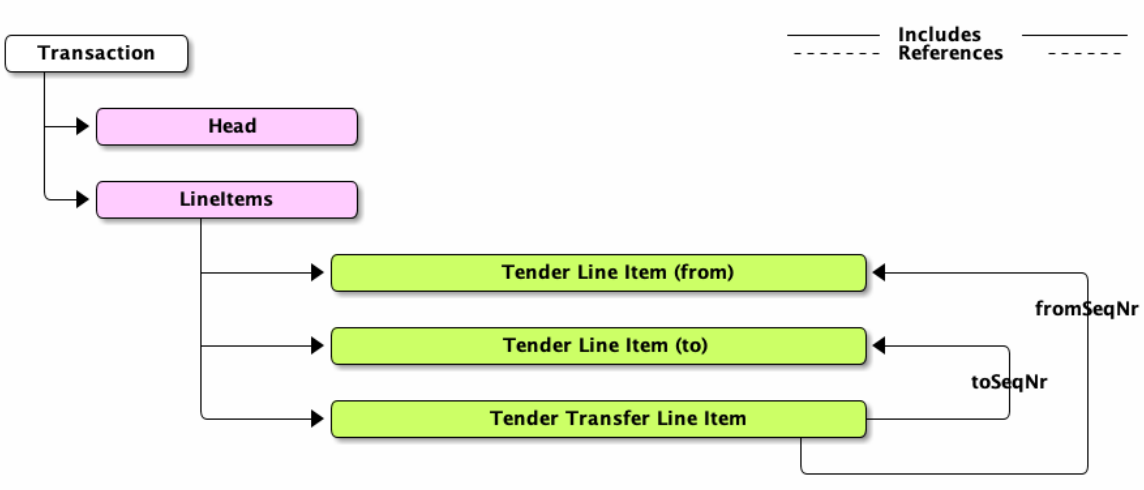
fromLineItemSequenceNumber
defines the ***Tender Line Item*** which represents the "from" tender. Is null, if case 3) (typeCode = '02') is recorded.

toLineItemSequenceNumber
defines the ***Tender Line Item*** which represents the "to" tender.

referencedTrUuid
can be set for cases 1) and 3) to reference the involved transaction.

primary, related, flags, amounts and timestamps
are always empty.

Figure 24: Diagram - Tender Transfer Line Item



2.2.15 Table Party Lock Line Item

A ***Table Party Lock Line Item*** records that a lock for a particular table/party combination has been acquired or released by the POS system. Locks are acquired before any sale related business cases are started. The POS prevents that concurrent orders can be created for a party at a table. Therefore, it always acquires a lock before a business case is started.

Some general information:

- Whenever an order for a party at a table is started, the POS will acquire a lock. When the lock was acquired, the POS will create a Log-Document which updates the transaction (see GTC flow diagram below).
- After the order was done by the waiter, the POS will release the lock. The set of GTC line items which represent the order will also include the ***Table Party Lock Line Item*** that releases the lock right before the ***Status Update Line Item*** (see GTC flow diagram below).
- A ***Table Party Lock Line Item*** will be created when the lock was acquired and also when it was released. The flag ***isAcquiredFlag*** then indicates, whether the lock was acquired or released.
- The POS will only continue with the business case when a lock was successfully acquired.
- A typical flow of GTC events will look like this. The POS will always send out two Log-Documents as shown in the following diagram:

Figure 25: Diagram - Table Party Lock GTC Flow

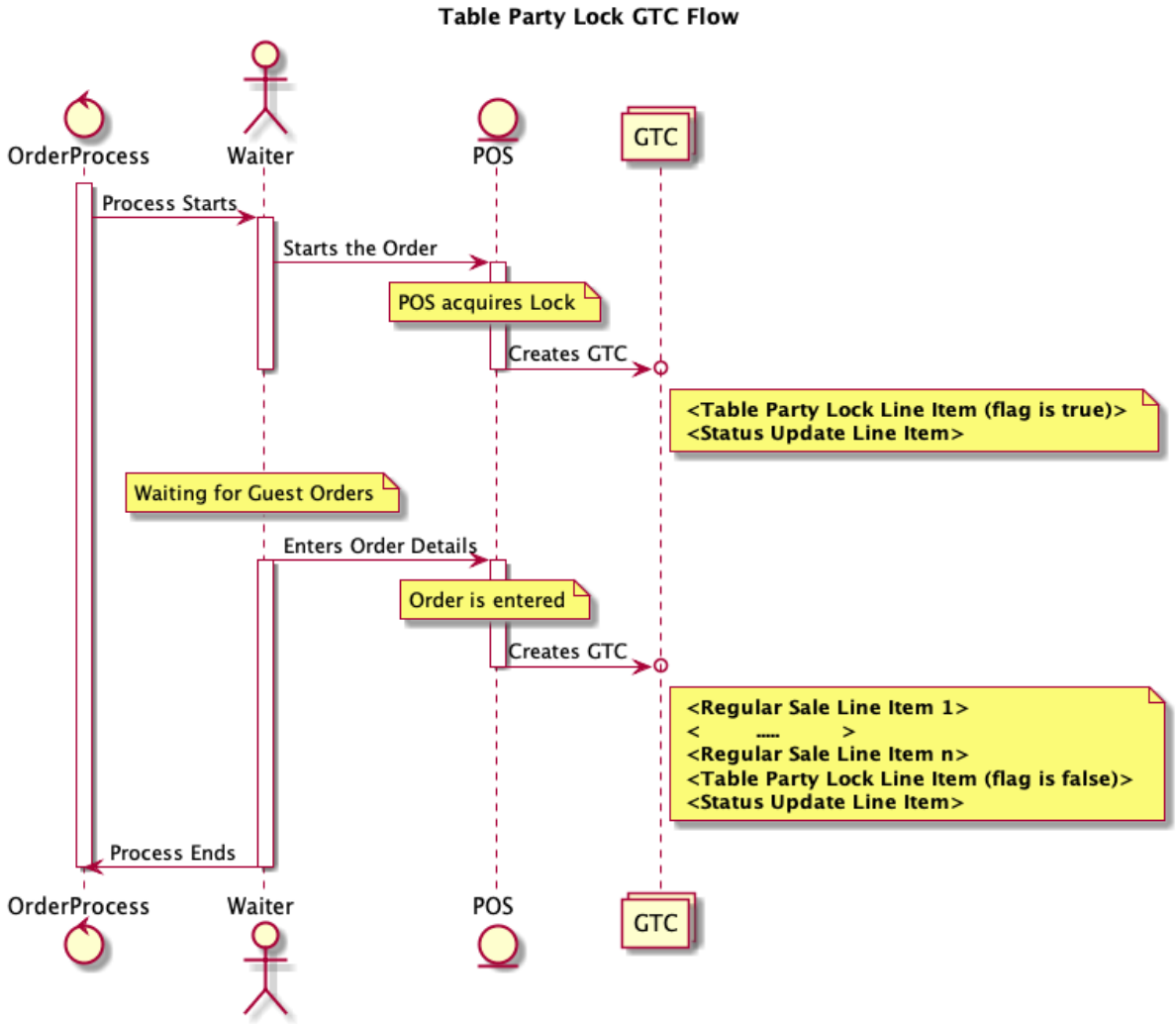


Figure 26: Line Items - Table Party Lock Line Item JSON Structure

```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "19",
            "sequenceNumber": 1,
            "primary": {},
```

```
            "related": {
                "deviceId": 17,
                "waiterId": 19,
                "tableId": 101
            },
            "flags": {
                "isAcquiredFlag": true
            },
            "amounts": {},
            "timestamps": {},
            "extras": {
                "party": 1,
                "partyName": null,
                "waiterName": "Herr Mustermann",
                "tableName": "Comfy Corner",
                "deviceUdid": "c82032a815ec4a54928922ac7ccac7c593088833"
            }
        }]
    }]
}
```


deviceId
contains the id of the device which was used to acquire or release the lock.

waiterId
contains the id of the employee who acquired or released the lock.

tableId
contains the id of the table for which the lock was acquired or released.

isAcquiredFlag
indicates if the ***Table Party Lock Line Item*** represents an acquired lock or a released lock. If true, the table lock was acquired, otherwise a table lock was released.

party
contains the id of the party at the table for which the lock was acquired or released.

partyName
contains the name of a party, if one was provided by the waiter.

waiterName
contains the name of the user/waiter who acquired or released the lock.

tableName
contains the name of the table for which the lock was acquired or released.

deviceUdid
contains the udid of the device which was used to acquire or release the lock.

2.2.16 Attachment Line Item

An ***Attachment Line Item*** is used to attach arbitrary data to a transaction. This can be a customer photo, or a receipt that was scanned for a disbursement, or the generated Z-report when a period is closed and so on. This is a very flexible mechanism to let the POS send any sort of data to the cloud. The line item uses a blob to contain the data. The blob can contain Base64 encoded binary data as well as not encoded text or JSON/YAML/XML structures. The blob should only use a Base64 encoding, if it contains binary data.

Figure 27: Line Items - Attachment Line Item JSON Structure

```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "16",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {},
            "amounts": {},
            "timestamps": {
                "recordedTimestamp": "2018-03-09T14:36:11Z"
            },
            "extras": {
                "attachmentTypeCode": "00"
                "attachmentSubTypeCode": "00"
                "blob": "ewVhZCI6IHsKICAgICAgICAiZG9jdW1lbnRUeXBlIjogIg=="
            }
        }]
    }]
}
```

recordedTimestamp
: contains the timestamp when the line item was created.

attachmentTypeCode
: defines what kind of object is in the blob. The following table shows the currently supported types of attachments.

Table 17: Attachment Type Codes

Type Code	Types of available transaction attachments
’00’	Image (JPEG/PNG Base64 encoded)
’01’	Period Close Info (JSON)
’02’	Period Close Z-Report Copy (Text)
’03’ - ’09’	≪ RESERVED ≫
’10’	Error Log
’11’	TSE Log Tar File
’20’	Device Registration Information
’21’	TSE Status Information

attachmentSubTypeCode
: defines a subcategory to the ***attachmentTypeCode***. It is for instance used to specify exactly what kind of error is recorded in an ***Attachment Line Item*** with the type ’10’ (Error Log). It might be used for other ***attachmentTypeCodes*** as well.

Table 18: Attachment Syb Type Codes

Type Code	Sub Type Code	Description
’00’	’00’	Disbursement Invoice Image
’00’	’01’	Funds Receipt Invoice Image
’10’	’00’	GTC Error Log

blob
: contains the data.

primary, related, flags and amounts
: are always empty.

2.2.17 Status Update Line Item

A ***Status Update Line Item*** describes an update of a transaction.

Some general information:

- As discussed in the Introduction, a ***Status Update Line Item*** always marks the end of a Log-Document that updates a transaction.
- A Log-Document always contains a complete transaction and therefore it usually contains many ***Status Update Line Items***. Each status line item indicates that an update of a transaction was done. The number of ***Status Update Line Items*** in a Log-Document shows how many updates were done to a transaction. The last one marks the most recent update and is the reason why the Log-Document was created by the POS. Updates just add line items, they will never delete line items. In many cases these updates are just orders or the transaction is settled.
- So in general, a ***Status Update Line Item*** will always be the last line present for a transaction independent of the status of the transaction. This holds for Log- and Day-Document since the transaction data has the same structure in both.
- After receiving a GTC-Document, a consumer might jump from one ***Status Update Line Item*** to the next of a transaction to see what happened in between. A variation is that a consumer can identify what happened with respect to a transaction, if the last known ***sequenceNumber*** of a ***Status Update Line Item*** is recorded for that transaction and then the consumer can just process all line items from there to the current ***Status Update Line Item***.

Figure 28: Line Items - Status Update Line Item JSON Structure


```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "17",
            "sequenceNumber": 1,
            "primary": {
                "transactionSequenceNumber": 111100,
                "dayDeviceTransactionSequenceNumber": 30,
                "invoiceNumber": "12123344"
            },
            "related": {
                "deviceId": 1,
                "waiterId": 1,
                "costCenterId": 1,
                "roomId": 1,
                "tableId": 1,
                "customerUuid": 1,
                "reasonId": 1,
                "reasonClassId": 1,
                "servicePeriodId": 1,
                "secureElementId": 1
            },
            "flags": {
                "isCancelFlag": false,
                "isBusinessCaseCompleteFlag": false,
                "isSuspendedFlag": false,
                "isOfflineFlag": false,
                "isRestoreFlag": false
            },
            "amounts": {
                "regularSubTotal": 199000,
                "actualSubTotal": 150000,
                "discountSubTotal": 49000,
                "tipSubTotal": 20000,
                "overAllSubTotal": 170000
            },
            "timestamps": {
                "updatedTimestamp": "2018-03-09T10:25:16Z",
                "signatureStartTimestamp":  "2019-07-19T12:37:42Z",
                "signatureEndTimestamp":  "2019-07-19T12:39:17Z"
            },
            "extras": {
                "waiterName": "Herr Mustermann",
                "reasonName": "Bank Deposit",
                "reasonCategoryCode": "00",
                "deviceUdid": "c82032a815ec4a54928922ac7ccac7c593088833",
```

```
                "party": 1,
                "partySize": null,
                "partyName": null,
                "comment": "",
                "appVersion": "",
                "signatureTransactionId": 123,
                "signatureCounter": 654321,
                "signatureSequenceNumber": 1,
                "signatureReferenceNumber": 1,
                "signatureProcessType": "Kassenbeleg-V1",
                "signatureProcessData": "1;\"Eiskaffee\";2.99",
                "signatureData": "..............",
                "signatureSerial":  "brrEJ3hWpk3K17jwQszm1wprC8rE+OhaA1dHxRBnzR0=",
                "signatureAlgorithm": "ecdsa-plain-SHA512",
                "signatureDateTimeFormat": "generalizedTime",
                "signaturePublicKey": "MIIEpAIBAAKCAQEAt...QuUlfkl5hR7LgwF+3q8bHQ==",
                "signatureError": "Online",
                "customerName",
                "customerGroupName",
                "customerCountryCode",
                "customerCity",
                "customerStreet",
                "customerPostalCode"
            },
            "salesSummaries": {
                " ..... ": "see document section 'Sales Summaries'"
            }
        }]
    }]
}
```


transactionSequenceNumber
: contains the unique serial id of a transaction. The transaction sequence number is a continuous, unique id which is assigned to every transaction by the POS-System. The sequence number is incremented for every transaction and runs forever.

dayDeviceTransactionSequenceNumber
: contains a unique serial transaction sequence number which enumerates all transactions done by a device during a period (business day). It starts with 1 for the first transaction done by a device for a period.

invoiceNumber
: contains the invoice / receipt number printed on the receipt. This number is also a unique serial id which is assigned to every fiscally relevant transaction and also runs forever. The invoice number is a string since the POS-System might support locally offline slaves in the future. In that case the slave must create its own invoice number which then will be a structured number to avoid collisions.

deviceId
: contains the id of the device which was used to update a transaction (create line items). If the transaction was finalized, contains the device which was used to process the payment.

waiterId
: contains the id of the employee who did the latest update to a transaction (create line items). If the transaction was finalized, contains the user/waiter who did the payment.

costCenterId
: contains the id of the cost center which is assigned to the device which did the update.

roomId
: contains the id of the room for which the update was done.

tableId
: contains the id of the table for which the update was done. If the transaction was finalized, contains the table which was paid.

customerUuid
: contains the uuid of the customer who is connected with the transaction. The field is empty, if no customer is connected.

reasonId
: contains the id of the reason which is currently connected with the transaction. The reason belongs to the class given by the ***reasonClassId***. If no reason is connected, the field is empty. Reasons are only used in the

context of ***Funds Receipt Transactions*** and ***Disbursement Transactions*** at the moment.

reasonClassId
contains the id of the reason class which is currently connected with the transaction. If a reason is connected, this field indicates which class the reason belongs to, otherwise it is empty. Classes are used to categorize reasons to better structure them. Reasons are only used in the context of ***Funds Receipt Transactions*** and ***Disbursement Transactions*** at the moment. The following table shows the possible values.

Table 19: Class Ids

Id	Description
9	Defines a reason of a Funds Receipt Transaction
10	Defines a reason of a Disbursement Transaction
..	See comment about new enumeration values in section 1.5.

servicePeriodId
contains the id of the service period for which the order and therefore the status update was created. A transaction can actually span multiple service periods, since every status update (order) is assigned the service period when it was done. The consumer must then decide based on its business rules which service period to use.

secureElementId
contains the id of the secure element that was used to create the digital signature if applicable.

isCancelFlag
indicates that the entire transaction has been canceled before it was completed at the POS. There will be no further line items created by the POS for the transaction.

isBusinessCaseCompleteFlag
indicates that the transaction is complete with respect to the business case it covers. The flag is set to true, when the transaction was settled or completed otherwise with respect to the business case. That means there will be no further line items issued by the POS-System that change the content of the business case. But the POS-System might issue further ***Attachment Line Items*** or ***Status Update Line Items*** that add supplementary information to the transaction (like a signature, printable representation and so on). ***Attachment Line Items*** or ***Status Update Line Items*** are the only line items that will be added by the POS-System after the flag changed to true. Once the flag changed from false to true, it will never revert back since the content can not get incomplete after it got complete already.

isSuspendedFlag
indicates that the transaction is not finalized yet (aka suspended), so the POS will create more line items for the transaction. The flag is set to false, when the transaction is finalized, then there will be also no further line items created by the POS for the transaction.

isOfflineFlag
indicates that the last line items were created by the POS while there was no connection to the cloud.

isRestoreFlag
indicates that a void is done because of a table restore. The flag actually serves as a default reason in that business case. The flag is set in the voiding transaction and not on the voided one.

regularSubTotal
contains the regular subtotal of the transaction up to this point in time. It represents the undiscounted subtotal of the transaction.

actualSubTotal
contains the actual subtotal of the transaction up to this point in time. It represents the discounted subtotal of the transaction.

discountSubTotal
contains the total discount given on the transaction. The following holds

```
regularSubTotal = actualSubTotal + discountSubTotal
```

tipSubTotal
contains the tip subtotal given by the customer.

overAllSubTotal
contains the overall subtotal of the transaction. The following holds

```
overAllSubTotal = actualSubTotal + tipSubTotal
```

updatedTimestamp
contains the timestamp when the ***Status Update Line Item*** was created.

signatureStartTimestamp
contains the timestamp which indicates when the business process that is secured with the digital signature was started. The timestamp is provided by the secure element. If the timestamp of the secure element can not be used since it is offline, then the POS uses its own timestamp.

signatureEndTimestamp
contains the timestamp which indicates when the business process that is secured with the digital signature was finished. The timestamp is provided by the secure element. If the timestamp of the secure element can not be used since it is offline, then the POS uses its own timestamp.

waiterName
contains the name of the user/waiter who did the latest update to the transaction.

reasonName
contains the name of the reason.

reasonCategoryCode
contains the category code of a connected reason. The code is only used in the context of ***Funds Receipt Transactions*** and ***Disbursement Transactions*** at the moment. The value is null, if a reason has no assigned category code. The following table lists the possible values:

Table 20: Reason Class Ids & Category Codes

Class Id	Category Code	Description
9	'00'	Safe Deposit
9	'01'	Private Investment
9	'02'	Balance Plus
10	'00'	Bank Deposit
10	'01'	Gratuity Payment
10	'02'	Wage Payment
10	'03'	Private Withdrawal
10	'04'	Balance Minus
..		See comment about new enumeration values in section 1.5.

deviceUdid
contains the udid of the device on which the last update to the transaction was registered.

party
contains the id of the party on the table that is involved in the transaction.

partySize
contains the number of people in that party (number of guests).

partyName
contains the name of a party, if one was provided by the waiter. A party can be named to easily find a suspended transaction by name.

comment
contains a free text comment that can be added to a transaction. It captures some arbitrary additional information.

appVersion
contains the version of the app that created the ***Status Update Line Item***.

signatureTransactionId
contains the transaction id that was issued by the secure element when the digital signature was created if applicable.

signatureCounter
contains the signature counter that was issued by the secure element when the digital signature was created if applicable.

signatureSequenceNumber
enumerates all digital signatures that were created for a transaction if applicable.

signatureReferenceNumber
contains the ***sequenceNumber*** of the ***Status Update Line Item*** for which the signature was created if applicable.

signatureProcessType
contains the official name of the business process if applicable. The official name is usually defined in tax regulations.

signatureProcessData
contains the original data of the business process that is used to create the digital signature (send to secure element) if applicable. The data is Base64 encoded.

signatureData
contains the digital signature that was created based on the given ***signatureProcessData*** and ***signature-ProcessType*** if applicable. The signature is Base64 encoded.

signatureSerial
contains the serial id of the secure element if applicable. The field is filled with the "TSE Serial" for Germany and with the serial of the certificate for Austria.

signatureAlgorithm
contains the algorithm that is used to create a signature if applicable. The field is filled with the "TSE Algorithm Identifier" for Germany and with the fixed string "AT2" for Austria.

signatureDateTimeFormat
contains a value of the enum as defined in DSFinV-K 2.0 Section 7 (TSE ZEITFORMAT) if applicable. The field is filled with the proper value defined by the TSE for Germany and with the fixed string "generalizedTime" for Austria.

signaturePublicKey
contains the public key of the secure element if applicable.

signatureError
contains a text that describes an error that happened during the creation of the digital signature if applicable. At least one of the following texts should be provided. This is not a complete list and depend on the actual implementation of the secure element and the tax regulations.

- "Online" when a digital signature was created successfully.
- "Offline" when a digital signature could not be created because the POS could not connect to the secure element.

salesSummaries
contains the sales data for the transaction. The structure of the sales summaries is described here. The field ***salesSummaries*** is only present with the last ***Status Update Line Item*** when the transaction is finalized. It includes only those summaries which correspond to items which show up in the transaction.

customerName
contains the full name of the customer who is connected with the transaction.

customerGroupName
contains the name of the group which the customer is assigned to.

customerCountryCode
contains the ISO 3166-1 alpha-3 country code of the customer's address.

customerCity
contains the city name of the customer's address.

customerStreet
contains the street name and house/unit number of the customer's address.

customerPostalCode
contains the postal code of the customer’s address.

2.2.18 Period Open Line Item

A ***Period Open Line Item*** is created as part of a ***Period Open Transaction***. It captures relevant data that needs to be recorded when a day is opened. The ***Period Open Line Item*** as well as the ***Period Close Line Item*** have subsections under the extras section which contain period specific data.

Some general information:

- A ***Period Open Line Item*** is only valid in a ***Period Open Transaction***.
- A ***Period Open Transaction*** contains exactly one ***Period Open Line Item***.
- The subsection ***openingBalances*** contains the opening balance of each cash tender for all waiters.

Figure 29: Line Items - Period Open Line Item JSON Structure

```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "20",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {},
            "amounts": {},
            "timestamps": {},
            "extras": {
                "openingBalances": {
                    "<waiterId>": [
                        {
                            "id": 1,
                            "typeCode": "00",
                            "currencyCode": "EUR",
                            "name": "Cash",
                            "amount": 123400,
                            "baseCurrencyAmount": 123400
                        }, {
                            "id": 2,
                            "typeCode": "01",
                            "currencyCode": "USD",
                            "name": "Dollar",
                            "amount": 250000,
                            "baseCurrencyAmount": 230000
                        },
                        { .... }
                    ],
                    "total": [ .... ]
                }
            }
        }]
    }]
}
```

openingBalances
contains a mapping that assigns an array of JSON objects to every configured waiter. Each object of the array represents the opening balance of a cash tender for the waiter. If the business does not accept foreign currencies, then the array will only have one entry which is the base currency. In the above example, "<waiterId>" represents the id of one of the configured waiters (e.g. "4711"). The mapping also contains a special key "total". An array object of "total" contains the sum of a cash tender over all waiters. The mapping has an entry for all configured waiters even if they have no closing balance of a cash tender in the previous period (contains 0 values). A configured waiter is an active waiter or an inactive waiter with an opening balance > 0 for one of the configured cash tenders. All cash tenders are always listed.

Essentially, the JSON structure represents a two-dimensional matrix. The first dimension is the list of configured waiters, the second dimension is the list of configure cash tenders and every cell contains the opening balance for the waiter-tender combination.

The following fields are present for each entry of the array:

id contains the id of the tender.

typeCode
identifies the type of the tender. A table with the possible codes is available here where the tender type code

of the tender line item is documented.

currencyCode
contains the ISO 4217 code of the currency e.g. USD or CHF.

name
contains the display name of the tender.

amount
contains the opening balance of the tender in the respective currency.

baseCurrencyAmount
contains the opening balance in base currency. If the tender actually is in base currency, then ***baseCurrencyAmount*** equals ***amount***. If the tender represents a foreign currency, then the current exchange rate is used to convert ***amount*** to base currency.

primary, related, flags, amounts and timestamps
are always empty.

2.2.19 Period Close Line Item

A ***Period Close Line Item*** is created as part of a ***Period Close Transaction***. It captures relevant data that needs to be recorded when a day is closed. The ***Period Close Line Item*** as well as the ***Period Open Line Item*** have subsections under the extras section which contain period specific data. Each subsection is explained in detail.

Some general information:

- A ***Period Close Line Item*** is only valid in a ***Period Close Transaction***.
- A ***Period Close Transaction*** contains exactly one ***Period Close Line Item***.
- The additional subsections of a ***Period Close Line Item*** contain necessary information to make a Day-Document self-contained. GTC is designed in such a way that a Day-Document or better a collection of Day-Documents can be used to run a tax audit without having access to the POS-System cloud database. That essentially means that a Day-Document needs to contain all information that are required to understand and verify the document solely based on the included data. The subsections provide this data. Some of the data is downloaded when the period is opened and the remaining data is created when the period is closed. The downloaded data is not allowed to change during a period.
- The subsection ***company*** contains the official information about the owner of the business.
- The subsection ***location*** contains information about the business location within the company. A company may operate multiple locations or just one location. A business location is a restaurant or a store or any another place where sales are done.
- The subsection ***vats*** contains information about all VAT groups that apply at the business location.
- The subsection ***devices*** contains information about the devices that were registered with the POS-System for the duration of the period at the business location. The subsection also contains information about the secure elements that were used to create digital signatures during the period.
- The subsection ***waiters*** contains information about the users/waiters that were configured with the POS-System during the period at the business location.
- The subsection ***businessCases*** contains information about the different business cases that were performed during the period. The data is relevant for the Z-Report and much of the data is printed on the Z-Report.
- The subsection ***payments*** contains the total received amount of each tender for all waiters per business case.
- The subsection ***closingBalances*** contains the closing balance of each tender for all waiters. The closing balance results from the opening balance plus payments.
- The subsection ***salesSummaries*** contains a summary of all sales done during the period. It shows how much was sold, voided and discounted during the period. It shows it on different levels of the merchandise hierarchy.

Figure 30: Line Items - Period Close Line Item JSON Structure

```
{
    "transactions": [{
        "lineItems": [{
            "typeCode": "20",
            "sequenceNumber": 1,
            "primary": {},
            "related": {},
            "flags": {},
            "amounts": {},
            "timestamps": {},
            "extras": {
                "company": { .... },
                "location": { .... },
                "vats": [{ .... }],
                "devices": { .... },
                "waiters": [{ .... }],
                "businessCases": { .... },
                "payments": { .... },
                "closingBalances": { .... },
                "salesSummaries": { .... }
            }
        }]
    }]
}
```

2.2.19.1 Company contains the official owner information as registered with the tax office. This might be an enterprise or a single person. The following JSON shows the structure of the ***company*** subsection.

Figure 31: Company JSON Structure

```
"company": {
    "id": 897,
    "name": "Burger & Grill",
    "countryCode": "DEU",
    "city": "Berlin",
    "street": "Alex-Wedding-Strasse 7",
    "postalCode": "10178"
}
```

id contains the id of the company. This is the id of the operator in the POS-System.

name
contains the name of the business owner.

countryCode
contains the ISO 3166-1 alpha-3 country code of the address.

city contains the city name of the address.

street
contains the street name and house/unit number of the address.

postalCode
contains the postal code of the address.

2.2.19.2 Location contains information about the business location within the company as registered with the tax office. The following JSON shows the structure of the ***location*** subsection.

Figure 32: Location JSON Structure

```
"location": {
    "id": 5315,
    "name": "Burgers",
    "taxId": "EU17787",
    "baseCurrency": "EUR",
    "timezone": "UTC+01:00",
    "countryCode": "DEU",
    "city": "Berlin",
    "street": "Alex-Wedding-Strasse 7",
    "postalCode": "10178"
}
```

id contains the id of the business location within the company. This is the id of the restaurant/store in the POS-System.

name
contains the name of the business location.

taxId
contains the tax id of the business location. A tax id is assigned by the tax office. Usually, business locations do not have their own tax ids. Usually, the field contains the tax id of the company.

baseCurrency
contains the ISO 4217 code of the base currency e.g. EUR.

timezone
contains the timezone of the business location. The value conforms to the regular expression `UTC[+-]\d\d:\d\d`. All timestamps in a GTC-Document are based on UTC, so the timezone value can be used to convert a timestamp to a local datetime. The timezone of a business location might change during the year, e.g. a German location would change from "UTC+01:00" to "UTC+02:00" when daylight saving time is active.

countryCode
contains the ISO 3166-1 alpha-3 country code of the address.

city contains the city name of the address.

street
contains the street name and house/unit number of the address.

postalCode
contains the postal code of the address.

2.2.19.3 VATs contains information about all VAT groups that apply at the business location. All VAT groups are listed here even those that were not used in transactions during a period.

Figure 33: VAT JSON Structure

```
"vats": [{
    "id": 1,
    "percent": 19000,
    "description": "full",
    "name": "vat name"
}]
```

vats
contains an array of JSON objects. Each object represents one VAT group that applies at the business location.

The following fields are present for each entry of the array:

id contains the id of the VAT group. It is an internal id which was assigned by the POS-System. There are a couple of places in a GTC-Document where a VAT group might be referenced using a ***vatId***. If you need more information about a VAT group based on the ***vatId***, scan through the array and match the ***vatId*** with the ***id***.

percent
contains the tax percentage of the VAT group.

description
contains a descriptive text of the VAT group. This could be for instance "Full", "Reduce", "Not Taxable", "Not Taxed" and so on.

name
contains the display name of the VAT group. This field is used to display or print VAT information and amounts.

2.2.19.4 Devices contains information about all POS-Devices that were registered with the POS-System for the duration of the period at the business location. Only those devices are able to record transactions. All devices of the location are listed here even devices that were not used during the period.

The subsection ***devices*** contains additional subsections. One that describes the master device, another one that contains a list of slave devices and one that lists all secure elements that were used to create digital signatures. If a business location does not use slave devices, then the slaves section is still present and it's value is null. If the tax regulations do not require that transactions are signed, then the secureElements section is still present and it's value is null.

Figure 34: Devices JSON Structure

```
"devices": {
    "master": {
        "udid": "fa443896d75f4953a4fd282694d6fed2",
        "name": "Waiter John Doe iPad",
        "ipAddress": "127.0.0.1",
        "isCustomerDisplayFlag": false,
        "costCenterId": 8642,
        "outletId": 4711,
        "priceLevelId": 1,
        "hardware": {
            "brand": "Apple",
            "model": "iPad5,3",
            "version": "12.4"
        },
        "software": {
            "brand": "<fix>",
            "version": "2.32.1234"
        }
    },
    "slaves": [{
        "udid": "82848de3cc3f4af29adfdf4cdda7b462",
        "name": "Waiter Jane Doe iPad",
```

```
        "ipAddress": "127.0.0.1",
        "isCustomerDisplayFlag": false,
        "costCenterId": 8642,
        "outletId": 4711,
        "priceLevelId": 1,
        "hardware": {
            "brand": "Apple",
            "model": "iPad5,3",
            "version": "12.4"
        },
        "software": {
            "brand": "<fix>",
            "version": "2.32.1234"
        }
    }],
    "secureElements": [{
        "id": 1,
        "serial":  "brrEJ3hWpk3K17jwQszm1wprC8rE+OhaA1dHxRBnzR0=",
        "algorithm": "ecdsa-plain-SHA512",
        "dateTimeFormat": "generalizedTime",
        "publicKey": "MIIEpAIBAAKCAQEAt...QuUlfkl5hR7LgwF+3q8bHQ==",
        "certificate": "MIICYzCCAcygAwIB... fjdWAGy6Vf1nYi/rO+ryMO",
        "certificateExpirationDate": "2020-01-30T23:00:00Z",
        "wearLevel": "Poor"
    }]
}
```


master
contains a JSON objects with detailed information about the master device.

The following fields are present for the master device:

udid
contains a unique device id. It is an internal id which is generated and assigned to the device by the POS-System. There are a couple of place throughout a GTC-Document where a device is referenced using a ***udid***. If you need more information about a device based on the ***udid***, scan through the array and match the ***udid***.

name
contains the display name of the device.

ipAddress
contains the IP address used by the device.

isCustomerDisplayFlag
if set to true, indicates that the device is used as a customer display, otherwise it is used as a POS device to register sales.

costCenterId
contains the id of the cost center which is assigned to the device.

outletId
contains the id of the outlet which is assigned to the device.

priceLevelId
contains the id of the price level which is assigned to the device.

hardware brand
contains information about the hardware vendor. The value is set to the constant string "Apple".

hardware model
contains the official model identifier issued by Apple, e.g. "iPad8,1.

hardware version
contains the version of the iOS running on the hardware.

software brand
contains information about the software vendor.

software version
contains the version of the POS app running on the hardware.

slaves
contains an array of JSON objects. Each object represents one slave device with detailed information about the device. The field is null if no slave devices are configured.

An object of the slave device section contains the same fields like the master device section.

secureElements
contains an array of JSON objects. Each object contains information about one secure element which was used to create digital signatures during the period. The field is null if the tax regulations at the business location do not require that transaction are signed.

The following fields are present for each of the secure elements:

id contains the id of the secure element. It is an internal id which was assigned by the POS-System.

serial
contains the serial id of the secure element. The field is filled with the "TSE Serial" for Germany and with the serial of the certificate for Austria.

algorithm
contains the algorithm that is used to create a signature. The field is filled with the "TSE Algorithm Identifier" for Germany and with the fixed string "AT2" for Austria.

dateTimeFormat
contains a value of the enum as defined in DSFinV-K 2.0 Section 7 (TSE ZEITFORMAT). The field is filled with the proper value defined by the TSE for Germany and with the fixed string "generalizedTime" for Austria.

publicKey
contains the public key of the secure element.

certificate
contains the certificate of the secure element.

certificateExpirationDate
contains the timestamp of the expiration date of the certificate that is used by the secure element.

wearLevel
contains the wear level of the secure element, if it uses components like NAND and so on that are subject to wear. It is indicated using the following grades:

- "Good" means that the secure element works normally.
- "Poor" means that the secure element has reached the point that a replacement should be prepared.
- "Broken" means that the secure element does not work reliably anymore.

2.2.19.5 Waiters contains information about all users/waiters that are configured with the POS-System at the business location for the period. Only those users/waiters are able to create transactions.

Figure 35: Waiters JSON Structure

```
"waiters": [{
    "id": 1,
    "firstName": "John",
    "lastName": "Doe",
    "personnelNumber": "1234567890",
    "isSafeUserFlag": false,
    "isActiveFlag": true
}]
```

waiters
contains an array of JSON objects. Each object represents a configured user/waiter at the business location. All configured users/waiters are listed here even those who did not work during the period. A waiter is included here if it is an active waiter or an inactive waiter with a current balance > 0 for one of the configured tenders.

The following fields are present for each entry of the array:

id contains the id of the user/waiter. It is an internal id which was assigned by the POS-System. There are a

couple of places in a GTC-Document where a user/waiter might be referenced using a ***waiterId***. If you need more information about a user/waiter based on the ***waiterId***, scan through the array and match the ***waiterId*** with the ***id***.

firstName
contains the first name of the user/waiter.

lastName
contains the last name of the user/waiter.

personnelNumber
contains the personnel number of the user/waiter.

isSafeUserFlag
indicates if the user/waiter represents the store safe.

isActiveFlag
indicates if the user/waiter is active at the location. The flag is only false if the waiter is not active anymore or was deleted on cloud, but still has a current balance > 0 for one of the configured tenders. As long as this is the case, the user/waiter must show up in the list.

2.2.19.6 Business Cases contains summary information about the performed business cases by each waiter in the period. The data is relevant for the Z-Report and much of the data is printed on the Z-Report. The data is also relevant for export files used by tax offices to run audits in some countries, e.g. Germany. The data might also be used to feed an accounting system.

Figure 36: Business Cases JSON Structure

```
"businessCases": {
    "<waiterid>": [
            {
              "typeCode": "00",
              "isSummaryFlag": true,
              "taxAmounts": [{
                  "taxSalesGrossAmount": 501190,
                  "taxSalesNetAmount": 468402,
                  "taxAmount": 32788,
                  "taxPercent": 19000,
                  "count": 100
              }],
              "details": {
                  "subTypeCode": "00",
                  "reasonId": 1,
                  "reasonClassId": 1,
                  "reasonName": "Bank Deposit",
                  "reasonCategoryCode": "00",
                  "tenderId": 1,
                  "typeCode": "00",
                  "tenderName": "Cash",
                  "currencyCode": "EUR",
                  "text": "Tainted"
              }
          },
          { .... },
          { .... }
    ],
    "total": [ .... ]
}
```

businessCases
contains a mapping that assigns an array of JSON objects to every configured waiter. Each object of the array holds totals over all transactions of a fundamental business case that were performed by the waiter in the period. In the above example, "<waiterId>" represents the id of one of the configured waiters (e.g. "4711"). The mapping also contains a special key "total". An array object of "total" contains totals of a fundamental business case over all waiters. The mapping contains an array for all configured waiters even if they did not work during the period. A configured waiter is an active waiter or an inactive waiter with a balance > 0 for one of the configured tenders.

A business case is identified by a ***typeCode***. A table with the supported business cases is provided with the description of the field ***typeCode***. But why is it then called fundamental business case in the previous

paragraph? It's hard to give an argument for this at this point since it actually requires that the whole sections was already read. But the definition is given in the next paragraph in the hope that it will get clearer while reading the rest of the section.

A fundamental business case is given by a unique combination of the following four properties ***waiterId***, ***typeCode***, ***isSummaryFlag*** and ***details***. When a transaction is evaluated, the values of the four properties are determined. A unique set of the four properties constitutes a fundamental business case. All transactions that yield the same set of values for the properties are summarized under that fundamental business case. That all sounds a bit theoretical. But it is easy to understand it intuitively, it just means that for instance every single discount type or every single disbursement reason and so on is seen as a single business case. This gets even clearer if you take a look at a Z-Report, since these values are printed there and the ***businessCases*** section is supposed to hold all data that is contained in the Z-Report. Therefore, a printed value of a Z-Report represent a fundamental business case. It is also intuitively clear that for certain business cases there is also an aggregating business case which is indicated by the ***isSummaryFlag***. That essentially means that there is also a fundamental business case which for instance aggregates all single fundamental discount business cases. It is important to get the intuition and then it is quite easy to derive the fundamental business cases from the description of the fields ***typeCode***, ***isSummaryFlag*** and ***details*** given below.

The array that is stored in the mapping for a waiter has an entry for all fundamental business cases even if one was not used (it's values are 0 then).

Essentially, the JSON structure represents a two-dimensional matrix. The first dimension is the list of all configured waiters, the second dimension is the list of all fundamental business cases and every cell contains the summary for the waiter-case combination.

The following fields are present for each entry of the array:

typeCode

indicates which business case the object represents. The following table shows the available codes.

Table 21: Business Cases Type Codes

Type Code	Business Case
'00'	Entered Items - Every entered item is included here, including corrections and voids.
'01'	Corrections - Changes to line items of an order before the order button is pressed.
'02'	Line Voids - Line items that are voided after the order was done.
'03'	Transactions Voids - Voiding a whole transaction.
'04'	Discounts - All applied discounts on the line level and transaction level.
'05'	Write-Offs - All write-offs done during the period.
'06'	Expenses - Everything a waiter spends on behalf of the customer. It will be reimbursed upon payment.
'07'	Taxed Voucher Sales - Issuance / Sales of taxed vouchers (e.g. EU Single-Purpose-Voucher).
'08'	Taxed Voucher Redemption - Redemption of taxed vouchers (e.g. EU Single-Purpose-Voucher).
'09'	Untaxed Voucher Sales - Issuance / Sales of untaxed vouchers (e.g. EU Multi-Purpose-Voucher).
'10'	Untaxed Voucher Redemption - Redemption of untaxed vouchers (e.g. EU Multi-Purpose-Voucher).
'11'	Turnover - Turnover in the period (under VAT regulations this is the taxable amount).
'12'	Funds Receipts - Receipt of tenders into the POS-System not related to sales.
'13'	Disbursements - Disbursement of tenders from the POS-System not related to sales.
'14'	Tender Transfer - The transfer of a certain amount from one tender to another tender.
'15'	Tender Loan - Movement of tender from safe to waiter so that change is available to serve customers.
'16'	Tender Pickup - Movement of tender from waiter to safe so that losses in case of theft are reduced.
'17'	On-Account - All payments on account (aka. Invoiced / auf Rechnung / tenderTypeCode == '05').
..	See comment about new enumeration values in section 1.5.

isSummaryFlag

indicates if an entry contains summary or detailed information about a business case. What is the difference? Some business cases can only be represented as a summary, for instance Line Voids. Other business cases can only be represented in a detailed way, for instance Disbursements since they depend on a reason or for instance Pickups since they depend on waiters plus tenders. And there are business cases that have a summary as well as a detailed representation, for instance Discounts since every single discount is seen as a business case on its own but there is also an overall Discount business case.

If the ***isSummaryFlag*** is set, then the ***details*** section is empty and has the value null, otherwise the ***details***

sections contains proper values. The following tables gives an overview about the different business cases and when detailed information is provided.

Table 22: Summary vs Detailed Business Case Information

Type Code	isSummaryFlag	Description
'00'	true	Summary of all entered items including corrections and voids.
'01'	true	Summary of all corrections.
'02'	true	Summary of all line voids.
'03'	true	Summary of all transactions voids.
'04'	true	Summary of all applied discounts.
'04'	false	Detailed summary of one particular discount type that was applied.
'05'	true	Summary of all write-offs.
'05'	false	Detailed summary of one particular write-off type that was applied.
'06'	true	Summary of all expenses.
'07'	true	Summary of all taxed voucher sales.
'08'	true	Summary of all taxed voucher redemption.
'09'	true	Summary of all untaxed voucher sales.
'10'	true	Summary of all untaxed voucher redemption.
'11'	true	Summary of the turnover in the period.
'12'	false	Detailed summary of one particular Funds Receipts cause.
'13'	false	Detailed summary of one particular Disbursements cause.
'14'	false	Detailed summary of one particular Tender Transfer cause.
'15'	false	Detailed summary of one particular tender loaned to the waiter.
'16'	false	Detailed summary of one particular tender picked up from the waiter.
'17'	true	Summary of all payments on account.
..		See comment about new enumeration values in section 1.5.

taxAmounts

contains an array of JSON objects. Each object represents a due tax amount of a VAT group that was triggered by the fundamental business case in the period. A fundamental business case might trigger several VAT groups, e.g. sales could be subject to a reduced tax or full tax depending on the usage (to-go or in-house) or for other reasons, therefore ***taxAmounts*** needs to be an array instead of just one object. If a fundamental business cases is not taxed, then the array contains exactly one object. The field is always set no matter if it is a detailed or summary fundamental business cases.

An object of the array has the following fields:

taxSalesGrossAmount

contains the cumulative sum of all amounts of the fundamental business case affected by the VAT group. The value is calculated based on the proper values of all transactions of the fundamental business case.

taxSalesNetAmount

contains the net amount calculated for the VAT group. The value is calculated based on the proper values of all transactions of the fundamental business case. If a fundamental business cases is not taxed, then the field equals ***taxSalesGrossAmount***.

taxAmount

contains the total tax amount calculated for the VAT group. The value is calculated based on the proper values of all transactions of the fundamental business case. If a fundamental business cases is not taxed, then the field contains 0.

taxPercent

contains the tax percentage of the VAT group. If a fundamental business cases is not taxed, then the field contains 0.

count

contains the number of times the fundamental business case was performed.

details

contains a JSON object that gives detailed information about a fundamental business case. The field is only set if ***isSummaryFlag*** is false, otherwise it has the value null. It was mentioned in the description of the ***businessCases*** section above that a fundamental business case is identified by the four properties ***waiterId***,

typeCode, ***isSummaryFlag*** and ***details***. But ***details*** is a compound object and not a single value, so how are they compared. Two ***details*** object are equal, if the fields ***subTypeCode***, ***reasonId***, ***reasonClassId*** and ***tenderId*** are equal. Any of those four fields might be null depending on the fundamental business case. The remaining fields can be seen as additional commentary fields.

It should be noted that the section also contains a ***tenderId***. A tender can be in a different currency. For that reason, the section does include a ***currencyCode***. If the tender is in a foreign currency, then the amounts that are contained in the ***taxAmounts*** section are also in foreign currency. This can actually only happen for the fundamental business cases Funds Receipt, Disbursement, Tender Transfer and Tender Pickup. These business cases are always detailed anyway. For all other business case like for instance Turnover, all values in ***taxAmounts*** are always in base currency.

subTypeCode

indicates the fundamental sub-business case that the object represents. The field is only used for the fundamental business cases Funds Receipt (typeCode '12'), Disbursements (typeCode '13') and Tender Transfer (typeCode '14'). The field is set to the value null in all other cases. The following table shows the available codes for the affected business cases.

Table 23: Business Cases Sub Type Codes

Type Code	Business Case	Sub Business Case
'00'	Funds Receipt	Miscellaneous
'01'	Funds Receipt	Private Investment
'02'	Funds Receipt	Safe Deposit
'03'	Funds Receipt	Balance Plus
'04' - '19'	≪ RESERVED ≫	≪ RESERVED ≫
'20'	Disbursement	Miscellaneous
'21'	Disbursement	Bank Deposit
'22'	Disbursement	Wage Payment
'23'	Disbursement	Private Withdrawal
'24'	Disbursement	Gratuity Payment
'25'	Disbursement	Balance Minus
'26' - '39'	≪ RESERVED ≫	≪ RESERVED ≫
'40'	Tender Transfer	Transfer In - Nonspecific
'41'	Tender Transfer	Transfer Out - Nonspecific
'42'	Tender Transfer	Transfer In - Payment Adjustment
'43'	Tender Transfer	Transfer Out - Payment Adjustment
'44'	Tender Transfer	Transfer In - Tip Adjustment
'45'	Tender Transfer	Transfer Out - Tip Adjustment
'46'	Tender Transfer	Transfer In - Tip Addition
'47' - '59'	≪ RESERVED ≫	≪ RESERVED ≫
..		See comment about new enumeration values in section 1.5.

reasonId

contains the id of the reason which specifies why the fundamental business case happened. A reason always belongs to a class. The class is given by the ***reasonClassId***. A reason is not mandatory. If no reason is provided, then the field is empty.

reasonClassId

contains the id of the reason class where the reason is grouped in. If a reason is given, this field indicates which class the reason belongs to, otherwise it is empty. Classes are used to categorize reasons to better structure them. Reasons are only used in the context of ***Funds Receipt Transactions*** and ***Disbursement Transactions*** at the moment. The following table shows the possible values.

Table 24: Class Ids

Id	Description
9	Defines a reason of a Funds Receipt Transaction
10	Defines a reason of a Disbursement Transaction

Continued on next page

Continued from previous page

Id	Description
..	See comment about new enumeration values in section 1.5.

reasonName
contains the name (description) of the reason. If no reason is given, the field is empty.

reasonCategoryCode
contains the category code of a connected reason. The code is only used in the context of ***Funds Receipt Transactions*** and ***Disbursement Transactions*** at the moment. The value is null, if a reason has no assigned category code. The following table lists the possible values:

Table 25: Reason Class Ids & Category Codes

Class Id	Category Code	Description
9	'00'	Safe Deposit
9	'01'	Private Investment
9	'02'	Balance Plus
10	'00'	Bank Deposit
10	'01'	Gratuity Payment
10	'02'	Wage Payment
10	'03'	Private Withdrawal
10	'04'	Balance Minus
..		See comment about new enumeration values in section 1.5.

tenderId
contains the id of the tender that is connected with the fundamental business case. If no tender is involved, the field is empty.

typeCode
identifies the type of tender. A table with the possible codes is available here where the tender type code of the tender line item is documented.

tenderName
contains the name of the tender, if ***tenderId*** is set, otherwise the field is empty.

currencyCode
contains the ISO 4217 code of the currency e.g. USD or CHF. The field is only set, if the ***tenderId*** is set, otherwise it is empty.

text
contains a free text. The field might be set for certain fundamental business cases. The POS sets the value, if an identifying name is given but it is not directly connected with an entity in a GTC-Document. One example is, the POS will put here the discount name, if a discount is given but it is not connected with a reason.

2.2.19.7 Payments contains the total received amount of each tender for all waiters. Tenders are received and sometimes spent while performing business cases.

As a side effect, the ***payment*** section also documents which tenders are configured. Therefore, this sections serves a similar purpose like the ***company***, ***location***, ***vat***, ***devices*** and ***waiters*** sections to record the fiscally relevant master data of a period.

Since tenders are also spent, the received amount could actually get negative in certain scenarios. For example, let's assume that the opening balance of the foreign currency Dollar USD is $1000 for a waiter. And, there were no transactions settled with USD during the period by the waiter, but $500 were picked up. Then the ***payments*** section would actually show -$500 Dollars. This is OK and is actually consistent with what is said with respect to the ***Closing Balance*** in the next paragraph that the closing balance results from the opening balance plus payments. Here the closing balance would be $500 = $1000 + (-$500) which is the correct calculation of the closing balance.

Figure 37: Payments JSON Structure

```
"payments": {
    "<waiterid>": [
        {
            "id": 1,
            "typeCode": "00",
            "currencyCode": "EUR",
            "name": "Cash",
            "salesAmount": 1234000,
            "tipAmount": 10000,
            "untaxedVoucherAmount": 50000,
            "fundsReceiptAmount": 25000,
            "loanAmount": 50000,
            "transferInAmount": 25000,
            "disbursementAmount": 17000,
            "expenseAmount": 5000,
            "pickupAmount": 100000,
            "transferOutAmount": 25000,
            "amount": 1247000,
            "baseCurrencyAmount": 1247000
        }, {
            "id": 2,
            "typeCode": "01",
            "currencyCode": "USD",
            "name": "Dollar",
            "salesAmount": 234000,
            "tipAmount": 3400,
            "untaxedVoucherAmount": 0,
            "fundsReceiptAmount": 0,
            "loanAmount": 0,
            "transferInAmount": 0,
            "disbursementAmount": 0,
            "expenseAmount": 0,
            "pickupAmount": 100000,
            "transferOutAmount": 0,
            "amount": 137400,
            "baseCurrencyAmount": 117870
        }, {
            "id": 3,
            "typeCode": "02",
            "currencyCode": "EUR",
            "name": "Visa",
            "salesAmount": 765100,
            "tipAmount": 15000,
            "untaxedVoucherAmount": 100000,
            "fundsReceiptAmount": 0,
            "loanAmount": 0,
            "transferInAmount": 0,
            "disbursementAmount": 0,
            "expenseAmount": 0,
            "pickupAmount": 0,
            "transferOutAmount": 0,
            "amount": 880100,
            "baseCurrencyAmount": 880100
        },
        { .... }
    ],
    "total": [ .... ]
}
```


payments

contains a mapping that assigns an array of JSON objects to every configured waiter. Each object of the array represents the total received amount of a tender for the waiter. In the above example, "<waiterId>" represents the id of one of the configured waiters (e.g. "4711"). The mapping also contains a special key "total". An array object of "total" contains the sum of a tender over all waiters. The mapping contains an array for all configured waiters even if they did not work during the period. A configured waiter is an active waiter or an inactive waiter with a balance > 0 for one of the configured tenders. All configured tenders are always in the list. The list of tenders can not change during a period.

Essentially, the JSON structure represents a two-dimensional matrix. The first dimension is the list of all configured waiters, the second dimension is the list of all configured tenders and every cell contains the total received amount for the waiter-tender combination.

The following fields are present for each entry of the array:

id contains the id of the tender. It is an internal id which was assigned by the POS-System. There are a couple of places in a GTC-Document where a tender might be referenced using a ***tenderId***. If you need more information about a tender based on the ***tenderId***, scan through the array of the "total" key and match the ***tenderId*** with the ***id***.

typeCode
identifies the type of tender. A table with the possible codes is available here where the tender type code of the tender line item is documented.

currencyCode
contains the ISO 4217 code of the used currency e.g. USD or CHF.

name
contains the display name of the tender.

salesAmount
contains the amount of the tender that was applied towards the sales transactions. The value is adjusted and takes voids into account. The amount does not include "non-turnover" items, it just includes genuine sales.

tipAmount
contains the amount of the tender that was received as tip. The value is adjusted and takes voids into account.

untaxedVoucherAmount
contains the amount of the tender that was received as payments for issued untaxed vouchers (see here, taxed vs. untaxed vouchers, taxed vouchers are included in salesAmount). The value is adjusted and takes voids into account.

fundsReceiptAmount
contains the amount of the tender that was received with funds receipt operations. Funds receipts could be many thing e.g. Personal Investment or Safe Deposits and so on. The details about the different business cases is contained in the ***Business Cases*** section. This field contains the overall total. Usually, this applies to the base currency only.

loanAmount
contains the amount of the tender that was loaned from the store safe. Loans in the context of a POS-System are used to fill a drawer/wallet with some initial coins and bills so that the user/waiter can give change to a customer. Usually, this applies to the base currency only.

transferInAmount
contains the amount of the tender that was transferred in using a ***Tender Transfer Transaction***.

disbursementAmount
contains the amount of the tender that was spent on disbursements. Disbursements could be many thing e.g. Wage Advance or Bank Transfer or Business Expenses and so on. The details about the different business cases is contained in the ***Business Cases*** section. This field contains the overall total. Usually, this applies to the base currency only.

expenseAmount
contains the amount of the tender that the user/waiter has spent on behalf of the customer to buy things like flowers, cigarettes and so on. The customer will reimburse the user/waiter upon payment. The field is only used for the base currency since the POS-System assumes that these out-of-pocket expenses are done with cash.

pickupAmount
contains the amount of the tender that was picked up from the drawer/wallet of the user/waiter and moved to the store safe.

transferOutAmount
contains the amount of the tender that was transferred out using a ***Tender Transfer Transaction***.

amount
contains the total amount of the tender that was received by the user/waiter in the respective currency. The amount is calculated using the following formula and might be negative as explained above:

```
amount = salesAmount
         + tipAmount + untaxedVoucherAmount
         + fundsReceiptAmount + loanAmount + transferInAmount
         - disbursementAmount - expenseAmount - pickupAmount - transferOutAmount
```

baseCurrencyAmount
contains the ***amount*** converted to base currency. If the tender is in base currency already, then ***amount*** equals ***baseCurrencyAmount***. If the tender is in a foreign currency, then the value is not simply calculated by using the current exchange rate. The rate might change during a period, that would be the wrong way to calculate it. Therefore, the amount is the sum of all amounts of all tender line items where the ***tenderId*** matches ***id*** and also the ***waiterIds*** match (in other word, payments with the tender accepted by the waiter).

2.2.19.8 Closing Balance contains information about the closing balance of each tender for all waiters. The closing balance results from the opening balance plus payments. The section ***closingBalances*** has the same fundamental structure as the section ***openingBalances*** as described here.

Figure 38: Closing Balance JSON Structure

```
"closingBalances": {
    "<waiterId>": [
        {
            "id": 1,
            "typeCode": "00",
            "currencyCode": "EUR",
            "name": "Cash",
            "amount": 123400,
            "baseCurrencyAmount": 123400
        }, {
            "id": 2,
            "typeCode": "01",
            "currencyCode": "USD",
            "name": "Dollar",
            "amount": 250000,
            "baseCurrencyAmount": 230000
        },
        { .... }
    ],
    "total": [ .... ]
}
```

closingBalances
contains a mapping that assigns an array of JSON objects to every configured waiter. Each object of the array represents the closing balance of a tender for a waiter. In the above example, "<waiterId>" represents the id of one of the configured waiters (e.g. "4711"). The mapping also contains a special key "total". An array object of "total" contains the sum of a tender over all waiters. The mapping contains an array for all configured waiters even if they did not work during the period. A configured waiter is an active waiter or an inactive waiter with a balance > 0 for one of the configured tenders. All configured tenders are always in the list.

Essentially, the JSON structure represents a two-dimensional matrix. The first dimension is the list of configured waiters, the second dimension is the list of configure tenders and every cell contains the closing balance for the waiter-tender combination.

The following fields are present for each entry of the array:

id contains the id of the tender. It is an internal id which was assigned by the POS-System.

typeCode
identifies the type of tender. A table with the possible codes is available here where the tender type code of the tender line item is documented.

currencyCode
contains the ISO 4217 code of the currency e.g. USD or CHF.

name
contains the display name of the tender.

amount
contains the closing balance of a tender in the respective currency. The value is calculated by adding the opening ***amount*** of the ***Period Open Line Item*** and the ***amount*** of the ***payments*** section.

baseCurrencyAmount
contains the closing balance in base currency. If the tender actually is in base currency, then ***baseCurrencyAmount*** equals ***amount***. If the tender represents a foreign currency, then the current exchange rate is used to convert ***amount*** to base currency.

2.2.19.9 Sales Summaries contain the sales summary information for all items, groups and divisions that show up in the transactions. Therefore in the context of a GTC Day-Document, the sales summaries contain the complete sales information of a period. Items, groups and divisions are levels of the merchandise hierarchy and are used for reporting. Sales summaries are shown for selected levels of the merchandise hierarchy only. A typical merchandise hierarchy has the following structure:

1. Division
2. Department
3. Sub-Department
4. Class
5. Sub-Class
6. (Product) Group
7. Item (Article)

The POS-System also uses the name "Supergroup" for division. In the context of a GTC-Document the more general (common) name ***Division*** is used to name any of the top-most (root) merchandise levels of a hierarchy.

A GTC-Document contains only sales summaries for important levels of the merchandise hierarchy and not for all levels. Which levels are important? A GTC-Document contains the sales summaries for the levels that a typical consumer wants to use. The following short table shows what levels certain consumers of a GTC-Document use.

Table 26: Consumers and Sales Summaries on Levels

Typical Consumer	Levels	Often used to
ERP-System	1) 6) and 7)	Reporting, Replenishment, Procurement
Accounting-System	1)	Compare Total Sales with Payments
PMS-Systems	Mostly 1) but sometimes 6)	Compare Total Sales with Payments

Since the levels 1), 6) and 7) are mainly used, they are considered to be the most important levels. Therefore only those levels are included in a GTC-Document. The levels are represented by the sections ***divisions - 1)***, ***groups - 6)*** and ***items - 7)***. These sections share a common structure, therefore they are documented together. An ***Id*** is followed by additional fields which contain the sales data and a tax section. The structure is shown for the division level in the following JSON, but it contains for all other levels as well.

Some general information:

- The value of a field on the Division level (e.g. ***salesAmount***) should be equal to the sum of the same field over all groups on the Group Level.
- The value of a field on the Group level (e.g. ***salesAmount***) should be equal to the sum of the same field over all items which belong to that group.
- The value of a field on the Division level (e.g. ***salesAmount***) should be equal to the sum of the same field over all items which belong to that division.
- The actual sale values can be calculated for each field starting with ***sales*** as follows:

  ```
  actual... = sales... - void...
  ```

- Discounts are already included in the sales values. In order to see how much discounts were given in total the ***discount*** values are provide. The ***discount*** values are already adjusted actual values (taking voids into account). So, to calculate the original undiscounted actual values you would do the following calculation:

  ```
  undiscounted_actual = actual + discount
  ```

- The ***tax*** section explains how the values of ***salesAmount***, ***salesTaxAmount***, ***voidAmount*** and ***voidTaxAmount*** are composed based on the involved vats. The following statements always hold (Python Code):

  ```
  salesAmount    = sum([t.salesAmount for t in divisons['tax']])
  salesTaxAmount = sum([t.salesTaxAmount for t in divisons['tax']])
  voidAmount     = sum([t.voidAmount for t in divisons['tax']])
  voidTaxAmount  = sum([t.voidTaxAmount for t in divisons['tax']])
  ```

Figure 39: Top Level - Sales Summaries JSON Structure

```
"salesSummaries": {
    "divisions": [{
        "divisionId": 10,
        "salesAmount": 1999990,
        "salesTaxAmount": 225370,
        "salesUnitCount": 100000,
        "salesTransactionCount": 24000,
        "salesGuestCount": 49000,
        "voidAmount": 2990,
        "voidTaxAmount": 480,
        "voidUnitCount": 1000,
        "voidTransactionCount": 1000,
        "discountAmount": 39990,
        "discountUnitCount": 3000,
        "discountTransactionCount": 2000,
        "name": "Food",
        "tax": [{
            "vatId": 1,
            "taxPercent": 19000
            "salesAmount": 1002990,
            "salesTaxAmount": 160140,
            "voidAmount": 2990,
            "voidTaxAmount": 480
        },{
            "vatId": 2,
            "taxPercent": 7000
            "salesAmount": 997000,
            "salesTaxAmount": 65220,
            "voidAmount": 0,
            "voidTaxAmount": 0
        }]
    }],
    "groups": [{
        "groupId": 10,
        " ..... ": "same structure as division"
    }],
    "items": [{
        "sku": 10,
        " ..... ": "same structure as division"
    }]
},
```

salesAmount
contains the monetary value of all sold items. An items is counted as sold even if it is later voided (so voided items are also included here).

salesTaxAmount
contains the total tax amount calculated for all sold items.

salesUnitCount
contains the number of all sold items.

salesTransactionCount
contains the number of transactions featuring the sale of an item.

salesGuestCount
contains the number of guests recorded for transactions featuring a sale.

voidAmount
contains the total monetary value of all voided items.

voidTaxAmount
contains the total tax amount calculated for all voided items.

voidUnitCount
contains the number of all voided items.

voidTransactionCount
contains the number of transactions where an item was voided.

discountAmount
contains the monetary value of all applied discounts (on the line and also transaction discounts).

discountUnitCount
contains the number of items that had a discount (on the line, also transaction discounts) applied.

discountTransactionCount
contains the number of transactions featuring the application of a discount to an item.

name
contains the short name of the division, group or item.

tax contains the detailed tax information for each involved VAT group. The section contains the additional fields ***vatId***, ***taxPercent*** plus the already defined fields ***salesAmount***, ***salesTaxAmount***, ***voidAmount*** and ***voidTaxAmount***. The fields contain sale and void values connected with the VAT group. Please refer to the general notes above to see the relations of the fields.

vatId
contains the id of the involved VAT group.

taxPercent
contains the tax percentage of the involved VAT group.