# IMS — Operating Concept

Every program written for real use, especially for record-keeping, rests on some underlying idea. Everything else follows from it.

The idea behind this program is **currency of data** — that the system always reflects the real, current state of things.

Which means:

1. When a user opens the system, they should see a picture as close as possible to the real state of the business.
2. Any company communicates with the world through documents — it sends something or reports something. That's what its activity consists of. Of all the documents a company might use, we only deal with the ones that change the quantity of goods/components, their location, or money (only the part tied to goods/components). Technical or process documentation, for instance, is out of scope.
3. From the "Document" concept it follows that a document in the system must appear **before** the real-world operation. If changes are recorded from the document only after the real operation has happened, the data will sooner or later drift from reality — the person doing the work no longer has any incentive to process the document accurately.
4. In small companies, operations happening inside the company are usually not tracked (not processed) — accounting doesn't need it for reporting. In our system, it's mandatory. Any change in the position or state of goods/components must be recorded as a document.
5. We assume that any person will make the right decision if they're working from correct data. If the data doesn't match reality, the decision will be wrong.
6. No record-keeping system makes life easier. It demands effort and constant discipline. It creates a lot of friction at the start. But it always leads to order and stronger control — which enables more accurate decisions.
7. The system works only through reference tables. Operators don't type free text, they only pick values from reference tables. This minimizes mistakes.
8. This approach lets an untrained person start working almost immediately. They see a document and enter it into the system — only by picking from reference tables. It's hard to get wrong.

Of course, real life is far richer and less predictable than any record-keeping system. But keeping our overarching goal in mind — making the system's state as current as possible — we allow any information in the system to be corrected, through special documents that themselves become part of the reporting trail (who corrected it, when, where, what changed to what). Correctness is verified by people.

**So:**
- Every action is a document.
- The document comes before the action.
- Only system-generated documents circulate within the company.
- Verbal orders or notes must not be used.

---

## What this actually is

IMS is an operational-control system. Not accounting, not an ERP, not a warehouse program in the usual sense. IMS keeps data current — it records what happens to goods and money: who, when, how much, from where, to

where. Everything else — reports, stock levels, balances owed — follows from the recorded operations rather than being independent data of its own.

---

## The main principle: currency of data matters more than convenience

The system doesn't block business activity, but it does demand order. If a document isn't recorded, the system loses its currency — and at that point it's useless. Therefore:

- Releasing goods without a document is **not allowed** — not because the system forbids it as such, but because the data would stop matching reality.
- Selling something that isn't in stock is **not allowed**. You can't write off what doesn't exist.
- Manufacturing from components that aren't in stock **is allowed**. The shortage is recorded but doesn't block the output. The exact behavior is left to each deployment's requirements.
- A document with a zero price is **not processed**. A zero price devalues every calculation and report downstream. This is a hard rule.

**Why the document must come before the action**: experience shows that if you ship goods first and record the document afterward, the document either never gets recorded at all, or gets recorded with errors. The person doing the work no longer cares — the goods are already gone, so there's no incentive to get the paperwork right. Hence: document in the system first, physical operation against the document second.

---

## The document — the unit of work

Everything in the system happens through documents. A document is a fact: "On June 15th, 100 units of item Z arrived at warehouse Y from supplier X at a price of 10."

**Why a document, and not individual operations**: because in reality, events come in groups. A supplier doesn't deliver one part, they deliver an invoice with 50 line items. A sale isn't one item, it's a receipt with several lines. A document ties together all the lines of a single event: one date, one supplier, one warehouse.

### Document lifecycle

1. **Creation** — the header is entered (number, date, supplier, warehouse)
2. **Filling in** — lines are added (item, quantity, price)
3. **Processing** — "Processing" is clicked. At this moment the system updates stock levels
4. **Done** — the document is processed, changes are reflected in stock

**Why processing is a separate step, not automatic**: because a document can take hours or days to fill in. While you're entering 50 lines, stock shouldn't twitch after every single one. Processing is the moment you say: "this is it, the document is ready, apply it."

Before processing, a document can be changed freely. After processing — only through a rollback.

---

## Operation types

The type defines what happened — the direction of goods or money movement:

- **Receiving** (+) — goods arrived into stock
- **Expenditure** (−) — goods left stock

- **Cash/Bank** — money movement
- **Order** — an intent (not yet goods, not yet money)
- **Manufacture** — a product was assembled from components (minus parts, plus product)
- **Correction** — a price change with no goods movement

Each type (receiving, expenditure, etc.) is only a direction. Within one type there can be any number of distinct named operations. For example: "Receiving from Supplier," "Receiving Against Debt Repayment," "Receiving Return." Any name you'll later want a report broken down by.

**Why it's set up this way**: because any business operation is one of two things — a movement of goods or a movement of money. The type defines the direction: goods arrived (+) or left (−). The operation name defines the reason.

---

## Two prices on every line

Every document line carries two prices:
- **the price BEFORE the operation**
- **the price AFTER the operation**

The actual meaning depends on the operation type. For receiving: the price AFTER is the supplier's price, the price BEFORE is the cost including extra charges. For expenditure: the price AFTER is the warehouse (cost) price, the price BEFORE is the sale price.

**Why two prices**: because buying something for 100 and getting it into stock at 100 are two different things. Shipping cost 20, customs cost 10 — the real unit cost is higher. Calculate distributes extra charges across all lines proportionally to their value.

---

## Stock — a dynamic reflection of reality

Stock is current as of "now," but can show its state as of any date in the past. The balance is the result of all processed documents — it isn't stored on its own, it's computed.

**Why**: because manually maintaining stock balances is a source of errors. If a balance is stored separately from the operations, it can drift out of sync. In our system, a balance can never be "corrupted" — it always follows from the documents. If a balance is wrong, that means some document is wrong, and that document needs to be found and fixed.

### Lot-based (batch) tracking

Every receipt creates a separate lot. If an item arrives twice at different prices, that's two lots. On a sale, goods are written off from a specific lot.

**Why by lot**: because the incoming price differs each time. A lot carries its own history: where from, when, at what price, from which supplier.

---

## Rollback — the only way to fix a processed document

A processed document cannot be edited. To fix it, you roll it back: the system undoes all the stock changes, and the document returns to "not processed" state.

**Why not just edit it directly**: because a processed document has already changed stock balances. A rollback guarantees consistency: the old changes are undone first, then the new ones are applied.

A bulk rollback lets you "roll back" the whole warehouse to a specific date — undoing every document after that date in reverse chronological order.

## Products — recipes (BOM)

A product is an item made up of other items. A TV made of parts. Borscht made of ingredients. A stool made of boards and nails.

During manufacture the system automatically creates a plus for the finished product and a minus for each component per the recipe. A shortage of components doesn't block the process — the system records what's available and what's missing.

## Orders — an intent, not an operation

An order doesn't move goods or money. It's a checkpoint: this much was ordered, this much was received, this much remains to be received.

An order can have several variants, for choosing the best composition for a budget. A component breakdown matrix shows: what's needed, how much is in stock, how much to order, what it costs.

## Reservation — a promise, not a fact

Goods are physically in stock, but marked as "promised to a specific customer." The system shows how much is really available for others.

**Why not write it off immediately**: a promise isn't a fact. The customer might change their mind.

## Reports — computed, not stored

A report is recomputed from scratch, from the operations, every time it's run. If an error in a January document is found and fixed in March, a March report covering January will show the corrected data.

A report as of a past date — the system takes the current balances and rolls back every operation after the requested date, in a temporary copy. The real data is never touched.

### User-defined queries

Large companies with regional dealers want reports broken down by arbitrary groups. A user creates a named filter: "Region 1" = dealers A, B, C. The filter can be plugged into any report. Dealers can move between regions — you only need to change the group's membership.

## Summary: why it's built this way

| Principle | Reason |
|---|---|
| Document before action | Otherwise the document is never recorded, or recorded with errors |
| Zero price forbidden | It devalues every calculation |
| Balance is computed | A separately stored balance will drift from the operations |
| Lot-based tracking | Same item, different price and history each time |
| Rollback instead of direct edit | Data consistency |
| Only through reference tables | Fewer mistakes, fast onboarding |
| Reports aren't stored | Data can be corrected after the fact |
| The system doesn't make life easier | But it leads to order and accurate decisions |

# IMS — Subsystems

The system consists of several subsystems, each responsible for its own area of work. All subsystems are connected through documents and reference tables — data entered in one subsystem is automatically available in the rest.

## Documents

The central subsystem. Everything that happens in the company involving goods and money is recorded as a document. A document has a header (number, date, counterparty, warehouse) and lines (item, quantity, price).

Document types are defined by the operations reference table. Each operation has a direction — receiving, expenditure, cash, order — and its own name. There can be several kinds of receiving alone: "Receiving from Supplier," "Receiving Return," "Receiving Free of Charge." The system distinguishes them and lets you build a report for each kind separately.

**What you can do:**
- Create documents of any type
- Add, edit, and delete lines
- Process a document — this is the moment changes are reflected in stock
- Roll back processing — to fix a mistake
- Print invoices and bills
- View a document's change history
- Attach serial numbers to items
- Reserve an item for a specific customer

Before processing, a document is a draft and can be changed freely. After processing — only through a rollback.

When creating a document, the user picks an operation from the reference table. The operation determines which fields need to be filled in: for receiving — supplier and warehouse, for expenditure — warehouse and customer, for a cash operation — bank account.

If the operations reference table has auto-numbering configured, the system assigns the document number itself and increments the counter for the next one.

## Customer orders

When a customer orders from us, an order is created. An order by itself doesn't move goods or money. It's a checkpoint: this much was ordered, this much was shipped, this much remains.

An order groups several documents under one number. For example: a customer ordered 100 units, we shipped 60 on one invoice and 40 on another — both documents are linked to the same order, and the system shows that the order has been fulfilled in full.

**What you can do:**
- See the list of active orders
- Track each order's fulfillment status
- Open the linked documents (receipts, shipments)
- Create new documents from an order

## Supplier orders (Projects)

The mirror situation — when we order from a supplier. The supplier-order form is one of the most developed in the system.

**Order lines** — exactly what's being ordered: item, quantity, price, unit of measure. Variants of the order can be saved for comparison — for example, one variant with cheap components, another with high-quality ones — and any of them can be loaded back.

**Linked documents** — which receipts have already been recorded against this order. You can see what's arrived and what's still in transit.

**Components** — automatic breakdown of the order by BOM. If 10 products are ordered and each consists of 4 parts, the system shows that 40 parts are needed. For each component you can see:
- how much is **required** by the order
- how much is **in stock**
- how much is **already ordered** from suppliers
- how much is **reserved**

This lets you pinpoint exactly what needs to be bought.

**Component matrix** — a summary report: rows are components, columns are orders. The intersection shows how much is needed. The bottom row totals each component. Indispensable when several orders sharing common parts are in progress at once.

**What you can do:**
- Create and edit order lines
- Automatically expand the BOM into components
- Refresh stock and reservation data
- Manually add or remove components
- Save and load order variants
- Print the supplier order
- Process the order

---

## Reports

89 reports (as of 2026-07-31, the count keeps growing), covering every aspect of the work. Before building, each report asks for parameters — period, warehouse, counterparty, operation — and is built fresh from current data.

**Stock reports:**
- Stock on hand — current and as of any past date
- Turnover — item movement over a period (receiving, expenditure, balance)
- Item card — full history of a single item: every operation, every document
- Inventory count — a list for reconciling against physical count
- Movement — a detailed report on transfers between warehouses

**Document reports:**
- Document registers — a list of documents for a period by operation type
- Invoices — a document's printed form
- Bills — for issuing to a customer

**Financial reports:**
- Bank operations — money movement across accounts
- Revenue — income over a period
- Balances owed — who owes us / who we owe
- Customer turnover — how much each customer bought over a period

**Order reports:**
- Order turnover — order fulfillment over a period
- Component matrix — a summary table of part requirements

**Other:**
- Employees and personnel

- Discounts and price lists
- Corrections

A report as of a past date is a special capability. The system takes current balances and mathematically rolls back every operation after the requested date. Real data is never touched. This lets you see the state of stock at any point in the past.

---

## Point of Sale

The cash-register module for retail sales. A fast interface for the cashier: find the item, add it to the receipt, complete the sale.

**What you can do:**
- Fast item search by name or SKU
- Add an item to the receipt with a quantity
- Automatic total calculation
- Apply discounts from price lists — the system automatically picks the right discount for the customer and item
- Complete the sale — a sale document is created
- Print the receipt

Discount price lists let you set different prices for different customers or customer groups. On sale, the system finds the applicable price list itself.

---

## History

View the history of changes. Who changed what and when — by document and by item.

**What you can do:**
- View the history of a specific item — every operation with it: receipts, expenditures, which documents, from which suppliers
- View the system event log

---

## Tools

Utility functions for managing data and configuring the system.

**Reservation** — mark stock as "promised" to a specific customer or order. The item stays physically in place, but the system shows how much is really available for others.

**Placement** — indicate exactly where on the warehouse the item sits: rack, shelf, cell. Helps find items quickly when shipping.

**Serial numbers** — attach a unique number to a unit of stock. On receiving, enter the number; on expenditure, pick from the ones on hand.

**Specifications (BOM)** — describe what a product is made of. A product is a list of components with quantities. Used in manufacturing and when breaking down orders into components.

**Expense distribution** — spread shipping, customs, and other extra charges across receiving lines proportionally to their value. After distribution, the item's stock price reflects the real acquisition cost.

**Extra sums** — services, taxes, and totals added to a document on top of the goods value.

**Stock viewer** — current balances for any warehouse, with search.

**Ad-hoc queries** — for power users: build a custom query against the data with arbitrary conditions and groupings.

**Event log** — a system log: which operations were performed, by whom, and when.

**Access management** — assign user permissions: who can see which sections, work with which warehouses.

**Employees** — the company personnel reference.

**Rollback** — undo document processing. A single document can be rolled back, or every document after a given date.

**Report manager** — configure report templates: which reports are available, which operation type they're tied to.

---

## Reference tables

Master data — everything documents and reports refer to. Work in the system starts with filling in reference tables.

**Main reference tables:**
- **Items** — everything the company receives and issues: parts, materials, products, services
- **Partners** — suppliers and customers. One company can be both
- **Warehouses** — physical storage locations
- **Operations** — document kinds: "Receiving from Supplier," "Retail Sale," "Transfer," etc.
- **Bank accounts** — for cash operations
- **Units of measure** — pieces, kilograms, meters, packs
- **Contracts** — linking documents to counterparty contracts
- **Payment methods** — cash, bank transfer, card, etc.
- **Specifications** — a product's recipe (what it's made of)
- **Price lists** — discount and special prices for customers

The system has over 50 reference tables in total (registry — the `sysfl` table, 52 records). All of them work the same way: search, view, add, edit. Deleted records don't disappear, they're marked inactive — so as not to break references from old documents.

---

## Security and login

**Login.** The user enters a username/password. The password is stored in the database as a bcrypt hash (not plain text) — comparison is done via `password_verify`, not a direct string match. On a successful login, the server creates a session — a random token is actually written to the database (sessions table: token → who logged in → expiry, 12 hours), not just handed to the client on trust.

**Every request to the server.** The app automatically attaches the current session's token to every action — this is done in a single, central place, not repeated per form, so every section of the app gets this the same way.

**Server-side check.** Before performing any action (creating a document, deleting, running a report — anything), the server checks: is there a valid token, does such a session exist, has it expired. No valid session — the action isn't performed at all. The exception is login itself and token verification (otherwise there'd be nothing to log in with).

**Login is mandatory.** Previously, in a test mode, the app opened the menu right away, without login. That's been removed — without login there's no session, without a session the server refuses the very first action, so login has become not just the first screen but effectively a required condition for using the app.

**What isn't checked yet.** The server currently only checks "is the user logged in at all," not "can this particular user perform this particular action." Role-based permissions are currently only enforced in the UI

(hiding unavailable buttons) — the server doesn't re-verify them. This is the next step in the system's development, not yet done.

---

### How the subsystems connect

Everything converges on documents and reference tables:

- A customer **order** generates an expenditure document when shipped
- A supplier **order** generates a receiving document when received
- **Point of Sale** creates a sale document
- **Reservation** ties to lines in stock
- A **specification** expands into order components or manufacturing operations
- **Reports** are built from documents and reference tables
- **History** shows every document for an item or a counterparty

Data is entered once and used everywhere. There's no duplication — there's a single source of truth.

---

## Orders (F3) in IMS2 — business logic for the user

Orders track **what has been ordered from us**: an external customer wants a certain quantity of items, we may not have that full quantity in stock yet, and we ship it gradually, tracking "ordered / shipped".

---

### 1. Structure

- **Order** (header) — customer, period (a start and end date), comment.
- **Line** — any item from the reference (raw material or finished product — no restriction), with fields:
- **Ordered** — how much the customer wants.
- **Shipped** — how much has actually been sent so far, accumulates as shipments happen. Starts at 0.
- **Remaining** — computed on screen as "Ordered minus Shipped", not stored separately.

---

### 2. How shipping happens

Orders does **not** ship from the Orders screen itself — it's a reference list of "what's owed". Actual shipping happens through a normal expenditure document (the "Order" button on the expenditure form) — you pick an open order line, enter the quantity to ship, and process the document as usual.

When the document is processed the system automatically:
- adds the shipped quantity to the line's shipped total;
- if the shipped total reaches what was ordered — the line **closes automatically**;
- if all lines of the order are closed — the **order closes automatically**.

No separate "close" click is needed if shipping goes through in full.

### 3. Manual close

The **Close Order** button forcibly closes an order even if something is not yet fully shipped (e.g. the customer canceled the remainder). This is a separate action from auto-close-on-shipment — used only when the order needs to be closed earlier than it's actually fulfilled.

---

### 4. Editing restrictions

- The ordered quantity of a line cannot be reduced below what's already shipped.
- A line cannot be deleted if anything has already been shipped against it.
- The whole order cannot be deleted if at least one line has shipments.

This protects against accidentally losing history — if something has already left the warehouse against a line, its data can't be erased, only closed.

---

### 5. Notifications

Closing a line/order is logged in the general event log (Tools → Events → the Events tab) — you can see there when and what closed. **These notifications do not pop up on the main-screen panel** — that panel only shows stock shortages (see `docs/en/business-logic/events_en.md`); order closures are intentionally left out to avoid distracting from the main signal. The full order history is always available in the log itself.

---

## Projects (F4) in IMS2 — business logic for the user

Projects are the mirror opposite of Orders: here **we ourselves** want to manufacture/assemble a certain quantity of products, and we track how much has already been made, plus see in advance which components will fall short.

---

### 1. Structure

- **Project** (header) — period (a start and end date), comment. Unlike Orders, there's no "customer" here — a project is internal.
- **Line** — a product (an item with a recipe/BOM, see Tools → Products), with fields:
- **Target** — how much needs to be made.
- **Made** — how much has already been made, accumulates as manufacture happens. Starts at 0.

A line can technically be added without a recipe — but then the system can't compute which components are short (see §3), and such a line is flagged with a warning ("⚠ N without BOM") in the list.

---

### 2. How manufacturing happens

Same as Orders — the Projects screen itself makes nothing, it's just a list of "what's needed". Actual manufacturing happens through the Workshop form (the "Project" button): pick an open project line, enter the quantity made, the document is processed as manufacture (consumes components per the recipe, receives the finished product).

When the document is processed:
- the made quantity is added to the line's made total;
- when the made total equals the target — the line closes automatically;
- when all lines are closed — the project closes automatically.

### 3. The "Components" tab — what's short for assembly

A separate tab inside a project computes **on the fly** (stores nothing) how much of each component is needed to make everything still remaining across all open lines:

1. Takes each product's recipe (BOM) from Products/BOM.

2. Multiplies by the line's **remaining** quantity (target minus already made).
3. Sums across all project lines — one component can be part of several different products at once, shown as a single row with the combined requirement.
4. Compares against current stock on hand — the **Deficit** column shows directly what and how much is short.

This is "what needs to be bought" reconnaissance, not a document and not a reservation — refreshed with the **Refresh** button, doesn't process or lock anything.

**Example.** A birthday party project: the target is 40 servings of Pizza Margherita. Each serving's BOM: 1 unit dough, 0.2 unit tomato sauce, 0.3 unit mozzarella cheese, 0.02 unit fresh basil. With 0 made so far and current stock of 50 dough / 10 sauce / 5 cheese / 0.3 basil, the Components tab computes the requirement for all 40 remaining servings (40 dough, 8 sauce, 12 cheese, 0.8 basil) and shows the Deficit against stock: dough 0 (covered by 50 on hand), sauce 0 (covered by 10), cheese 7 short (need 12, have 5), basil 0.5 short (need 0.8, have 0.3). This tells the buyer exactly what to order before the party, without reserving anything or blocking the project.

---

### 4. Manual close

The **Close Project** button forcibly closes a project even if something isn't finished (same as Close Order in Orders).

---

### 5. Editing restrictions

Same rules as Orders: the target can't be reduced below what's already made, a line with an already-made quantity can't be deleted, a project with at least one non-empty line can't be deleted.

---

### 6. Notifications

Same as Orders — closing a line/project is visible in the general log (Tools → Events → Events), but **does not pop up** on the main-screen panel (that panel only shows stock shortages).

### 7. Report

**Project Cost Report** — a separate button inside a project, cost based on what was actually made (not on the target) — separate from the general reports system, called directly.

---

## Events in IMS2 — business logic and user guide

For users who configure and work with tracking rules (Tools → Events). Explains how the system works at the business-logic level, not how to click buttons.

---

### 1. What this actually is

Events is a mechanism that **watches** stock and operations by itself and reports when something needs attention — no need to manually check stock levels every day. Two parts:

- **Rule** (Rules tab) — what exactly to track and how often to check. Configured once.

- **Message** (Events tab + the main-screen panel) — the result of a rule firing. Appears by itself once the condition is met.

---

## 2. What a rule can track

A rule checks one of two things:

### 2.1. Stock below a threshold (the main scenario)

You specify an item and a minimum threshold (**Min Stock Threshold**) — the system compares current stock against the threshold on every check.

- If stock falls **below** the threshold — a "shortage" message opens.
- If stock rises **above** the threshold again (a new lot arrived, stock was moved in) — the message **closes itself**, no action needed. The problem is considered resolved once the stock is sufficient again.
- Repeated messages for the same unresolved case are not spawned — while the problem persists, a single message stays open (its current stock value just gets refreshed), not a dozen identical ones.

**Warehouse** — an optional refining parameter. If not specified, the threshold is checked against the **combined stock across all warehouses**. If a specific warehouse is specified, only that one is checked.

This matters if you have several warehouses with different roles (e.g. a main purchasing warehouse and a working location/kitchen/shop floor where the needed quantity is moved to). In that case the combined total can be misleading: the item is "sufficient overall" while the place it's actually consumed from is already empty. Specify a particular warehouse if consumption happens from that warehouse specifically, not from anywhere.

### 2.2. Whether an operation happened within a period

You specify company/item/operation type/order number (any combination) — the system checks whether at least one matching operation happened during the tracked period. Used less often, to check "did we forget to make a delivery/payment."

---

## 3. One-time or recurring check

- **One-time** (Period field empty) — checked once, then the rule deactivates itself.
- **Recurring** (Period in the format `W;2;6` or `M;1;15`) — checked regularly, the next date is recalculated automatically after each check:
- `W;2;6` — weekly, on the given days of the week (1=Sunday, 2=Monday ... 6=Friday — Delphi-compatible numbering, 1-based starting on Sunday).
- `M;1;15` — monthly, on the given days of the month (the 1st and 15th).

For "stock below threshold" rules the period is effectively irrelevant — they're checked **on every login**, continuously, until closed manually (see §5). The period format is primarily meaningful for the §2.2 scenario.

---

## 4. Where the result shows up

- **The main-screen panel** (left side) — shows **only** open "stock below threshold" messages. Nothing else — not operations, not orders, so it doesn't distract from what matters most.

- **Tools → Events → the Events tab** — the full log of all messages (open and closed, both types), with company/item filters.

---

### 5. How to close a message

**There is no "dismiss" button on the main-screen panel.** This is a deliberate decision: only whoever configured the rule and understands what's happening should be able to close a problem — not any user with a single click.

The only way to close a message manually:

1. Tools → Events → the **Rules** tab.
2. Select the rule.
3. **Close Rule** button — deactivates the rule and closes all its open messages at the same time.

If the problem was real and was resolved (stock arrived, stock was moved in) — the message closes **by itself**, no manual step needed (see §2.1). Manual closing is for "false alarm" cases or when a rule is no longer relevant (e.g. an item was discontinued).

**Delete Rule** is a different action — it fully removes the rule from the reference table (not just closes it). Use it if a rule was created by mistake or will never be needed again.

---

# Taxes in IMS2 — Business Logic and Reference Setup Guide

This document is for specialists who maintain reference data (accountant, system administrator). Regular users never deal with tax setup while entering documents — everything is configured once here and then applied automatically.

---

### 1. Main rule: receiving is manual, expenditure is automatic

| Document type | How tax is determined |
|---|---|
| **Receiving** | Entered **manually** by the operator in the document's extra sums (Charges → Taxes), taken as-is from the supplier's invoice. The system never calculates or validates it. |
| **Expenditure / Sale** | Calculated **automatically** when the document is processed. The operator enters nothing. |

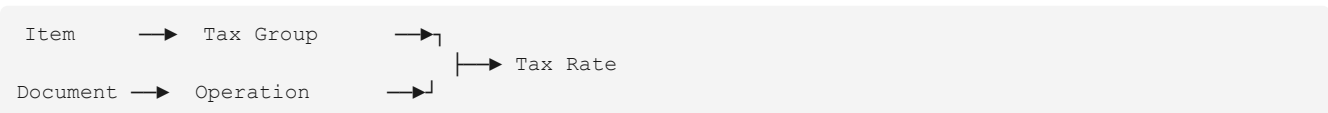
**Why**: the tax amount on a receiving document is whatever the supplier billed — it must not be changed, even if it "looks wrong" (accountants have confirmed this) — it's entered exactly as stated. On a sale/expenditure, however, computing the tax correctly is our own responsibility, so the engine handles it automatically.

If an expenditure document ends up with no calculated tax, the tax configuration for the involved item(s)/operation is simply missing in the reference data (see §3) — this is not a system bug, it's an unconfigured reference.

---

### 2. Where tax rates come from

Three references participate in the calculation:



### 2.1. Priority: operation overrides tax group

For each document line, up to 3 taxes are resolved, slot by slot (1st, 2nd, 3rd):

- If the document's **operation** has a tax code set for that slot — **it** is used.
- If the operation's slot is empty (0) — the tax code from the line's **item tax group** is used instead.

**Example**: a regular sale (operation "Sale") has no taxes configured of its own — the item's tax group is used (e.g. "Standard Goods" → 20% VAT). Operation "Export Sale" has slot 1 explicitly set to "0% Export" — an export sale then goes through with no tax **even though** the item has its normal VAT tax group.

### 2.2. The rate depends on the document date

The rate reference stores **validity periods** — when a rate changes, the old record is not deleted, it is closed with an end date, and a new record is added. Calculation always uses the rate that was in effect **on the document's date**, not the current one — so historical documents never silently move to a new rate retroactively.

---

## 3. Reading the three key tax fields

| Field | Question | Values |
|---|---|---|
| **Calc Level** | At what level is it calculated? | `0` = per line (each item line separately, then summed), `1` = per document (calculated once on the grand total) |
| **Apply Order** | In what order relative to other taxes? | integer — lower applies earlier |
| **Base** | Which amount is the rate applied to? | `0` = the original amount (ignoring other taxes), `1` = original amount **+ taxes already computed with a lower Apply Order** — tax-on-tax |

### Worked examples

**Example A — simple VAT (single rate, EU/most countries)**

Line: 100.00, tax "VAT 20%": `Calc Level=0` (per line), `Apply Order=1`, `Base=0`.

```
Step 1: VAT 20% of 100.00 = 20.00
Total tax: 20.00
```

**Example B — GST + QST (Canada, tax-on-tax)**

Line: 100.00. The item's tax group has 2 taxes configured:
- slot 1 = "GST 5%": `Apply Order=1`, `Base=0` (from the original amount)
- slot 2 = "QST 9.975%": `Apply Order=2`, `Base=1` (from the amount **including** GST)

```
Step 1: GST 5% of 100.00              = 5.00
Step 2: QST 9.975% of (100.00+5.00)=105.00 = 10.47
Total tax: 15.47
```

**Example C — per-line excise + per-document sales tax**

Document with two lines: an excisable item at 50.00 and a regular item at 30.00.
- "Excise 10%" (only on the excisable item): `Calc Level=0` (per line), `Apply Order=1`, `Base=0`
- "Sales Tax 8%" (set on the document's operation): `Calc Level=1` (per document), `Apply Order=2`, `Base=0` (calculated from the line total, no tax-on-tax needed here)

```
Step 1 (per line "Excise"): 10% of 50.00 = 5.00
Step 2 (per document "Sales Tax"): 8% of (50.00+30.00)=80.00 = 6.40
Total tax: 11.40
```

**Example D — tax-free export (operation override)**

Item with a regular tax group "Standard Goods" (VAT 20%), but the document's operation is "Export Sale" with slot 1 = "Export 0%".

```
 Operation overrides the item's tax group → "Export 0%" is used instead
Step 1: Export 0% of 100.00 = 0.00
Total tax: 0.00
```

---

## 4. Setup guide: filling in the references

Tax rates ("Taxes") and tax groups ("Category of Goods") are ordinary references, opened through **Directory (F9)** → pick the name from the list (the same universal editor used for every reference — see `docs/en/manual/24_directory.md`). The operation type is the only one of the three with its own dedicated button on the **Tools** panel, bypassing the general Directory menu (see `docs/en/manual/13_tools_overview.md`).

### 4.1. Directory → Taxes — create a new tax

1. New → fill in:
2. **Tax code** — a number you choose, must be unique among **currently active** taxes — later periods of the same tax reuse the same code, see below
3. **Name** — e.g. "VAT 20%", "GST", "Export 0%"
4. **Rate** — in % (fractional rates supported — 9.975)
5. **Calc Level / Apply Order / Base** — see §3
6. **Valid From** — the date the rate takes effect
7. **Valid To** — leave empty (this is the current active period)
8. **Accounting code** — if bookkeeping is used (optional)

### 4.2. Directory → Taxes — changing an existing tax's rate

**Do not edit the existing record directly** — doing so loses the record of what rate applied previously (breaking calculations for older documents). Instead:

1. Find the tax's current active record (**Valid To** is empty).
2. Set its **Valid To** = the day before the new rate takes effect.
3. Create a **new** record (New) with the same tax code, the new rate, a new **Valid From** (the effective date), **Valid To** left empty.

The old record stays in the reference forever — it's history, not clutter.

### 4.3. Directory → Category of Goods — attach a tax to an item group

1. Open (or create) a tax group.
2. In the three tax-slot fields click "..." — this opens the tax list, pick the one you need. If several taxes apply (tax-on-tax), fill in several slots — the order between them is determined by each tax's own **Apply Order** field (§3), not by which slot you put it in here.
3. Items are attached to a group via the item card's tax group field.

### 4.4. Tools → Operations — override tax for an operation

Use this when a specific kind of operation must **always** be taxed a certain way regardless of the item (export, tax-free services, a special rate, etc.):

1. Open the operation (e.g. "Export Sale").
2. In the three tax-slot fields, pick the tax(es) that should apply for THIS operation.
3. Leaving a slot empty (0) means that slot falls back to the line's item tax group (no override for that slot).

### 5. Verifying the calculation — "how was this tax calculated" report

For any processed document you can get a step-by-step breakdown of the calculation (no need to take the system's word for it) — the same logic as the examples in §3, but for a specific document: each step with the tax name, level, base, rate, and amount. Ask the developer / use the in-app report (F5 in the relevant section) when you need to explain to an accountant where a figure in a specific document came from.

### 6. Checklist when setting up a new tax

- [ ] There is a record in **Directory** → **Taxes** with the correct rate and **Valid From**
- [ ] **Calc Level** is set (per line / per document)
- [ ] **Apply Order** is set, if this tax participates in a chain with others
- [ ] **Base** is set, if this tax is "on top of" another tax
- [ ] The tax is attached to either an item's **Tax Group** or a document's **Operation** (or both — the operation wins)
- [ ] Verified on a test expenditure document via the "how was this tax calculated" report

## Why the app can feel slow, and how it's addressed

IMS is a build-free web app (a deliberate choice: all code is open and readable, nothing hidden behind a compiled bundle). Because of this, a screen loads not one file but dozens of separate ones — and there are a few typical reasons this can feel slow. Below is what's already been fixed, and what you can check yourself if it slows down again.

### Already fixed in this release

1. `127.0.0.1` **instead of** `localhost`**.** On some Windows machines the word `localhost` first tries to connect over IPv6, gets no response for about 2 seconds, and only then tries the regular address — and this happens on EVERY new connection, of which a single screen can make dozens. We measured this directly (see "How to check" below) — actual server work responds in 40-130 milliseconds, while the connection wait was eating up to 4 seconds. That's why all our scripts and the browser address use `http://127.0.0.1:8010`, not `localhost`.
2. **Browser cache for UI files.** Previously the browser re-downloaded every interface file on each visit. Now the server tells the browser "this file can be cached", so on a repeat visit anything unchanged is read from disk, not over the network.
3. **"Busy" cursor while waiting for a server response.** Previously, clicking a button gave no sign the app was already working — which tempted users to click again while it was still processing. Now the cursor turns into a "wait" icon for the duration of the request.
4. **Don't store the app inside OneDrive/Google Drive/Dropbox.** Folders that sync in real time noticeably slow down file access — and this app has a lot of files. Keep `IMSD` on a plain local disk.

### If it slows down again — how to check what's wrong

The most reliable approach is not to guess but to see where the time is actually going:

1. Open the app in the browser, press `F12` (opens developer tools).
2. Go to the **Network** tab.

3. Check **Preserve log**, if available.
4. Repeat the action that feels slow (open a screen, click a button).
5. Right-click the request list → **Save all as HAR** — this saves a `.har` file.

That file can be handed off for analysis — it shows exactly how much time went into connecting, how much into the server response, how much into downloading the file. That's how cause #1 above was found — without such a file it would have been just a guess.

## What we deliberately don't do (and why)

- **We don't bundle JS into a single file** (so-called "bundling"). This would speed up loading, but the code would stop being directly readable file-by-file — the whole project is built on the principle "what you see is what runs", with no build step. We consider this more important than the initial-load speed gain (especially since the browser cache already solves this on repeat visits).
- **We're not moving to Electron/Tauri** (packaging as a separate desktop app). This would give speed on par with the old Delphi version (the code sits on disk once, only data travels over the network/locally) — but that's a separate, larger effort, not a point fix. It's under consideration as a possible future direction.
- **We don't tune the app for weak/old computers.** The browser has to parse the interface code on every first visit (afterward it's served from cache) — on an old CPU this is noticeably slower, and that's a limitation of that specific computer, not of the app's settings. Requirements — see `RUN_ME.md`, "Computer requirements" section. If a computer doesn't meet them, the sensible path is to replace the computer, not to endlessly optimize the app for old hardware.