



MD 29/04/2024  
REGISTERED CASH REGISTER  
SYSTEM – TECHNICAL  
SPECIFICATIONS -

DETAILED DESCRIPTION OF THE  
OPERATION AND COMMUNICATION  
BETWEEN THE CASH REGISTER  
SYSTEM AND THE FISCAL DATA  
MODULE

VERSION 1.1 – 18/07/2024

## CHANGELOG

Version	Date	Summary
1.0	26/06/2024	Original text
1.1	18/07/2024	<b><u>Error corrections:</u></b> <ul style="list-style-type: none"><li>- Input SaleInput: fdmRefs (Ch. 2, 2.3.4)</li><li>- lineTotal: remark composite product – formulae to be send to FDM (Ch. 2, 2.2.2)</li><li>- type MessageItem: message (Ch. 2, 2.3.2)</li><li>- input ReportUserXInput: reportBookingDate twice mentioned (Ch. 2, 2.3.4)</li><li>- input InvoiceInput: insert costCenter (Ch. 2, 2.3.4)</li><li>- whitelist replaced by allowlist (Ch. 2, 2.2.4 and 2.3.3.)</li></ul> <b><u>Clarifications</u></b> <ul style="list-style-type: none"><li>- physical connections POS – FDM (Ch 2, 1.)</li><li>- Object fdmRefs – (Ch. 2, 2.2.2)</li><li>- lineTotal – (Ch. 2, 2.2.2)</li></ul>

INTRODUCTION .....	1
CHAPTER 1 – TECHNICAL REQUIREMENTS FOR THE CASH REGISTER SYSTEM ...	2
1. REGISTRATION OF EVENTS .....	2
1.1. DIRECT SALE .....	2
1.2. SALES REGISTRATION WITH A complete REfund OF PREVIOUSLY ISSUED VAT RECEIPT .....	2
1.3. INTERRUPTED SALE .....	2
1.4. THE ACCOUNT OVERVIEW (PROVISIONAL BILL) .....	3
1.5. REGISTRATIONS IN TRAINING MODE.....	3
1.6. REGISTRATIONS OF PURE FINANCIAL TRANSACTIONS .....	3
1.7. CREATION OF INVOICES BASED ON PREVIOUSLY CREATED EVENTS N ...	3
1.8. COPIES OF PREVIOUSLY CREATED EVENTS.....	4
1.9. REGISTRATIONS OF EMPLOYEE PRESENCE .....	4
2. NUMBERING OF EVENTS .....	4
3. CONDITIONS FOR EVENT REGISTRATION .....	4
4. DATA RETENTION .....	6
5. VAT-CODES .....	6
6. THE VAT RECEIPT.....	7
6.1. IDENTIFICATION OF THE CASH REGISTER SYSTEM .....	7
6.2. IDENTIFICATION OF THE TERMINAL.....	7
6.3. IDENTIFICATION OF THE INPUT DEVICE.....	7
6.4. IDENTIFICATION OF THE USER.....	8
6.5. QR-CODE.....	8
6.6. GENERAL.....	8
7. ROUNDING OF PAYMENTS .....	8
7.1. THE OPERATOR ROUNDS CASH PAYMENTS ONLY.....	9
7.2. THE OPERATOR ROUNDS ALL PAYMENT METHODS .....	9
8. CONSOLIDATION AND PRINTING RULES.....	10
8.1. PRINTING.....	10
8.2. CONSOLIDATION .....	10
CHAPTER 2 – INTERACTION BETWEEN POS - FDM .....	12
DEFINITIONS AND ABBREVIATIONS.....	12
1. PHYSICAL CONNECTION.....	14
2. COMMUNICATION BETWEEN POS AND FDM .....	15
2.1. GENERAL.....	15
2.2. CONTENT OF THE COMMUNICATION BETWEEN POS AND FDM.....	17
2.2.1. POS → FDM – complete overview .....	17

2.2.2. POS → FDM – detailed description.....	18
Language.....	18
Delivery Method.....	18
POS Identification Details .....	18
POS Date and Time.....	19
Company and User Identification .....	19
References .....	20
Transactions Registration – Use of Management Options .....	20
Transactions Registration – Transaction Lines.....	22
Transactions Registration – Financial Movements .....	27
Reports.....	31
Communication Rules.....	36
2.2.3. FDM → POS – complete overview.....	37
2.2.4. FDM → POS – detailed overview.....	37
Message.....	40
Data.....	41
2.2.5. FDM → POS errorhandling .....	41
2.3. GRAPHQL SCHEMA COMMUNICATION BETWEEN POS AND FDM.....	43
2.3.1. Mutations.....	43
2.3.2. Types .....	44
2.3.3. Enums .....	45
2.3.4. Input Objects .....	48
2.4. DIGITAL SIGNATURE .....	56
2.4.1. Canonical JSON for the reproducible hash.....	56
2.4.2. The 'enriched JSON' whose content is digitally signed .....	56
2.4.3. Certificate to be used for the signature .....	57



## INTRODUCTION

The detailed descriptions provide in a more technical and detailed way the technical provisions of the ministerial decree of 29/04/2024 concerning the technical aspects and certification of the cash register system.

This is the detailed technical description regarding the operation of the cash register system and its communication with the Fiscal Data Module.

For the operation of the Fiscal Data Module, please refer to the relevant detailed description.

This document may undergo minor changes due to potential regulatory decisions or to correct any errors.

Additional information can be obtained from the competent service, NCI department RCRS via [secr.gksce@minfin.fed.be](mailto:secr.gksce@minfin.fed.be).

# CHAPTER 1 – TECHNICAL REQUIREMENTS FOR THE CASH REGISTER SYSTEM

Reference MD: Title II, Chapter 1

## 1. REGISTRATION OF EVENTS

Reference MD: Section 1, subsections 1, 7, and 8

### 1.1. DIRECT SALE

In this case, the event N is immediately registered.

The booking (and the GraphQL message) always contains the registered product lines in chronological order. Price changes are attached to those product lines. Corrections in the number of products are always separate lines, which take their place in the chronology.

The final printout of the VAT receipt can be consolidated, and the chronology can also be changed (for example, for grouping by department). The printout may also contain more information than legally required (for example: GTIN, unit price, ...).

The printout always includes the mention "VAT RECEIPT".

Examples in the Use Cases (UC).

### 1.2. SALES REGISTRATION WITH A COMPLETE REFUND OF PREVIOUSLY ISSUED VAT RECEIPT

If a previously created event N is COMPLETELY refunded, the **reference** of the original VAT receipt is mentioned in the GraphQL message and on the printout of the VAT receipt. The 'fdmRefs' array is used for this purpose. This new event N may contain additional product lines. A maximum of one previously created event N may be refunded per event N.

If the VAT receipt contains only a complete refund of a previous event N (and therefore contains only negative product lines), the printout of this VAT receipt will also include the additional mention 'REFUND' (Article 5, 3° of the MD).

Examples in the UC.

### 1.3. INTERRUPTED SALE

In this case, the sales registration is interrupted in time and one or more partial registrations (bookings) are made. Examples are table management, customer management, transfer to hotel account, acceptance of web orders, acceptance of kiosk orders, placing a registration on-hold.

For this, event P is used. The complete sales registration must always result in a final registration event N for the total of these events P (except in the case of transfer to hotel account).

A 'reference' is used to keep all bookings of a sales transaction together.

Each individual booking follows the same rules for the GraphQL message as for direct sales (corrections, printouts, ...).

Each printout of such a booking always includes the mention "PRO FORMA" and at the bottom also the mention "THIS IS NOT A VALID VAT RECEIPT".

Examples in the UC.

## 1.4. THE ACCOUNT OVERVIEW (PROVISIONAL BILL)

This event P can be created during an "interrupted sale" functionality, where an overview of the booked orders and/or the amount to be paid is displayed before the payment is registered, the "normal" event is created, and the VAT receipt is issued.

This also refers to calling up and displaying the status of a table on a tablet, smartphone, or any other device connected to or part of the cash register system.

In accordance with Article 34 of the MD, the product lines on both the printout and in the GraphQL message to the FDM may be consolidated for this account overview. Adding new product lines or making changes to existing ones is not allowed when creating the account statement.

A printout always includes the mention "PRO FORMA" and "PROVISIONAL BILL". At the bottom, it also includes the mention "THIS IS NOT A VALID VAT RECEIPT".

Examples in the UC.

## 1.5. REGISTRATIONS IN TRAINING MODE

All registrations made while the entire cash register system is in training mode or by a user who is in training mode are sent to the FDM as a training event. They always carry the label "T".

A printout of such an event always includes the designation "TRAINING". At the bottom, it also includes the mention "THIS IS NOT A VALID VAT RECEIPT".

Examples in the UC.

## 1.6. REGISTRATIONS OF PURE FINANCIAL TRANSACTIONS

Examples in the UC.

## 1.7. CREATION OF INVOICES BASED ON PREVIOUSLY CREATED EVENTS

If a cash register system has the capability to create an invoice, regardless of the format, based on one or more previously issued VAT receipts, this invoice is registered as an "INVOICE" event. The product lines of these VAT receipts are included in the printout of this event (but **not** in the GraphQL message to the FDM).

The payment lines of the original VAT receipts are **not** included.

The printout always includes the mention "INVOICE," and at the bottom, it also includes the mention "THIS IS NOT A VALID VAT RECEIPT".

Examples in the UC.

## 1.8. COPIES OF PREVIOUSLY CREATED EVENTS

If a **copy** of a previously created VAT receipt (event "NORMAL") needs to be printed (for example, after a printer malfunction or an unreadable ticket), a copy event must be created for this. A reference to the original VAT receipt is always provided.

This event can also be used for copies of previously created 'P,' 'I,' 'S,' 'F,' or 'R' events.

Any printouts always include the mention "COPY" and at the bottom it also includes the mention "THIS IS NOT A VALID VAT RECEIPT".

## 1.9. REGISTRATIONS OF EMPLOYEE PRESENCE

Every cash register system must include this functionality; actual use is not required unless the employer opts for certain social security or other benefits.

Via the "SOCIAL" event, the registration of the presence (start/end) of the "users" can be sent to the FDM. Any login routine can be used for this, provided the cash register system creates the correct data for the event and sends it to the FDM via the GraphQL message. This event is **not** printed and is **not** a sales transaction.

## 2. NUMBERING OF EVENTS

Article 13 requires continuous numbering for each event. This can mean one of the following:

- a separate continuous numbering per event type (label);
- a general continuous numbering across events;
- a general continuous numbering across events per terminal;
- a separate continuous numbering per event type (label) per terminal.

## 3. CONDITIONS FOR EVENT REGISTRATION

1) A user who wants to register events must first log in to the cash register system. No operation should be possible on the cash register system without a logged-in user. A separate login procedure can be provided for registering events of the type 'S' (Social).

Users of the cash register system, regardless of their role within the company, must be clearly identifiable by their INSZ number, which consists of either the national registry number or BIS number, depending on the case. This number is stored in the waiter software or in the "users" database of the cash register system. The INSZ number is composed of 11 numeric characters.

A user of the cash register system, external to the company (e.g., a technician), who records actions using the cash register system, is always identified in the cash register system with the number "00000000097".



For online and kiosk orders, and all future ordering possibilities that do not require the intervention of a physical person but fall under the full responsibility of the operation, the number "0000000029" should be used as the 'robot user'.

An overview of these program settings or database tables must be easily accessible to the controlling officer upon request.

2) The cash register system may only register deliveries of goods and/or services when the FDM is connected and fully operational. A transaction, whose registration has started, cannot be completed as long as a signature from the FDM cannot be received.

## 4. DATA RETENTION

Reference MD: Section 1, subsection 2

The taxpayer-user of a registered cash register system is responsible for the retention of the data created by the cash register system, in accordance with VAT legislation (and by extension, accounting legislation). The taxpayer-user is particularly responsible for the retention of the data on the cash register system and on the FDM. Backups of cash register system data remain part of the cash register system and are therefore subject to the same retention periods.

On the cash register system, the data created by the cash register system itself (database, JSON, the data of the GraphQL messages) and those received from the FDM (JSON) are retained in their original form, in accordance with the applicable fiscal retention period. It is reminded that, although no specific format is prescribed, all data created by the cash register system must be able to be presented in a readable and understandable form in accordance with Article 61, §1 of the VAT Code. To facilitate the copying of this data, at least one port (of a common type) of the cash register must be accessible/activated for an external data carrier.

The manufacturer of the cash register system must provide a detailed description of the database, and more specifically, all tables that contain event data or a complete description of any other method of storing the data mentioned in the ministerial decree, when applying for certification.

Manufacturers of cash register systems are explicitly reminded that they are responsible for the way the database data related to the aforementioned events is secured. The manufacturer must specify the security mechanisms used (whether built into the software itself or provided through third parties) when applying for certification.

## 5. VAT-CODES

Reference MD: Section 1, subsection 3

The cash register system must provide the following VAT codes:

VAT-CODE	VAT RATE DESCRIPTION	VAT-RATE
<b>A</b>	High	21 %
<b>B</b>	Medium	12 %
<b>C</b>	Low	6 %
<b>D</b>	Zero Rate	0 %
<b>X</b>	Outside VAT Scope	None

The cash register software may calculate the taxable amounts and the VAT due by itself. However, this result is **not** sent to the FDM. The firmware of the FDM calculates the taxable amounts and the VAT due itself, based on the transmitted product lines with the corresponding VAT code. The rates considered by the FDM are kept up-to-date by the online connection to the servers of the Federal Public Service Finance. Therefore, no manual action is needed on the FDM during a rate change.

The taxable amounts and VAT amounts received from the FDM are indicated on the VAT receipt. This also applies to the creation of reports.

**Use of VAT Code X:** for the registration of the 'sale' of goods and services that fall completely outside the scope of VAT. This includes the charge and return of empty goods, the charge and refund of deposits and the sale of a multipurpose voucher (MPV).

**Important Note:** items with dual VAT codes.

In exceptional cases, items may have two VAT codes. Typical examples are: a magazine sold together with a CD (non-hospitality) or an all-in menu (hospitality).

The data structures used allow such items to be correctly sent to the FDM as one product with different VAT codes; each VAT code is then provided with the part of the sales price (including VAT) to which the VAT code relates, supplemented by any price changes applicable to this part of the sales price.

These items can be printed as one product line on the VAT receipt, preferably with a combined VAT code (for example: AB). If this is not technically feasible, the VAT code of the component with the highest value may be printed, provided the actual distribution is sent to the FDM.

Of course, it is always possible to opt to print a separate product line per VAT code, with the corresponding value.

**Note:** If a menu is available with a separately displayed value of the drinks package, two separate product lines must always be opted for (both on the printout and in forwarding to the FDM).

## 6. THE VAT RECEIPT

Reference MD: Section 1, subsection 4

See also earlier in this document, registration of event N.

### 6.1. IDENTIFICATION OF THE CASH REGISTER SYSTEM

This refers to the assigned serial number from Article 46. This is displayed in the GraphQL message to the FDM in the `posId` field.

### 6.2. IDENTIFICATION OF THE TERMINAL.

If a RCRS setup consists of multiple cash registers under the same `posId`, then a unique `terminalId` must be used for each cash register. Examples are a logical name (bar, terrace, etc.) or any other identification (number, etc.). This is displayed in the GraphQL message to the FDM in the `terminalId` field.

### 6.3. IDENTIFICATION OF THE INPUT DEVICE.

An input device is the physical device on which the event is registered. This can be the stand-alone cash register, the terminal, the kiosk, the handheld, the smartphone, ....

For webshops located in the cloud or provided by a third party, the name of that party or the url of the webshop is used.

In the GraphQL message between the cash register and FDM, the identification is included in the `deviceId` field.

## 6.4. IDENTIFICATION OF THE USER

The user's INSZ is **NEVER** printed on the VAT receipt. However, the user must be linked internally in the cash register system via the identification mentioned on the receipt to this INSZ, which is also sent as part of the GraphQL message to the FDM.

## 6.5. QR-CODE

This is generated by the cash register software itself based on the URL provided by the FDM.

The URL consists of 38 alphanumeric characters, fitting into a 25x25 module type 2 QR with medium error correction feature (ECC).

This QR code is printed on the paper VAT receipt. For a digital print, the URL is provided as a link on the VAT receipt.

## 6.6. GENERAL

The VAT receipt may contain more information than legally required.

# 7. ROUNDING OF PAYMENTS

Reference MD: Section 1, subsection 9

The new articles VI.7/1 and VI.7/2 of the Code of Economic Law require the merchant to round the total payment amount in cash by the consumer to the nearest multiple of 5 cents.

Specifically, this means that payments ending in 1, 2, 6, or 7 cents are rounded down to the nearest multiple of 5 cents, while payments ending in 3, 4, 8, or 9 cents are rounded up to the nearest multiple of 5 cents. Payments of less than 5 cents are never rounded.

If the merchant wants to round all payments, regardless of the payment method, the RCRS must provide this option. Given the currently optional nature of rounding payments other than cash, the RCRS does not apply rounding for other payments unless the merchant opts for it.

It is noted that this choice is not made per transaction but applies to all non-cash payments. Rounding is mandatory for cash payments.

Note: The above **NEVER** applies to meal vouchers, eco vouchers, and gift vouchers, which always have a fixed value. When paying with meal vouchers, eco vouchers, and gift vouchers, they must always be processed first.

For the types of payment lines to be used, see Chapter 2 further in this document.

**Important note:** A VAT receipt can have a total rounding of a maximum of 2 euro cents (positive or negative).

If the RCRS is used in a hospitality environment, it is not uncommon for larger groups to make payments per person or per couple. Depending on the organization of the business and/or the cash register system used, this can be done via direct sales per "consumer", "table split" or "split payment".

In "direct sales", the consumption per consumer is registered exactly in the cash register and rounded or not, depending on the payment method (and the merchant's choice). A VAT receipt is issued per consumer.

In "table split", a VAT receipt is also issued per consumer (either with the exact consumption or with their proportional share). Payment roundings are identical to "direct sales".

However, in "split payment", only one VAT receipt is ultimately issued for all consumers together. In such a case, the operator must work methodically (this applies not only to hospitality and RCRS).

Considering the important note mentioned above, the following procedures should be followed:

## 7.1. THE OPERATOR ROUNDS CASH PAYMENTS ONLY

Methodology:

1. The payments with meal vouchers and gift vouchers (collectively or separately, as chosen) are registered. These appear as a payment line on the receipt. The cash register calculates the balance.
2. Then the cash payments are processed. These appear as one payment line on the receipt, **ROUNDED**. The cash register calculates the balance again.
3. Finally, electronic payments (**NOT ROUNDED**) are processed for the remaining balance. Multiple payment lines per type of payment can be used if desired.

The total rounding must not exceed 2 euro cents (positive or negative).

## 7.2. THE OPERATOR ROUNDS ALL PAYMENT METHODS

Methodology:

1. The payments with meal vouchers and gift vouchers (collectively or separately, as chosen) are registered. These appear as a payment line on the receipt. The cash register calculates the balance.
2. **THE CASH REGISTER ROUNDS THIS BALANCE.**
3. Then the cash payments are processed. These appear as one payment line on the receipt, **ROUNDED**. The cash register calculates the balance again.
4. Finally, electronic payments are processed for the remaining balance, which will inevitably be a **ROUNDED** amount. Multiple payment lines per type of payment can be used if desired.

The total rounding must not exceed 2 euro cents (positive or negative).

The Federal Public Service Finance provides examples of specific situations for interested POS developers on the website [www.geregistreerdkassasysteem.be](http://www.geregistreerdkassasysteem.be).

## 8. CONSOLIDATION AND PRINTING RULES

Reference MD: Section 1, subsections 7 and 8

### 8.1. PRINTING

For the application of this decree, a clear distinction must always be maintained between the following concepts: EVENT <> PRINT (ticket) <> MESSAGE to the FDM.

EVERY event ALWAYS results in a message to the FDM.

Only the events NORMAL (N) and INVOICE ALWAYS result in a printout, on paper, digitally, or both. The VAT receipt must be issued to the end customer in accordance with Article 21bis of KB No. 1. The same applies to the created invoice, which is issued pursuant to Article 53 §2 of the VAT Code.

The event REPORT (R) ALWAYS results in the creation of a file, BUT it must also be easily printable by the operator, either on paper or digitally.

All other events MAY (but do not have to) be printed. These printouts always contain, at the bottom and clearly legible, the statement "THIS IS NOT A VALID VAT RECEIPT".

An account overview additionally bears the clearly legible statement "PROVISIONAL BILL".

Moreover, printouts of events TRAINING (T) and PRO FORMA (P) contain, in addition to the statement "THIS IS NOT A VALID VAT RECEIPT", at the top and clearly legible, respectively, the statement "TRAINING TICKET" and "PRO FORMA TICKET". For the printouts of event COPY (C), the additional statement "COPY TICKET" is included.

Kitchen and bar tickets NEVER display amounts. If they are printed as a direct result of the creation of a PRO FORMA event, they must also contain the statement "THIS IS NOT A VALID VAT RECEIPT".

### 8.2. CONSOLIDATION

To optimize the readability of the tickets, consolidations are allowed in certain cases, including all corrections (negative or positive product lines) and price changes (decreases or increases, linked to the correct product line). The cash register software may not create a separate 'P' event for this purpose.

**Important note:** These consolidations are never applied to the messages the cash register system sends to the FDM. This means that a ticket content-wise corresponds to the message, but there can be formal differences.

Consolidation is not mandatory; each manufacturer determines whether their cash register software consolidates or not. If consolidation is used, the following rules must be followed.

#### **Direct Sale (event 'N'):**

The message to the FDM contains all entered data, in chronological order, including corrections (negative or positive product lines) and price changes (decreases or increases, linked to the correct product line).

Example:

1	Cola	2.50	A
1	Water	3.00	A
2	Spaghetti	20.00	B
-1	Cola	- 2.50	A
1	Water	3.00	A

In the printout on the VAT receipt, the corrections may be consolidated, and the order may be adjusted at the discretion of the cash register software (e.g., grouped by item departments or per VAT rate).

In the previous example, this would be:

2	Water	6.00	A
2	Spaghetti	20.00	B

**Interrupted Sale (table management):**

The same rules apply as described above. Concretely, the event P (which contains the **original** order) will be sent chronologically and including the corrections. The corresponding printout (provisional bill or kitchen/bar ticket) may be consolidated and regrouped.

At the closing event N, both the message and the printout of the final VAT receipt may be consolidated.

## CHAPTER 2 – INTERACTION BETWEEN POS - FDM

When the FDM is connected to the cash register system, it will:

- receive specific event data from the cash register system;
- update its relevant counters;
- generate control data;
- store these messages and responses;
- send the control data back to the cash register system;
- the cash register system will be able to print the ticket with the received control data, provided it receives such a response from the FDM

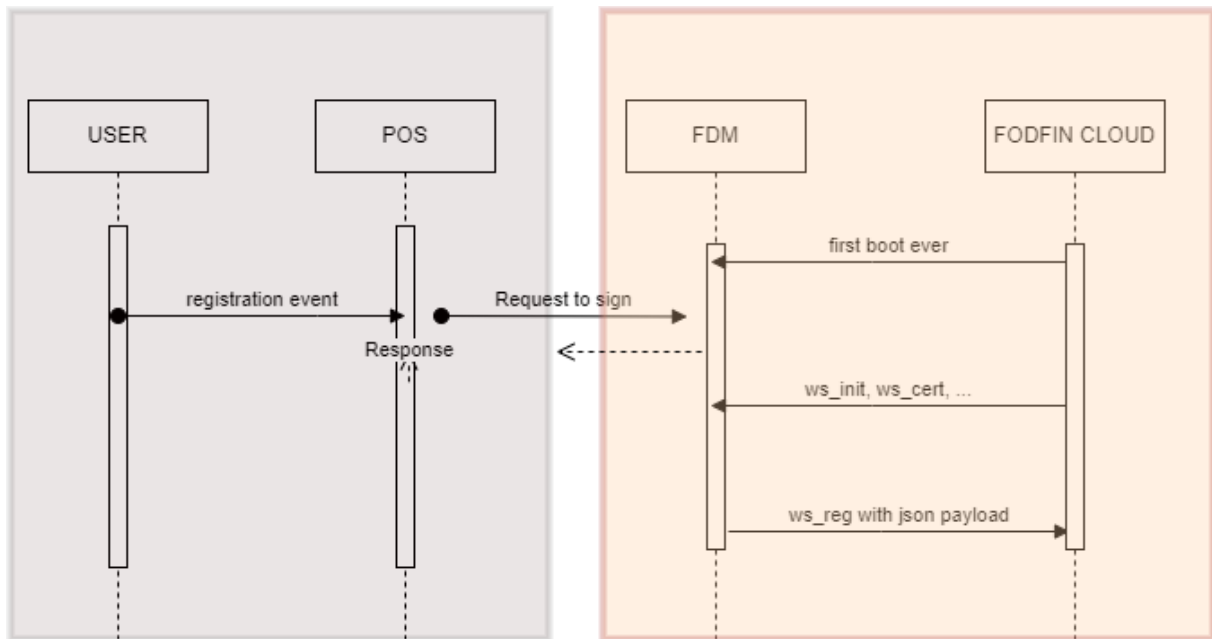
All registered cash register systems, as defined in the ministerial decree, must communicate this data using the communication protocol and according to the data formats described in this document as part of the aforementioned ministerial decree.

### DEFINITIONS AND ABBREVIATIONS

API	Application Programming Interface
EFT	Electronic Funds Transfer
ENUM	Enumeratie (list)
FDM	Fiscal Data Module
RCRS	Registered Cash Register System
GRAPHQL	Graph Query Language
JSON	JavaScript Object Notation
LAN	Local Area Network
POS	Point Of Sale
RTC	Real Time Clock
TCP/IP	Transmission Control Protocol/Internet Protocol
TRANSACTIONS	All events where data is recorded
UTC	Coordinated Universal Time
UTF	Unicode Transformation Format
WiFi	Wireless Fidelity



The process flow of the data can be briefly summarized as follows:



Chronologically:

1. The FDM is started for the very first time and receives the necessary settings from the FODFIN cloud.
2. The user registers events on the cash register system.
3. The cash register system immediately sends these events to the FDM in the form of JSON messages, which perform several calculations depending on the type of event and signs the enriched JSON.
4. The FDM sends the response with the control data back to the cash register system.
5. The FDM buffers the enriched JSON of the events.
6. The FDM contacts the FODFIN web service at regular intervals to update its parameters or certificates.
7. The FDM regularly sends the buffered JSON messages to the FODFIN cloud.
8. The received JSON messages are stored in the FODFIN cloud in a secure environment, where they are available for analysis.
9. The cash register system displays notifications from the FDM and/or from the FODFIN cloud via its user interface.

## 1. PHYSICAL CONNECTION

The cash register system and FDM are connected to each other via a cable (LAN) or wirelessly (WiFi). The configuration of this is done in the back office of both the cash register system and the FDM.

**IMPORTANT:** In theory, the wireless connection can also be established via the internet. If the internet connection fails, the connection between the cash register system and FDM will automatically be lost, and the cash register system will stop registering.

The FDM manufacturer provides sufficient options to ensure a secure connection between the cash register system and FDM. Typical examples are: HTTP with token, HTTPS with TLS using self signed certificate, HTTPS with mTLS using self-signed certificate based on the intermediate certificate of the FDM manufacturer.

The ultimate responsibility for correct and secure installation and configuration lies with the distributor/installer.

The cash register system and FDM use the TCP/IP protocol to exchange their messages. These messages are further described below. Other messages are only allowed as long as they do not affect the mandatory features and contribute to the proper functioning of the whole system. The FDM makes its GraphQL service available to the POS via HTTP(S).

The FDM is connected to the FODFIN cloud API via the internet and regularly receives instructions through this connection. In certain cases, see the relevant detailed description, such instructions may lead to a message being displayed on the cash register system and/or a manual intervention via the cash register system.

## 2. COMMUNICATION BETWEEN POS AND FDM

Requests are sent by the cash register system to the GraphQL service of the FDM. JSON data is used as the payload for sending the event information. This payload is sent to the path “/graphql” on the FDM using the HTTP verb “POST” and Content-Type: application/json.

All JSON objects follow the RFC 8259 standard.

String fields **never** contain leading or trailing whitespace.

The sent data includes, among other things:

- transaction data (events P and N);
- financial data (event F);
- social data (event S);
- copies (event C);
- training events (event T);
- invoices (event I);
- reports (event R).

These objects and their conditions are further described in the complete GraphQL schema.

### 2.1. GENERAL

The following events were defined in the ministerial decree:

- NORMAL (label N);
- PRO FORMA (label P);
- TRAINING (label T);
- COPY (label C);
- FINANCIAL (label F);
- SOCIAL (label S);
- INVOICE (label I);
- REPORT (label R).

All these events are sent to the FDM to be digitally signed as a safeguard against data manipulation. For this purpose, various mutations are provided by the GraphQL service of the FDM. These mutations can be used to correctly populate the JSON structure of the registrations on the FDM, as described below. The JSON structure of the FDM can contain any type of event; through the various mutations, the FDM accepts only those fields and objects that are allowed for each type of event.

The following mutations are defined:

For the event S:

- signWorkIn
- signWorkOut

For the event I:

- signInvoice

For the event N:

- signSale

For the event P:

- signCostCenterChange
- signOrder
- signPreBill

For the event F:

- signMoneyInOut
- signDrawerOpen
- signPaymentCorrection

For the event R:

- signReportTurnoverX
- signReportTurnoverZ
- signReportUserX
- signReportUserZ

For the event C:

- signCopy

## 2.2. CONTENT OF THE COMMUNICATION BETWEEN POS AND FDM

The payload from the FDM to FODFIN (see detailed description of the FDM) is entirely based on the structure and validation schema described below.

Consequently, the event data of the mutations must also respect that structure and validation rules. Section 2.3 describes the conversion to the GraphQL schema.

### 2.2.1. POS → FDM – complete overview:

```
{  
  language (scalar value)  
  ticketMedium (scalar value)  
  posId (scalar value)  
  posFiscalTicketNo (scalar value)  
  posSwVersion (scalar value)  
  terminalId (scalar value)  
  deviceId (scalar value)  
  posDateTime (scalar value)  
  bookingPeriodId (scalar value)  
  bookingDate (scalar value)  
  vatNo (scalar value)  
  estNo (scalar value)  
  employeeId (scalar value)  
  customerId (scalar value)  
  customerVatNo (scalar value)  
  invoiceNo (scalar value)  
  fdmRefs (array of fdmRef objects)  
  costCenter (object)  
  transfer (object)  
  transaction (object)  
  drawer (object)  
  financials (array of paymentLine of moneyInOutLine objects)  
  reportNo (scalar value)  
  reportBookingDate (scalar value)  
  posDevices (array of posDevice objects)  
  fdmDevices (array of fdmDevice objects)  
  turnover (object)  
  users (array of user objects)  
}
```

## 2.2.2. POS → FDM - detailed description

### Language

**language** (enum, mandatory)

The language in which the cash system wishes to receive messages from the FDM, to be chosen from the following values:

language	Description
EN	English
NL	Dutch
FR	French
DE	German

### Delivery Method

**ticketMedium** (enum, mandatory )

Type of ticket, to be chosen from the following table:

ticketMedium	Description
NONE	No print
PAPER	Print on paper
DIGITAL	Digital delivery
PAPER_DIGITAL	Print on paper and digital delivery

### POS Identification Details

**posId** (string, mandatory, 14 characters in uppercase)

Serial number of the POS, as registered via the e-service RCRS. Example: CFOD0061234567

**posFiscalTicketNo** (numeric, mandatory, value between 1 and 999999999)

Internal ticket number of the POS. This unique number can be generated across all events, across all terminals, per event type or per terminal, as long as the combination of this number and the respective event type or terminal id is unique. When the maximum value is reached, the POS starts again with the number "1".

**posSwVersion** (string, mandatory, minimum length 1 and maximum length 36 characters)

Current software version of the cash register software. Example: 1.8.3

**terminalId** (string, mandatory, minimum length 1 and maximum length 600 characters)

If a RCRS setup consists of multiple cash registers under the same posId, then a unique terminalId must be used for each cash register. Examples are a logical name (bar, terrace, etc.) or any other identification (number, etc.). The term terminal is completely separate from any hardware component.

If the RCRS setting does not use this, the same value must always be entered in this field. Example: '1', 'n/a', fixed guid.

**deviceId** (string, mandatory, length min. 1 and max. 600 characters)

The hardware identification of the device on which the transaction was recorded.

## POS Date and Time

**posDateTime** (string, mandatory)

Date and time from the POS system, in local time (Brussels), in the ISO 8601 format shown below.

Example: 2022-10-20T15:01:25+02:00 (daylight saving time), 2022-11-03T15:01:25+01:00 (standard time)

**bookingPeriodId** (string, mandatory, format = GUID with hyphens and lowercase letters)

Operational period of the POS: typically, this is an opening day, aligning with the period of a fiscal day report. It can also be a shift as part of an opening day. In the latter case, a fiscal day report will contain data from multiple booking periods.

Example: dffcd829-a0e5-41ca-a0ae-9eb887f95637

**bookingDate** (string, mandatory)

Accounting date on which the transaction is booked, in the ISO 8601 date format shown below. This value refers to the date the bookingPeriodId relates to. A booking period from 00:30 to 04:30 in the morning can indeed be included in the daily turnover of the previous day (date).

Example: 2023-10-20

## Company and User Identification

**vatNo** (string, mandatory, format: BE followed by 10 digits, total length = 12)

VAT number of the company. The numeric part always starts with a '0' or a '1'. The FDM will perform a validity check on this number (using the modulo 97 method).

Example: BE0499999960

**estNo** (string, mandatory, length = 10)

Establishment unit number of the establishment. An establishment unit number always starts with a '2' to '8'. The FDM will perform a validity check on this number (using the modulo 97 method).

Example: 8789456149

**employeeld** (string, mandatory, length = 11)

Social Security registration number (INSZ), consisting of either the national register number or the bis number of the operator or the employed staff member. The FDM will perform a validity check on this number (using the modulo 97 method).

Example: 75061189731

## References

**customerVatNo** (string, conditional, length min. 1 and max. 600 characters)

In the event of 'I' (Invoice), this field is mandatory and contains the VAT number of the co-contractor. The format used is as described in VatNo or the foreign equivalent if the customer is not a Belgian company.

**invoiceNo** (string, conditional, length min. 1 and max. 600 characters)

Invoice number, only to be used for event type 'I'.

**fdmRefs** (optional, array of fdmRef objects, see fdmRef)

This array contains the references assigned by the FDM to previous referenced events. This can be a VAT receipt (event N) that is cancelled **completely**, one or more VAT receipts that are part of an invoice (event I), the VAT receipt whose payment method is corrected (event F) or the creation of a copy of a previous event (event C).

The fdmRef object is described in Chapter 4 – RESPONSE.

## Transactions Registration – Use of Management Options

Intermediate registrations of sales transactions can be assigned to tables, seats, customers, rooms, etc. Such bookings can follow each other frequently or change allocation before resulting in a final VAT receipt. The objects 'costCenter' and 'transfer' are used to register this correctly.

### costCenter

**costCenter** (object, conditional)

Identifies the 'location' of the transaction. Mandatory for transactions where the registration is interrupted (such as table management, customer management, etc.). The object is structured as follows.

{

**id** (string, mandatory, length min. 1 and max. 600 characters).

Description of the costCenter. Example: "T17", "K08"



**type** (enum, mandatory)

A value to be chosen from tb\_costCenterType.

tb\_costCenterType

type
TABLE
CHAIR
ROOM
CUSTOMER
ON_HOLD
OTHER

Note: the value **CHAIR** may only be used if at the same or higher level the value **TABLE** was used.

**reference** (string, mandatory, length min. 1 and max. 600 characters)

Session of a costCenter, grouping all connected transactions together.

Example: "e4eb2509-dce9-44db-8322-e445aaeb19c4"

**costCenter** (costCenter object, optional, max. 2 levels deep)

This nested structure allows, for example, registrations to be made at table and seat level simultaneously.

}

## transfer

**transfer** (object, conditional) The transfer object is used to move already registered deliveries of goods/services from the costCenter they were booked to another costCenter.

Examples are table transfer, table split, table join, transfer to room, transfer to customer account, etc.

{

**from** (array of one or more transferItem objects)

Original costCenter(s), *from where the transfer starts*

**to** (array of one or more transferItem objects)

New costCenter(s), *where the transfer arrives*

}

The transferItem objects in from and to can have a 'one-to-one', a 'many-to-one', or a 'one-to-many' relationship. A 'many-to-many' relationship, where goods or services are transferred from multiple costCenters to multiple other costCenters, is prohibited.

## transferItem

{

**costCenter** (object, mandatory, see costCenter)

Identification of the costCenter from where the transfer starts or to which a transfer arrives.

**transaction** (object, mandatory, see transaction)

The delivered goods/services being transferred from the costCenter.

}

**Important note:**

When using table management, a table is typically closed when all registrations have resulted in event 'N', with the creation of the VAT receipt.

There is one possible exception to this, namely when transferring the contents of the table to the hotel room, where that content is transferred to the overall hotel bill. Two possible ways to close the table are accepted:

- either close with an event 'N' and payment method 'ROOM\_CREDIT' (in this case, the bookings on the table DO appear in the daily turnover of the turnover report),
- or close with an event 'P', where a transfer is registered from a costCenter TABLE to a costCenter ROOM (in this case, the bookings do NOT appear in the daily turnover of the cash register system).

## Transactions Registration – Transaction Lines

**transaction** (object, conditional)

This object contains the registrations of the deliveries of goods and/or services in the POS system.

This object is **mandatory** for the following eventOperations:

- SALE;
- PRE\_BILL;
- ORDER;
- COST\_CENTER\_CHANGE.

This object is **prohibited** to be used in all other eventOperations.

The transaction object always contains a transactionLines array. This can be empty in the case of a zero receipt.

{

**transactionLines** (array of transactionLine objects, see transactionLine for description)

Transaction lines including all products and their (price) changes.

**transactionTotal** (numeric, two decimals)

Total price of the entire event (sum of all lineTotal). This amount can be positive, negative or zero (0).

}

## transactionLine

This object contains a single transaction line in which a maximum of one product is included.

{

**lineType** (enum, mandatory)

Value to be chosen from the table below.

tb\_lineType

lineType
SINGLE_PRODUCT
COMPOSITE_PRODUCT

**mainProduct** (product object, mandatory)

Contains all information about the product included in the transactionLine. If it is a composite product (for example a menu), the array of subProduct objects must also be used. See the description further below.

**subProducts** (array of product objects, conditional)

Contains all information about the products included in the composite product mentioned in the mainProduct object. See the description further below.

**costCenter** (object, optional)

Used if the cash register allows assigning products to a costCenter at the lowest level (for example: chair). The costCenter object was previously described.

Note: the value CHAIR may only be used if at the same or higher level the value TABLE was used.

**lineTotal** (numeric, mandatory)

Total price, VAT included, of the transaction line after applying price changes. Can be positive, negative or zero.

In the case of a SINGLE\_PRODUCT there is only a mainProduct object and the formula is:  
[quantity \* unitPrice] + Sum[priceChangeAmount].

}

**Important note:** composite\_product

Working with a composite\_product allows products to be combined into one new product, with or without discounts applied to arrive at a single price.

A composite\_product may be subject to multiple VAT rates. The following guidelines should be observed when using a composite\_product:

→ **calculation** of the taxable amount and VAT amount by the FDM:

For this, the FDM will rely on the information provided by the cash register about the sub-products.

The basis then becomes, per VAT per subProduct:

$\text{sum}(\text{vatPrice} + \text{sum}([\text{priceChangeAmount}]))$

→ **forwarding** of lineTotal to the FDM:

For the mainProduct, the  $\text{sum}(\text{vatPrice} + \text{sum}([\text{priceChangeAmount}]))$  of all subProducts involved will be send to the FDM.

→ **printing** of the article line of the composite\_product:

On the printout of the VAT receipt, at product line level, only the composite\_product appears, with all VAT rates involved, with as total price the sum(vatPrice + sum([priceChangeAmount])) of all subProducts involved.

## product

Is the subject of a line in a transaction where it contains the details of the supply of a good or service or is part of a composite product.

{

**gtin** (string, optional, length min. 8 and max. 20 characters)

Global Trade Item Number.

Example: "0811571013579"

**productId** (string, mandatory, length min. 1 and max. 600 characters)

Internal product number in the POS system.

**productName** (string, mandatory, length min. 1 and max. 600 characters)

Product name as stored in the POS system.

**departmentId** (string, mandatory, length min. 1 and max. 600 characters)

Internal department ID in the POS system, to which the involved product is linked.

**departmentName** (string, mandatory, length min. 1 and max. 600 characters)

Name of the department to which the involved product is linked in the POS system.

**quantity** (numeric, mandatory, format = multiple of 0.0001)

Number of units of the product included in this item line. Quantity can be positive, negative, or zero (0).

Examples: 1245.4789, 2.5, -17.23.

**negQuantityReason** (enum, conditional)

Reason why a negative quantity was recorded, to be chosen from the table below.

negQuantityReason
REFUND
CORRECTION
PRICE_CHANGE
COST_CENTER_CHANGE
PRODUCT_SUBSTITUTION
VOUCHER
OTHER

- REFUND: A 'good' or 'service', part of a previously fully completed sales transaction, that is being taken back.
- CORRECTION: Any adjustment of a quantity in a not yet definitively completed sales transaction (examples: line cancellation, correction, ticket cancellation).

- PRICE\_CHANGE: When an 'artificial' product line is added to correctly register a price change on a previously booked product line in a not yet definitively completed transaction.

Example: the original order contained 2 beers; after sending the order, a discount is later given on 1 of those 2 beers; for this, 1 beer will first be corrected to then re-register 1 beer with a discount.

**quantityType** (enum, mandatory)

The unit in which the quantity is expressed, value to be chosen from the tb\_quantityType

quantityType
PIECE
KILOGRAM
METER
LITRE
HOUR

**unitPrice** (numeric, two decimals, mandatory)

Normal price of 1 unit of the product, included in the product line, before applying price changes. Can be positive, negative, or zero (0).

**vats** (array of VAT objects, mandatory)

For each VAT label linked to the relevant product, a VAT object is added. This array can contain a maximum of 5 VAT objects.

Note: With a composite\_product, the vats array at the mainProduct will be **empty**. All values are included under the subProducts.

The VAT object is described in vat.

}

**vat**

Each line on a transaction is covered by at least one VAT rate (see number 18 of the ministerial decree). This may also include one or more price changes.

{

**label** (enum, mandatory)

The VAT code, as provided in number 18 of the ministerial decree, applicable to the relevant item.

label	VAT RATE DESCRIPTION	VAT RATE
A	High	21 %
B	Medium	12 %
C	Low	6 %
D	Zero rate	0 %
X	Outside VAT Scope	None

**price** (numeric, two decimals, mandatory)

Part of the sales price (including VAT) of the relevant item that is subject to the aforementioned VAT rate, before applying price changes. This amount can be positive, negative, or zero.

**priceChanges** (array of priceChange objects)

This array is only used if price changes were applied to this part of the sales price of the relevant item during this event. This array can contain multiple objects if multiple price changes are applicable. A maximum of 99 objects are allowed. The priceChange object is described under priceChange.

}

### priceChange

For each applied price change, an object is used. This change can be positive or negative.

{

**groupingId** (numeric, forbidden for scope PRODUCT, mandatory for scope LINE and EVENT, values between 1 and 99)

This number groups a price change applicable to multiple products. These can be multiple products within the scope of one particular transaction line or within the scope of the entire transaction.

**id** (string, mandatory, length min. 1 and max. 600 characters)

ID of the price change, as provided in the POS system, language-independent.

**name** (string, mandatory, length min. 1 and max. 600 characters)

Name of the price change, as provided in the POS system. Examples: "menu discount", "happy hour", "on the house".

**scope** (enum, mandatory)

Indicates to which the price change applies. Value to be chosen from the table tb\_priceChangeScope.

scope	description
PRODUCT	product price change
LINE	transactionLine price change
EVENT	Event price change

**type** (enum, mandatory)

Identification of the type of price change, to be chosen from the table tb\_priceChangeType.

type	description
PUBLIC	Visible to customer
INTERNAL	Internal recalculation

**amount** (numeric, two decimals, mandatory)

Amount of the price change. This can be positive, negative or zero.

}

## Transactions Registration – Financial Movements

### Drawer

**drawer** (object, optional)

May be used in the root of the JSON structure only with an eventOperation DRAWER\_OPEN, when the cash drawer is opened without a transaction and is registered as such in the POS system. drawer can be used if the business wants to specify exactly which cash drawer/cash pouch is being/was used, provided that the POS system supports this.

For using drawer in other transaction types, see financialLine.

```
{  
  id (string, mandatory, length min. 1 and max. 600 characters)  
    Identification of the cash drawer/pouch.  
  name (string, mandatory, length min. 1 and max. 600 characters)  
    Name of the cash drawer/pouch, as provided in the POS system.  
}
```

### Financial Movements

**financials** (array of paymentLine or moneyInOutLine objects)

This array contains all financial lines of an event, in EUR or currency. This array can be used for 'N' events when they relate to a sales transaction, or 'F' events when it concerns a mere financial movement.

The objects paymentLine and moneyInOutLine include a financial movement of a payment method and the meaning of the amount.

### Payment Lines

For an 'N' event, this means the registration of a payment or deferred payment. If the amount to be paid is zero, an empty financials array is allowed.

In the case of a deferred payment, the payment method 'CUSTOMER\_CREDIT' is used, when transferring to a hotel room within an event N, ROOM\_CREDIT is used.

### Mere Financial Movements

For an 'F' event, this means the registration of financial movements unrelated to sales or a change of a payment method of an already completed sales transaction.

**paymentLine** (object)

The paymentLine object allows for the correct registration of a **payment** to enable perfect management of revenues.

{

**id** (string, mandatory, length min. 1 and max. 600 characters)

ID of the payment method, as provided in the POS system, language-independent.

**name** (string, mandatory, length min. 1 and max. 600 characters)

Name of the used payment method, as coded in the POS system. Examples: "Cash", "Debit card".

**type** (enum, mandatory)

Type of payment method, to be chosen from the table `tb_paymentTypes`.

tb\_paymentTypes

type	Note
UNKNOWN	Type not registered
CASH	Cash
CARD_DEBIT	Debit card payment
CARD_UNKNOWN	Card payment with unspecified card
CARD_CREDIT	Debit card payment
CARD_OTHER	Card payment not previously mentioned
CHEQUE_MEAL	Payment with meal voucher
CHEQUE_OTHER	Payment with other cheque (eco, culture, ...)
APP	Payment with payment app
ONLINE	Online Payment
CUSTOMER_CREDIT	Payment on credit/deferred
ROOM_CREDIT	Payment via global hotel invoice
LOYALTY_REWARDS	Payment with loyalty rewards
VOUCHER_STORE	Payment with store voucher
VOUCHER_SUPPLIER	Payment with supplier voucher
VOUCHER_OTHER	Payment with other voucher not previously mentioned
OTHER	Any other not previously mentioned

**provider** (string, optional, max. length = 600 characters)

Issuer of the payment method (for example: Visa, MasterCard, Edenred, ...).

**inputMethod** (enum, mandatory)

One of the values from the table `tb_inputMethod`:

Type	description
MANUAL	Manually registered
AUTOMATIC	EFT, other integrations

If the POS system is directly connected to the payment terminal (and thus registers the payment via EFT), the payment terminal itself sends back which type of card was used.

**amount** (numeric, two decimals, mandatory)

Amount of the payment line or financial movement, expressed in euro. This amount can be positive, negative or zero (0).



**amountType** (enum, mandatory)

Describes what the amount pertains to, type to be chosen from the tb\_amountType:

Type	description
<b>PAYMENT</b>	Payment
<b>TIP</b>	Tip
<b>ROUNDING</b>	Rounding on payment

- Tip: used to register received tips. A tip can be received within the context of a transaction or can be registered with a financial movement.
- Rounding: mandatory to register the rounding on payment methods<sup>1</sup>.

Example: amount to be paid, 9.97 and the customer pays cash. POS must round to 9.95. This means that two payment lines must be mentioned.

```
{ "name" : "CONTANT", "type" : "CASH", "inputMethod": "MANUAL", "amount" : 9.97, "amountType": "PAYMENT" },
```

```
{ "name" : "CONTANT", "type" : "CASH", "inputMethod": "MANUAL", "amount": -0.02, "amountType": "ROUNDING" }
```

**foreignCurrency** (object, optional, see foreignCurrency)

This object is only used if the POS system supports payments in foreign currencies.

Note: this object may not be used in combination with amountType ROUNDING.

{

**amount** (numeric, mandatory)

The amount in foreign currency. This amount can be positive, negative, or zero (0).

**iso** (string, mandatory, length = 3)

ISO 4217 notation (alphabetic code in uppercase) of the relevant currency.

Examples: "USD", "GBP", "SEK".

}

**reference** (string, optional, length min. 1 and max. 600 characters)

If used by the POS system, the payment reference can be given here.

**drawer** (object, optional, description see earlier)

}

**moneyInOutLine** (object)

The moneyInOutLine object allows for the correct registration of a **financial movement** to enable perfect management of revenues.

<sup>1</sup> Articles VI.7/1 and VI.7/2 of the Economic Law Code require the merchant to round the total amount of the consumer's cash payment to the lower or higher multiple of 5 cents. Provided certain conditions are met (see FPS Economy website), the merchant may also round all other payment methods. See paragraphs 46 and following of this ministerial decree for full information on this.

{

**id** (string, mandatory, length min. 1 and max. 600 characters)

ID of paymentLine, as provided in the POS system, language-independent.

**name** (see paymentLine.name)

**type** (see paymentLine.type)

**provider** (see paymentLine.provider)

**inputMethod** (see paymentLine.inputMethod)

**amount** (see paymentLine.amount)

**amountType** (enum, mandatory)

Describes what the amount pertains to, type to be chosen from the tb\_amountType:

Type	description
<b>MONEY_IN_OUT</b>	Change in + or - of the content of the cash drawer
<b>TIP</b>	Tip
<b>DRAWER_DECLARATION</b>	Counting of cash drawer contents

Tip: used to register received tips. A tip can be received within the context of a transaction or can be registered with a financial movement.

**foreignCurrency** (see paymentLine.foreignCurrency)

**reference** (see paymentLine.reference)

**drawer** (see paymentLine.drawer)

}

## Reports

These are built using the contents of the values, objects, or arrays below.

**reportNo** (numeric, mandatory only for Z, value between 1 and 999999999)

Continuous numbering of the report (only Z), see number 27 of the ministerial decree. Part of both the turnover and user report.

**reportBookingDate** (string, mandatory)

The bookingDate to which this report refers, in the previously mentioned ISO 8601 format.

**posDevices** (array of posDevice objects, mandatory)

Contains the information of all POS systems whose figures are included in this report. Part of both the turnover and user report.

**fdmDevices** (array of fdmDevice objects, mandatory)

Contains the information of all FDMs linked to the POS system and whose figures are included in this report. Part of both the turnover and user report.

**turnover** (object, mandatory part of the turnover report)

Contains all sales-related information for the relevant reportBookingDate.

**users** (array of objects, mandatory part of the users report)

Contains all relevant information about the POS users for the relevant reportBookingDate.

### posDevice

This object contains the identity and the timeframe of all POS systems whose data is included in this report.

{

**posId** (string, mandatory)

The posId as used during event registration.

**terminalId** (string, mandatory)

The terminalId as used during event registration.

**firstPosDateTime** (string, mandatory)

The timestamp of the first registration with this posId and terminalId combination for the given bookingDate.

**lastPosDateTime** (string, mandatory)

The timestamp of the last registration with this posId and terminalId combination for the given bookingDate.

}

## fdmDevice

This object contains the details of all FDMs whose figures are included in the report and the reference to the first and last transaction included.

```
{  
  fdmId (string, mandatory)  
  firstFdmDateTime (string, mandatory)  
  firstTotalCounter (numeric, mandatory, min. 1 and max. 999999999)  
  lastFdmDateTime (string, mandatory)  
  lastTotalCounter (numeric, mandatory, min. 1 and max. 999999999)  
}
```

## turnover

This conditional object contains the total realized turnover, the turnover per department, and per VAT rate. It is only used in the turnover report.

```
{  
  transactions (array of all used eventLabels with their totals)  
    [  
      {  
        eventLabel (string, mandatory, see previous description)  
        ticketCount (integer, mandatory)  
          Total number of events of the relevant eventLabel during the relevant bookingDate.  
        amount (numeric, mandatory, two decimals)  
          Sum(VatInput.Price)+Sum(VatInput.PricesChanges.Amount) of each ProductInput with  
          this departmentId.  
      }  
    ]  
  departements (array of objects containing turnover data for all used departments, mandatory)  
    [  
      {  
        departmentId (string, mandatory, see previous description)  
        departmentName (string, mandatory, see previous description)  
        amount (numeric, two decimals, mandatory)  
          Total amounts transactionTotal limited to the eventLabel "N" for the relevant Department  
          during the relevant bookingDate.  
      }  
    ]  
}
```

]

**vats** (array of objects containing turnover data for all used VAT rates, mandatory)

[

{

**label** (string, mandatory, see previous description)

**rate** (numeric, mandatory, min. value = 0 and max. value = 100, two decimals, see description in Chapter 4 of this appendix)

**taxableAmount** (numeric, two decimals, mandatory)

The total amount of the values of the taxableAmount (see description under 2.2.4) for the relevant label.

**vatAmount** (numeric, two decimals, mandatory)

The total amount of the values of the vatAmount (see description under 2.2.4) for the relevant label.

**totalAmount** (numeric, two decimals, mandatory)

The total amount of the values of all totalAmount (see description under 2.2.4) for the relevant label.

}

]

**payments** (array of objects containing the received amounts related to the previously mentioned turnovers, per payment type, mandatory)

[

{

**id** (string, mandatory, see previous description)

**name** (string, mandatory, see previous description)

**type** (enum, mandatory, see previous description)

**amount** (array of PaymentTotalAmount objects, mandatory)

[

{

**type** (enum PaymentLineType, mandatory)

**normalAmount** (numeric, two decimals, mandatory)

Contains the total amount of all payments related to realised sales, to be reported in euro (Sale).

**negativeCorrections** (numeric, two decimals, mandatory)

Contains the total amount of negative payment corrections.

**positiveCorrections** (numeric, two decimals, mandatory)

Contains the total amount of positive payment corrections.

**correctionsCount** (integer, mandatory)

The total number of payment corrections.

**totalAmount** (numeric, two decimals, mandatory)

The total of the normalAmount, negativeCorrections and positiveCorrections

**normalForeign** (array of foreignCurrency objects, conditional)

Contains the total amount per currency of all foreign currency payments related to realised sales (Sale).

**correctionsForeign** (array of foreignCurrency objects, conditional)

Contains the total amount of payment corrections in foreign currency. These amounts are already included in euro in the other fields (negativeCorrections, positiveCorrections and totalAmount).

}

]

}

]

**drawersOpenCount** (numeric, min. 0 and max. 999999999, mandatory)

Contains the number of times one or more cash drawers were opened (via the POS system) without sales registration.

**negQuantities** (array of objects, mandatory)

Contains the number and total amount for all used negative quantities.

[

{

**reason** (enum, mandatory, see previous description)

**count** (numeric, min. 1 and max. 999999999)

The number of times the relevant negative registration was used during the bookingDate.

**ticketCount** (numeric, min. 1 and max. 999999999, mandatory)

The number of tickets for that Reason.

**amount** (numeric, two decimals, mandatory)

The total amount of these negative registrations for this type.

}

]

**priceChanges** (array of objects, mandatory)

Contains a complete overview of the booked price changes.

[

{

**id** (string, mandatory, length min. 1 and max. 600 characters)

Id of the price change, as provided in the POS system, language-independent.

**name** (string, mandatory, see previous description of priceChange.name)

**type** (enum, mandatory, see previous description of priceChange.type)

To be limited to the priceChange.type PUBLIC

**amount** (array of objects, mandatory)

Contains the total amount of all price changes during the relevant bookingDate, broken down by used VAT rate and whether it concerns positive or negative price changes.

```
[
```

```
{
```

**label** (string, mandatory, see previous description)

**negative** (numeric, two decimals, mandatory)

Is the sum of all **negative** price changes for the given VAT rate.

**positive** (numeric, two decimals, mandatory)

Is the sum of all **positive** price changes for the given VAT rate.

```
}
```

```
]
```

```
}
```

```
]
```

**invoices** (array of objects, conditional)

Overview of the invoices created with the POS during the relevant bookingDate.

```
[
```

```
{
```

**invoiceNo** (string, mandatory, see previous description)

**amount** (numeric, mandatory, see previous description of transactionTotal)

```
}
```

```
]
```

```
}
```

## users

This array of objects contains the information of the user report, broken down per user.

```
[
```

```
{
```

**employeeId** (string, mandatory, see previous description)

**totalAmount** (numeric, two decimals, mandatory)

The total amount of all transactionTotal that were registered via the events 'N' by the relevant user.

**firstPosDateTime** (string, mandatory, format as described for posDateTime)

Timestamp of the **first** registration of an event by the relevant user in this period.

**lastPosDateTime** (string, mandatory, format as described for posDateTime)

Timestamp of the **last** registration of an event by the relevant user in this period.

**socialEvents** (array of objects, mandatory)

Contains the timestamps of the login/out of the POS for the relevant user via the use of the event 'S'.

```
[  
{
```

**posDateTime** (string, mandatory, see previous description)

Timestamp of the registration of the relevant event 'S'.

**inOut** (enum, mandatory)

Value to be chosen from the tb\_inOut

inOut	description
IN	Work In
OUT	Work Out

```
}  
]
```

**payments** (array of objects, mandatory)

This array contains the summary of all registrations of all payments for the relevant user for this bookingDate, broken down by used type. This has the same structure as that in the turnover object.

## Communication Rules

To ensure the correct handling of requests and uniquely distinguish them, the following key fields are defined:

- **posId**
- **posDateTime**
- **terminalId**
- **eventLabel**
- **posFiscalTicketNo**

When a POS sends a request to the FDM within a time span of 10 minutes, with these 5 key fields identical to those sent in a previous message, the FDM must verify whether this is a request to which a response has already been given (including updating the relevant counters on the FDM).



The following situations can occur:

1. The combination of key fields is different: the FDM handles this request in the normal way.
2. The content of the JSON request is completely identical: the FDM handles this request as a "duplicate" submission and sends back the same data in the response as in the original request; the FDM does not update its counters; this duplicate transaction is not recorded on the FDM.
3. The combination of key fields is identical, but the other values in the JSON request are different from those in the previously processed request: the FDM refuses to process this request and sends an error code in its response.

### 2.2.3. FDM → POS – complete overview

After receiving the query described above, the FDM performs the necessary calculations, updates the necessary counters, creates the canonical JSON object, and signs the data.

The GraphQL service of the FDM sends the requested fields back to the POS.

Below is the complete response overview:

**posId** (scalar value)  
**posFiscalTicketNo** (scalar value)  
**posDateTime** (scalar value)  
**terminalId** (scalar value)  
**deviceId** (scalar value)  
**eventOperation** (scalar value)  
**fdmRef** (object)  
**fdmSwVersion** (scalar value)  
**digitalSignature** (scalar value)  
**shortSignature** (scalar value)  
**verificationUrl** (scalar value)  
**vatCalc** (array of objects)  
**bufferCapacityUsed** (scalar value)  
**warnings** (array of objects)  
**informations** (array of objects)

### 2.2.4. FDM → POS – detailed overview

A number of arrays, objects, or values not previously described are described at this point.

**eventOperation** (enum, mandatory)

Value from the table `tb_eventOperation`.

`tb_eventOperation`

eventOperation	eventLabel	description
<b>WORK_IN</b>	S	Clocking in on the POS
<b>WORK_OUT</b>	S	Clocking out on the POS
<b>SALE</b>	N	Closing sales transaction
<b>INVOICE</b>	I	Invoice of previously created VAT receipt
<b>COST_CENTER_CHANGE</b>	P	Change of cost center allocation
<b>ORDER</b>	P	Registration of order on cost center
<b>PRE_BILL</b>	P	Creation of provisional bill
<b>MONEY_IN_OUT</b>	F	Registration of money movement outside sales
<b>DRAWER_OPEN</b>	F	Opening cash drawer outside sales

eventOperation	eventLabel	description
PAYMENT_CORRECTION	F	Change of previously recorded payment method
REPORT_TURNOVER_X	R	Turnover report X
REPORT_TURNOVER_Z	R	Turnover report Z
REPORT_USER_X	R	User report X
REPORT_USER_Z	R	User report Z
COPY	C	Copy of an eventOperation mentioned above
(eventOperation)	T	All previous eventOperation in training mode

**fdmRef** (object, mandatory)

Contains the identification, timestamp, and counter values of the FDM. This uniquely identifies the signed transaction.

{

**fdmId** (string, mandatory)

The serial number of the FDM responding to the request. Length = 11.

Example: FOD01987654

**fdmDateTime** (string, mandatory)

The date and time of the response from the RTC of the FDM. **UTC+00** is always used (thus: no use of summer or winter time). The following ISO 8601 format is mandatory.

Example: 2022-10-20T15:01:26Z

**eventLabel** (enum, mandatory, length = 1)

The eventLabel corresponding to the mutation sent by the POS to the GraphQL service of the FDM, based on which the FDM increased the appropriate counter. This response therefore returns the event type itself (see Chapter 3).

Example: 'N'

**eventCounter** (numeric, mandatory, min. 1 and max. 999999999)

The event counter. The FDM has a number of built-in counters, see number 64 of the ministerial decree. The FDM provides the latest update (after adjustment based on the content of the relevant request) of the counter of the relevant event.

Example: 46895

**totalCounter** (numeric, mandatory, min. 1 and max. 999999999)

The total event counter. The FDM has a number of built-in counters, see number 64 of this decree. The FDM provides the latest value (after adjustment based on the content of the relevant request) of the total counter.

Example: 53896

}

**digitalSignature** (string, mandatory, min 1 character)

This field contains the digital signature placed by the signature certificate of the FDM. This signature is placed according to the rules described in section 2.4.

**shortSignature** (string, conditional, min 1 character)

This field contains the short digital signature. This signature will be **printed** on the VAT receipt. Therefore, it is **only** sent in the case of a 'NORMAL' event. It is calculated as follows:

$\text{shortSignature} = \text{Hex}(\text{SHA1}(\text{Base64Decode}(\text{digitalSignature})))$ .

**verificationUrl** (string, conditional, min. 1, max. 60)

This URL is generated by the FDM to be printed as a QR code on the VAT receipt (see section 6.5). The prefix is provided by the FODFIN to the FDM, which further creates the URL based on certain event data.

**This string is only added to the response JSON for events of type 'N'.**

**fdmSwVersion** (string, mandatory, min. 1 and max. 10 characters)

The version of the FDM firmware at the time of response. Example: "1.2.0"

**bufferCapacityUsed** (numeric, mandatory, two decimals)

The used buffer capacity expressed as a percentage.

**vatCalc** (array of vatCalcItem objects, mandatory if vat objects are included in the request)

The FDM is responsible for correctly calculating the taxable amount and VAT for each applicable rate. This calculation is only performed for transaction type/event label 'N'.

For each VAT code, the taxable amount and VAT amount are calculated.

The contents of the fields vatId and rate are determined according to the data provided by the FPS Finance (art. 66 of the ministerial decree). The contents of the fields taxableAmount and vatAmount are calculated by the FDM.

An object is filled in only for the vatLabels for which product lines were sent in the request.

[

{

**label** (enum, mandatory)

The VAT label as received from the FPS Finance.

Example: "B"

**rate** (numeric, mandatory, min. value = 0 and max. value = 100, two decimals)

The percentage for the relevant VAT label, as received by the FDM from the FPS Finance.

Example: 12

**taxableAmount** (numeric, two decimals, mandatory)

Taxable amount for the relevant VAT rate. For the calculation, refer to number 66 of the ministerial decree. This value can be positive, negative, or zero (0).

Example: 8.93

**vatAmount** (numeric, two decimals, mandatory)

VAT amount for the relevant VAT rate and calculated based on the taxable amount provided above. For the calculation, refer to number 66 of the ministerial decree. This value can be positive, negative, or zero (0).

Example: 1.07

**totalAmount** (numeric, two decimals, mandatory)

**outOfScope** (true or false, mandatory)

}

]

**warnings** (array of message objects, optional, max. 50 objects, see description)

List of warnings the FDM wants/needs to alert the POS about.

### Message

{

**category** (enum, mandatory)

Value to be chosen from the table tb\_messageCategory.

tb\_messageCategory

categorie	omschrijving
<b>SPF_FOD</b>	Messages from the FPS Finance sent to the FDM via the API. Codes from tb_Codes must be used for these.
<b>FDM</b>	Messages from the FDM – freely filled in by the producer.
<b>OTHER</b>	Other messages.

**code** (string, mandatory)

The content of the warning or information provided to the POS by the FDM. For the SPF\_FOD category, only codes from tb\_messages are used. For other categories, they are freely chosen by the producers.

**message** (string, mandatory)

Description of the message, as defined in this decree or by the producer of the FDM. The language used corresponds to the language indicated in the request's language field. For messages related to the SPF\_FOD category, these must be based on the messages from tb\_messages below.

TB\_messages

Code	Message
CLIENT_CERT_NEAR_EXPIRATION	Client certificate expires in less than 4 months
CLIENT_CERT_EXPIRED	Client certificate has expired
RTC_SYNC_FAILED	FDM could not synchronize real-time clock
UPDATE_URLS_FAILED	FDM could not update URLs
UPDATE_TRUST_CERT_FAILED	FDM fails to update trust certificates
UPDATE_TASK_LIST_FAILED	FDM could not download the task list
TASK_FEEDBACK_FAILED	FDM could not report the task result
NOP_FAILED	FDM could not post a NOP message
BUFFER_NEAR_FULL	FDM buffer usage exceeds 70 %
SERVER_CERT_RESOLVE_FAILED	FDM could not resolve server certificate

TRANSACTION_UPLOAD_FAILED	FDM has more than 10 failed attempts to post business transaction
INITIALIZATION_FAILED	FDM initialization has failed
RTC_NOT_INITIALIZED	FDM Real Time Clock not synchronized
CORRUPT_RECORD_ENCOUNTERED	FDM memory contains corrupt transaction
UPDATE_PARAMS_FAILED	FDM could not update parameters
UPDATE_CLIENT_CERT_FAILED	FDM could not update the client certificate
UPDATE_VAT_RATES_FAILED	FDM could not update the vat rates
UDPATE_POS_ALLOWLIST_FAILED	FDM could not update the POS allowlist

**data** (array of data objects, optional)

May contain additional useful information intended for developers or support staff. Not intended to be displayed via the POS user interface.

**showPos** (boolean, mandatory)

This value (true, false) determines whether the message is shown to the user via the POS user interface. For the SPF\_FOD category, the default value is 'true'.

}

Example:

```
{
  "category": "SPF_FOD",
  "code": "BUFFER_NEAR_FULL",
  "message": "FDM buffer usage exceeds 70 %",
  "data": [],
  "showPos": true
}
```

## Data

{

**name** (string, mandatory)

Contains the name of the value. For example: Offset (shows where the error is), Expected (what was expected), Encountered (what was sent).

**value** (string, mandatory)

The extra value provided to a Message.

}

### 2.2.5. FDM → POS errorhandling

Errors are returned in the errors array according to the GraphQL format. Each error object in the array must contain a 'message' string, which provides a description of the error, optionally a 'locations' array indicating where in the query the error occurred, and optionally a 'path' array indicating the path to the incorrect data.

All errors related to syntax, validation, and runtime are determined by the FDM.

**Example:**

```
{
  "errors": [
    {
      "message": "Variable '$data' got invalid value 'N' at 'data.language'; Value 'N' does not exist in 'Language' enum. Did you mean the enum value 'EN' or 'NL'?",
      "locations": [
        {
          "line": 1,
          "column": 20
        }
      ]
    }
  ]
}
```

Errors determined by the administration are added to the error object with the structure of the 'message' object in the extensions object.

For the message object, the same structure applies as the message object in the warnings and informations array.

{

**category** (enum, mandatory)

Value to be chosen from the tb\_messageCategory (see above under message).

**code** (enum, mandatory)

The content of the error provided to the POS by the FDM. If requested, this content is shown on the POS screen. For the SPF\_FOD category, only codes from the tb\_error\_messages are used. For other categories, they are freely chosen by the producers.

**message** (string, mandatory)

Description of the message, as defined in this decree or by the producer of the FDM. The language used corresponds to the language indicated in the request's language field. For messages related to the SPF\_FOD category, these must be based on the messages from tb\_error\_messages below.

tb\_error\_messages

Code	Message
<b>BUFFER_FULL</b>	FDM buffer is full
<b>UNKOWN_POS</b>	POS is not registered for use with this FDM
<b>FDM_LOCKED</b>	FDM is locked
<b>TOO_MANY_MEMORY_ERRORS</b>	FDM's memory contains too many corrupt transactions

**data** (array of data objects, optional)

May contain additional useful information intended for developers or support staff. Not intended to be displayed via the POS user interface. (See description of data object under point 2.2.4).

**showPos** (boolean, mandatory)

This value (true, false) determines whether the error message is shown to the user via the POS user interface. For the category SPF\_FOD, the default value is 'true'.

}

### Example error response:

```
{
  "errors": [
    {
      "message": "FDM error",
      "extensions": {
        "category": "SPF_FOD",
        "code": "BUFFER_FULL",
        "message": "FDM buffer is full.",
        "showPos": true
      }
    }
  ]
}
```

### General note on error handling

Certain errors and warnings are displayed to the user through the POS user interface. In addition to the provisions above, the FDM manufacturer provides the necessary additional information for the user so that they can take appropriate action to resolve the error.

This is extensively described in the documentation provided with the certification application.

## 2.3. GRAPHQL SCHEMA COMMUNICATION BETWEEN POS AND FDM

The complete schema is described below.

### 2.3.1. Mutations

Mutations available on the FDM:

- signWorkIn(data: WorkInOutInput!, isTraining: Boolean! = false): SignResult
- signWorkOut(data: WorkInOutInput!, isTraining: Boolean! = false): SignResult
- signInvoice(data: InvoiceInput!, isTraining: Boolean! = false): SignResult
- signSale(data: SaleInput!, isTraining: Boolean! = false): SignResult
- signCostCenterChange(data: CostCenterChangeInput!, isTraining: Boolean! = false): SignResult
- signOrder(data: OrderInput!, isTraining: Boolean! = false): SignResult
- signPreBill(data: PreBillInput!, isTraining: Boolean! = false): SignResult
- signMoneyInOut(data: MoneyInOutInput!, isTraining: Boolean! = false): SignResult
- signDrawerOpen(data: DrawerOpenInput!, isTraining: Boolean! = false): SignResult
- signPaymentCorrection(data: PaymentCorrectionInput!, isTraining: Boolean! = false): SignResult
- signReportTurnoverX(data: ReportTurnoverXInput!, isTraining: Boolean! = false): SignResult
- signReportTurnoverZ(data: ReportTurnoverZInput!, isTraining: Boolean! = false): SignResult
- signReportUserX(data: ReportUserXInput!, isTraining: Boolean! = false): SignResult

- signReportUserZ(data: ReportUserZInput!, isTraining: Boolean! = false): SignResult
- signCopy(data: CopyInput!, isTraining: Boolean! = false): SignResult

### 2.3.2. Types

```
type SignResult {  
  posId: String!  
  posFiscalTicketNo: Int!  
  posDateTime: String!  
  terminalId: String  
  deviceId: String!  
  eventOperation: EventOperation!  
  fdmRef: FdmReference!  
  fdmSwVersion: String!  
  digitalSignature: String!  
  shortSignature: String  
  verificationUrl: String  
  vatCalc: [VatCalcItem!]  
  bufferCapacityUsed: Float!  
  warnings: [MessageItem!]  
  informations: [MessageItem!]  
}
```

```
type FdmReference {  
  fdmId: String!  
  fdmDateTime: String!  
  eventLabel: EventLabel!  
  eventCounter: Int!  
  totalCounter: Int!  
}
```

```
type VatCalcItem {  
  label: VatLabel!  
  rate: Float!  
  taxableAmount: Float!  
  vatAmount: Float!  
  totalAmount: Float!  
  outOfScope: Boolean!  
}
```

```
type MessageItem {  
  category: Category!  
  code: Code!  
  message: String!  
  data: [DataItem!]  
  showPos: Boolean!  
}
```

```
type DataItem {  
  name: String!  
  value: String!  
}
```



### 2.3.3. Enums

This section describes the various enums. Always consult the most recent version on the website [www.geregistreerdkassasysteem.be](http://www.geregistreerdkassasysteem.be).

```
enum Language {  
    EN  
    NL  
    FR  
    DE  
}
```

```
enum TicketMedium {  
    NONE  
    PAPER  
    DIGITAL  
    PAPER_DIGITAL  
}
```

```
enum InOut {  
    IN  
    OUT  
}
```

```
enum PriceChangeType {  
    PUBLIC  
    INTERNAL  
}
```

```
enum PriceChangeScope {  
    PRODUCT  
    LINE  
    EVENT  
}
```

```
enum VatLabel {  
    A  
    B  
    C  
    D  
    X  
}
```

```
enum EventLabel {  
    N  
    P  
    F  
    S  
    I  
    R  
    C  
    T  
}
```

```
enum QuantityType {  
    PIECE  
    KILOGRAM  
    METER  
    LITRE  
    HOUR  
}
```

```
enum TransactionLineType {  
    SINGLE_PRODUCT  
    COMPOSITE_PRODUCT  
}
```

```
enum NegQuantityReason {  
    REFUND  
    CORRECTION  
    PRICE_CHANGE  
    COST_CENTER_CHANGE  
    PRODUCT_SUBSTITUTION  
    VOUCHER  
    OTHER  
}
```

```
enum PaymentType {  
    UNKNOWN  
    CASH  
    CARD_UNKNOWN  
    CARD_DEBIT  
    CARD_CREDIT  
    CARD_OTHER  
    CHEQUE_MEAL  
    CHEQUE_OTHER  
    APP  
    ONLINE  
    CUSTOMER_CREDIT  
    ROOM_CREDIT  
    LOYALTY_REWARDS  
    VOUCHER_STORE  
    VOUCHER_SUPPLIER  
    VOUCHER_OTHER  
    OTHER  
}
```

```
enum InputMethod {  
    MANUAL  
    AUTOMATIC  
}
```

```
enum PaymentLineType {  
    PAYMENT  
    TIP  
    ROUNDING  
}
```

```
enum MoneyInOutLineType {  
    MONEY_IN_OUT  
    TIP  
    DRAWER_DECLARATION  
}
```

```
}
```

```
enum CostCenterType {
```

```
TABLE  
CHAIR  
ROOM  
CUSTOMER  
ON HOLD  
OTHER
```

```
}
```

```
enum EventOperation {
```

```
WORK_IN  
WORK_OUT  
SALE  
INVOICE  
COST_CENTER_CHANGE  
ORDER  
PRE_BILL  
MONEY_IN_OUT  
DRAWER_OPEN  
PAYMENT_CORRECTION  
COPY  
REPORT_TURNOVER_X  
REPORT_TURNOVER_Z  
REPORT_USER_X  
REPORT_USER_Z
```

```
}
```

```
enum Category {
```

```
SPF_FOD  
FDM  
OTHER
```

```
}
```

```
enum Code {
```

```
CLIENT_CERT_NEAR_EXPIRATION  
CLIENT_CERT_EXPIRED  
RTC_SYNC_FAILED  
UPDATE_URLS_FAILED  
UPDATE_TRUST_CERT_FAILED  
UPDATE_TASK_LIST_FAILED  
TASK_FEEDBACK_FAILED  
NOP_FAILED  
BUFFER_NEAR_FULL  
SERVER_CERT_RESOLVE_FAILED  
TRANSACTION_UPLOAD_FAILED  
INITIALIZATION_FAILED  
RTC_NOT_INITIALIZED  
CORRUPT_RECORD_ENCOUNTERED  
UPDATE_PARAMS_FAILED  
UPDATE_CLIENT_CERT_FAILED  
UPDATE_VAT_RATES_FAILED  
UPDATE_POS_ALLOWLIST_FAILED  
BUFFER_FULL  
UNKOWN_POS  
FDM_LOCKED  
TOO_MANY_MEMORY_ERRORS
```

```
}
```

## 2.3.4. Input Objects

```
input FdmReferenceInput {  
  fdmId: String!  
  fdmDateTime: String!  
  eventLabel: EventLabel!  
  eventCounter: Int!  
  totalCounter: Int!  
}
```

```
input DrawerInput {  
  id: String!  
  name: String!  
}
```

```
input CostCenterInput {  
  id: String!  
  type: CostCenterType!  
  reference: String!  
  costCenter: CostCenterInput  
}
```

```
input PriceChangeInput {  
  groupingId: Int  
  id: String!  
  name: String!  
  scope: PriceChangeScope!  
  type: PriceChangeType!  
  amount: Float!  
}
```

```
input VatInput {  
  label: VatLabel!  
  price: Float!  
  priceChanges: [PriceChangeInput!]  
}
```

```
input ProductInput {  
  gtin: String  
  productId: String!  
  productName: String!  
  departmentId: String!  
  departmentName: String!  
  quantity: Float!  
  quantityType: QuantityType!  
  negQuantityReason: NegQuantityReason  
  unitPrice: Float!  
  vats: [VatInput!]  
}
```

```
input ForeignCurrencyInput {  
  amount: Float!  
  iso: String!  
}
```

```
input TransactionLineInput {  
  lineType: TransactionLineType!
```

```
mainProduct: ProductInput!  
subProducts: [ProductInput!]  
costCenter: CostCenterInput  
lineTotal: Float!  
}
```

```
input TransactionInput {  
  transactionLines: [TransactionLineInput!]!  
  transactionTotal: Float!  
}
```

```
input PaymentLineInput {  
  id: String!  
  name: String!  
  type: PaymentType!  
  provider: String  
  inputMethod: InputMethod!  
  amount: Float!  
  amountType: PaymentLineType!  
  foreignCurrency: ForeignCurrencyInput  
  reference: String  
  drawer: DrawerInput  
}
```

```
input TransferInput {  
  from: [TransferItemInput!]!  
  to: [TransferItemInput!]!  
}
```

```
input TransferItemInput {  
  costCenter: CostCenterInput!  
  transaction: TransactionInput!  
}
```

```
input MoneyInOutInput {  
  language: Language!  
  vatNo: String!  
  estNo: String!  
  posId: String!  
  posFiscalTicketNo: Int!  
  posDateTime: String!  
  posSwVersion: String!  
  terminalId: String  
  deviceId: String!  
  bookingPeriodId: String!  
  bookingDate: String!  
  ticketMedium: TicketMedium!  
  employeeId: String!  
  financials: [MoneyInOutLineInput!]!  
}
```

```
input MoneyInOutLineInput {  
  id: String!  
  name: String!  
  type: PaymentType!  
  provider: String
```

```
inputMethod: InputMethod!  
amount: Float!  
amountType: MoneyInOutLineType!  
foreignCurrency: ForeignCurrencyInput  
reference: String  
drawer: DrawerInput  
}
```

```
input WorkInOutInput {  
  language: Language!  
  vatNo: String!  
  estNo: String!  
  posId: String!  
  posFiscalTicketNo: Int!  
  posDateTime: String!  
  posSwVersion: String!  
  terminalId: String  
  deviceId: String!  
  bookingPeriodId: String!  
  bookingDate: String!  
  ticketMedium: TicketMedium!  
  employeId: String!  
}
```

```
input InvoiceInput {  
  language: Language!  
  vatNo: String!  
  estNo: String!  
  posId: String!  
  posFiscalTicketNo: Int!  
  posDateTime: String!  
  posSwVersion: String!  
  terminalId: String  
  deviceId: String!  
  bookingPeriodId: String!  
  bookingDate: String!  
  ticketMedium: TicketMedium!  
  employeId: String!  
  invoiceNo: String!  
  customerVatNo: String!  
  costCenter: CostCenterInput  
  fdmRefs: [FdmReferenceInput!]!  
}
```

```
input SaleInput {  
  language: Language!  
  vatNo: String!  
  estNo: String!  
  posId: String!  
  posFiscalTicketNo: Int!  
  posDateTime: String!  
  posSwVersion: String!  
  terminalId: String  
  deviceId: String!  
  bookingPeriodId: String!  
  bookingDate: String!  
  ticketMedium: TicketMedium!  
  employeId: String!  
  fdmRef: FdmReferenceInput
```

```
costCenter: CostCenterInput
transaction: TransactionInput!
financials: [PaymentLineInput!]
}
```

```
input CostCenterChangeInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
  terminalId: String
  deviceId: String!
  bookingPeriodId: String!
  bookingDate: String!
  ticketMedium: TicketMedium!
  employeeId: String!
  transfer: TransferInput!
}
```

```
input OrderInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
  terminalId: String
  deviceId: String!
  bookingPeriodId: String!
  bookingDate: String!
  ticketMedium: TicketMedium!
  employeeId: String!
  costCenter: CostCenterInput
  transaction: TransactionInput!
}
```

```
input PreBillInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
  terminalId: String
  deviceId: String!
  bookingPeriodId: String!
  bookingDate: String!
  ticketMedium: TicketMedium!
  employeeId: String!
  costCenter: CostCenterInput
  transaction: TransactionInput!
}
```

```
input DrawerOpenInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
  terminalId: String
  deviceId: String!
  bookingPeriodId: String!
  bookingDate: String!
  ticketMedium: TicketMedium!
  employeeId: String!
  drawer: DrawerInput
}
```

```
input PaymentCorrectionInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
  terminalId: String
  deviceId: String!
  bookingPeriodId: String!
  bookingDate: String!
  ticketMedium: TicketMedium!
  employeeId: String!
  financials: [PaymentLineInput!]!
  fdmRef: FdmReferenceInput!
}
```

```
input CopyInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
  terminalId: String
  deviceId: String!
  bookingPeriodId: String!
  bookingDate: String!
  ticketMedium: TicketMedium!
  employeeId: String!
  fdmRef: FdmReferenceInput!
}
```

```
input ReportTurnoverXInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
```



```
terminalId: String
deviceId: String!
bookingPeriodId: String!
bookingDate: String!
ticketMedium: TicketMedium!
employeeId: String!
reportBookingDate: String!
posDevices: [PosDeviceInput!]!
fdmDevices: [FdmDeviceInput!]!
turnover: TurnoverInput!
}
```

```
input PosDeviceInput {
  posId: String!
  terminalId: String
  firstPosDateTime: String!
  lastPosDateTime: String!
}
```

```
input FdmDeviceInput {
  fdmId: String!
  firstFdmDateTime: String!
  firstTotalCounter: Int!
  lastFdmDateTime: String!
  lastTotalCounter: Int!
}
```

```
input TurnoverInput {
  transactions: [EventTotalInput!]!
  departments: [DepartmentTotalInput!]!
  vats: [VatTotalInput!]!
  payments: [PaymentTotalInput!]!
  drawersOpenCount: Int! = 0
  negQuantities: [NegQuantityTotalInput!]!
  priceChanges: [PriceChangeTotalInput!]!
  invoices: [InvoiceTotalInput!]!
}
```

```
input EventTotalInput {
  eventLabel: EventLabel!
  ticketCount: Int!
  amount: Float!
}
```

```
input DepartmentTotalInput {
  departmentId: String!
  departmentName: String!
  amount: Float!
}
```

```
input VatTotalInput {
  label: VatLabel!
  rate: Float!
  taxableAmount: Float!
  vatAmount: Float!
  totalAmount: Float!
}
```

```
input PaymentTotalInput {
  id: String!
  name: String!
  type: PaymentType!
  amount: [PaymentTotalAmountInput!]!
}
```

```
input PaymentTotalAmountInput {
  type: PaymentLineType!
  normalAmount: Float!
  negativeCorrections: Float!
  positiveCorrections: Float!
  correctionsCount: Int!
  totalAmount: Float!
  normalForeign: [ForeignCurrencyInput!]
  correctionsForeign: [ForeignCurrencyInput!]
}
```

```
input NegQuantityTotalInput {
  negQuantityReason: NegQuantityReason!
  negQuantityCount: Int!
  ticketCount: Int!
  amount: Float!
}
```

```
input PriceChangeTotalInput {
  id: String!
  name: String!
  type: PriceChangeType!
  amount: [PriceChangeVatTotalInput!]!
}
```

```
input PriceChangeVatTotalInput {
  label: VatLabel!
  negative: Float!
  positive: Float!
}
```

```
input InvoiceTotalInput {
  invoiceNo: String!
  amount: Float!
}
```

```
input ReportTurnoverZInput {
  language: Language!
  vatNo: String!
  estNo: String!
  posId: String!
  posFiscalTicketNo: Int!
  posDateTime: String!
  posSwVersion: String!
  terminalId: String!
  deviceId: String!
  bookingPeriodId: String!
  bookingDate: String!
  ticketMedium: TicketMedium!
  employeeId: String!
  reportNo: Int!
  reportBookingDate: String!
```

```
posDevices: [PosDeviceInput]!  
fdmDevices: [FdmDeviceInput]!  
turnover: TurnoverInput!  
}
```

```
input ReportUserXInput {  
  language: Language!  
  vatNo: String!  
  estNo: String!  
  posId: String!  
  posFiscalTicketNo: Int!  
  posDateTime: String!  
  posSwVersion: String!  
  terminalId: String  
  deviceId: String!  
  bookingPeriodId: String!  
  bookingDate: String!  
  ticketMedium: TicketMedium!  
  employeeId: String!  
  reportBookingDate: String!  
  posDevices: [PosDeviceInput]!  
  fdmDevices: [FdmDeviceInput]!  
  users: [UserItemInput]!  
}
```

```
input UserItemInput {  
  employeeId: String!  
  totalAmount: Float!  
  firstPosDateTime: String!  
  lastPosDateTime: String!  
  socialEvents: [InOutItemInput]!  
  payments: [PaymentTotalInput]!  
}
```

```
input InOutItemInput {  
  posDateTime: String!  
  inOut: InOut!  
}
```

```
input ReportUserZInput {  
  language: Language!  
  vatNo: String!  
  estNo: String!  
  posId: String!  
  posFiscalTicketNo: Int!  
  posDateTime: String!  
  posSwVersion: String!  
  terminalId: String  
  deviceId: String!  
  bookingPeriodId: String!  
  bookingDate: String!  
  ticketMedium: TicketMedium!  
  employeeId: String!  
  reportNo: Int!  
  reportBookingDate: String!  
  posDevices: [PosDeviceInput]!  
  fdmDevices: [FdmDeviceInput]!  
  users: [UserItemInput]!  
}
```

## 2.4. DIGITAL SIGNATURE

The signature mentioned in section 4.2 is placed on the 'enriched JSON' described further. The signature is placed using the private certificate, after applying the JSON re-encoding, also described further. This ensures the reproducibility of the signature.

### 2.4.1. Canonical JSON for the Reproducible Hash

The flexibility that JSON offers allows the same information to be encoded in different ways. To always obtain the same signature for the same source data, a number of rules are applied. These rules lead to a canonical JSON.

The following rules should be strictly applied by the FDM to the fields from the JSON mentioned further in section 2.4.2.

- **Numbers** may only contain the characters 0 to 9, the minus sign (-), and a decimal point. This means, among other things, that values in scientific notation must be re-encoded to their most basic form. Trailing zeros after a decimal point are not allowed.
- All **white spaces**, as described in the JSON specifications, are removed, and no white space is added.
- Characters with a **code point > U+001F and < U+007F** are **never** escaped, with the exception of code points **U+0022** (double quotation mark) and **U+005C** (backslash), which must always be escaped according to the JSON specifications. These two must be escaped in the shortest possible notation (respectively \" and \\).
- Characters with a **code point < U+0020 or > U+007E** are **always** escaped and always in their shortest possible notation. Backspace, form feed, line feed, carriage return, and horizontal tab have such short notations (respectively \b, \f, \n, \r, and \t).
- When escaping a code point as one or more sets of 4 hexadecimal characters, the hexadecimal characters are always expressed in **uppercase** (for example: luA AFF).
- The JSON decoder must adhere to the **strict** interpretation of the JSON specifications. For example: the literals true, false, and null are always in lowercase and without quotation marks, the names of the name/value pairs in the objects must always be enclosed in quotation marks, etc.
- The **name/value pairs** of each object in JSON are **sorted** in ascending order based on the numerical value of the code points in the name. It is assumed, as a best practice, that no duplicate names are used, as no JSON standard exists in this regard.

### 2.4.2. The 'enriched JSON' whose content is digitally signed

The 'enriched' JSON object (**enrichedEventData**) consists of the values and objects below, depending on whether they were used in the message or not.

```
{  
  language  
  posId  
  vatNo  
  estNo  
  terminalId  
  deviceId  
  posDateTime  
  posFiscalTicketNo  
  ticketMedium  
  eventOperation
```

```
copyOfEvent
employeeId
customerVatNo
invoiceNo
fdmRefs
bookingPeriodId
bookingDate
costCenter
transfer
transaction
vatCalc
financials
drawer
reportNo
reportBookingDate
posDevices
fdmDevices
turnover
users
fdmSwVersion
posSwVersion
bufferCapacityUsed
verificationUrl
fdmId
fdmDateTime
eventLabel
eventCounter
totalCounter
}
```

### 2.4.3. Certificate to be used for the signature

The signature must be placed using the hardware certificate, the details of which are described in detailed description of the operation of and communication between the fiscal data module and the cloud service fod.